第五章

角色動作及場景的更新

在介紹完角色動作預測之後，本章將接著介紹系統如何進行角色動作及場景的更新。在這一章裡，5.1 節是運作流程，將會簡單的描述這裡的執行流程。之後的 5.2 節則是元件介紹，將介紹系統裡跟角色動作及場景更新有關的元件，包括動作選擇器、動作產生器及場景更新器。最後的 5.3、5.4、5.5 則會詳細說明我們進行角色動作及場景更新的步驟，內容包括動作樹的分析及選擇、調整可行動作樹及修正可行動作樹。

5.1 運作流程

角色動作更新的目的是要更新角色組態，讓角色的動作能符合場景的要求。這裡的運作流程主要是在分析可行動作樹裡的預測結果，並根據分析結果來選擇角色的下一個動作組態。場景更新的目的則是要根據角色動作及使用者命令來更新場景中的狀態，主要流程是更新場景並根據場景的變動內容來適時修正可行動作樹裡的預測結果。

圖 5.1是我們的系統迴圈演算法，其中的第 8 至 24 行便是主要的運作流程。在這個演算法裡，當系統進行完角色動作預測後，程式會將可行動作樹輸入到動作選擇器(Motion_Selector)中，讓動作選擇器決定角色的下個動作。如果角色有選擇下一個動作，系統便必須根據選擇結果來調整可行動作樹(Adjust_MotionTree)，否則便直接利用動作產生器(Motion_Generator)來取得角色的下個動作。之後系統會根據使用者命令來更新場景，並根據結果來決定是否必須修正動作樹裡的預測結果(Fix_MotionTree)。

```
Algorithm: System_Loop()
1.  // 初始化Feasible Motion Tree
2.  FMT* Tree = new FMT(Avatar, Environment);
3.  // 進入系統迴圈
4.  while System.Status is Not EXIT )
5.  begin
6.    // Step1: 角色動作預測
7.    Predict_Motion(FMT);
8.    // Step2: 角色動作更新
9.    Selection = Motion_Selector(FMT);
10.   if Selection is All_Collision then
11.     // 角色無法繼續下個動作
12.     return Avatar_Dead;
13.   else if Selection is Not No_Selection then
14.     // 調整可行的動作樹
15.     Adjust_MotionTree(FMT);
16.   end if
17.   // 根據選擇結果更新角色動作
18.   Motion_Generator(Avatar, Selection);
19.   // Step3: 場景更新
20.   Scene_Update(System.GetUserCommand());
21.   // 如果有非預期狀況,修正可行的動作樹.
22.   if Scene.NewCondition is Not empty then
23.     Fix_MotionTree(FMT, Scene.NewCondition);
24.   end if
25. end while
```

圖 5.1：系統迴圈的演算法

5.2 元件介紹

- **動作選擇器**

動作選擇器是用來判斷角色是否該選擇下一個動作片段的元件。在這個元件裡，我們會先判斷角色所正在進行的動作片段是否已經到達最後一個動作格。如果角色的動作格還沒到達，這個元件便直接以下一個動作格來更新角色組態，並輸出“不須要進行下個反應動作”的訊息。否則便將可行的動作樹送到動作選擇器中，讓動作選擇器分析動作樹根節點的所有子節點，從中選出一個最符合場景環境要求的子節點，將其輸出為下一個反應動作。圖 5.2是我們的動作選擇器演算法，這個演算法共有三種可能的輸出結果。第一種輸出結果為 No_Selection，代表角色目前的動作片段還沒到達最後一個動作

格，所以不須要進行下一個動作片段的反應。第二種輸出結果為 All_Collsion，代表沒有任何可行的動作片段能夠選擇。在飛彈閃躲的問題裡，這種情況相當於是飛彈已經到達角色的面前，系統已找不到任何反應動作可以讓角色閃躲過這個飛彈。第三種輸出結果則是角色所選擇的下一個動作片段的編號。

```
Algorithm: Motion_Selector(FMT)
Input. A feasible motion tree FMT.
Output. A selection of avatar.

1.  // 判斷角色目前的動作片段是否已到結尾
2.  if Avatar.NowClip.NowFrame is not END then
3.    return No_Selection;
4.  end if
5.  // 計算角色的下一個動作片段
6.  Selection = Get_NextMotion(FMT);
7.  // 沒回傳任何選擇代表角色已沒有任何活路可走.
8.  if Selection is No_Selection then
9.    return All_Collision;
10. end if
11. // 回傳所選擇的動作
12. return Selection;
```

圖 5.2：動作選擇器的演算法

- **動作產生器**

動作產生器是將角色組態轉換成角色動作動畫的元件。圖 5.3是動作產生器的示意圖，在這個元件中，當系統輸入角色組態後，它會根據組態裡的動作參數，到動作資料庫裡取出角色的動作數據。

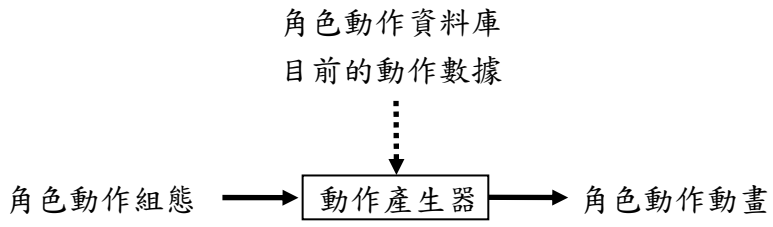


圖 5.3：動作產生器示意圖

- **場景更新器**

場景更新器的功用是根據使用者命令及角色動作來更新場景。圖 5.4是場景更新器的示意圖，在這個元件中，當系統輸入使用者命令後，它會到系統中取出目前的角色動作組態及場景資訊，並根據這些資訊來輸出新的場景資訊。

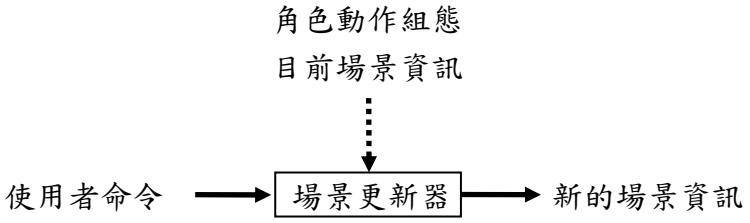


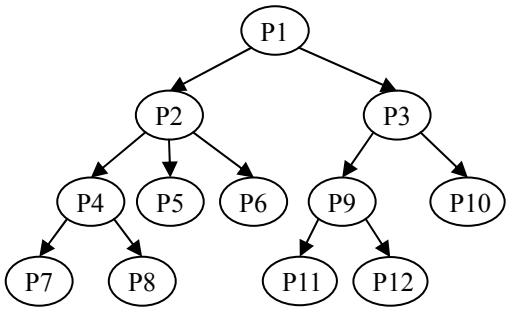
圖 5.4：場景更新器示意圖

5.3 可行動作樹的分析及選擇

如何分析動作樹根節點的所有子節點，並從中選出一個最符合場景環境要求的子節點，是動作選擇器裡最重要的一個功能。這裡我們提出了兩種分析及選擇的方式：分別是搜尋最佳路徑(best path)及搜尋最佳子樹(best subtree)。

- **搜尋最佳路徑**

搜尋最佳路徑的方式是分析可行動作樹裡的所有可能路徑，從中挑選出最適合的路徑來當選擇的依據。圖 5.5是個例子，在這個例子裡，可行動作樹有 P7、P8、P5、P6、P11、P12 及 P10 共七個葉節點(leaf node)，系統可以根據這些葉節點來找出 Path1~Path7 共七條路徑。我們會分別找出屬於這些路徑的動作樹節點，並利用這些節點裡所記錄的資訊來計算出整條路徑的品質，選擇其中品質最好的路徑所代表的子節點來當成下一個根節點。這裡計算路徑品質的方式會隨著應用的改變而有所不同，在這個範例裡，我們計算路徑品質的方式是將路徑中所有節點的擴展優先權加總起來，取其中總合值最大的路徑來當成品質最好的路徑。範例中因為 Path1 所計算出來的值最大，所以我們便選它為最佳路徑，並以此路徑中的 P2 節點為下一個根節點。



節點	節點優先權	節點	節點優先權
P1	1.0	P7	0.64
P2	0.9	P8	0.56
P3	0.8	P9	0.76
P4	0.81	P10	0.72
P5	0.63	P11	0.68
P6	0.855	P12	0.60

(a) 所要分析的可行的動作樹

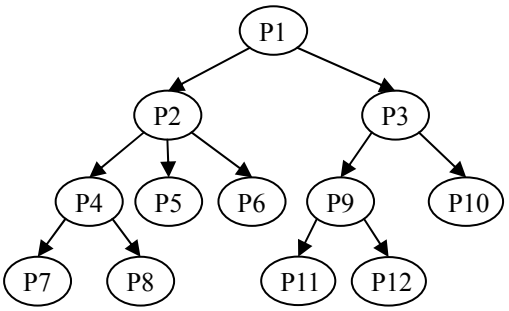
路徑	路徑所包含的節點	路徑品質
Path1	P1→P2→P4→P7	3.35
Path2	P1→P2→P4→P8	3.27
Path3	P1→P2→P5	2.53
Path4	P1→P2→P6	2.75
Path5	P1→P3→P9→P11	3.24
Path6	P1→P3→P9→P12	3.16
Path7	P1→P3→P10	2.52

(b) 分析出來的路徑及其品質

圖 5.5：搜尋最佳路徑範例：有七條路徑可以選擇，其中 Path1 為最佳路徑。

- **搜尋最佳子樹**

搜尋最佳子樹的方式是分析可行的動作樹的根節點的所有子樹，從中挑選出最適合的子樹來當選擇的依據。圖 5.6是個例子，在這個例子裡，可行的動作樹的根節點共有 Subtree1 和 Subtree2 兩個子樹。我們可以分別找出屬於這兩個子樹的所有路徑，並計算出這些路徑品質的平均值來當成比較子樹好壞的參考。在這個例子裡，因為 Subtree1 是最佳子樹，所以我們會選擇 P2 為下一個根節點。這種方式因為是以多條路徑來進行比較參考，所以即使未來動作樹會再進一步的擴展，某些路徑的品質可能會因此而變好或變壞。但在各條路徑的品質好壞有加有減的情況下，所受到的衝擊相對的會比搜尋最佳路徑的方式來得小。但它的缺點則是當最佳的路徑身處在一堆不佳的路徑裡時，這條最佳路徑所處的子樹會因為其它路徑的關係而無法被選擇到。



路徑	路徑所包含的節點	路徑品質
Path1	P1→P2→P4→P7	3.35
Path2	P1→P2→P4→P8	3.27
Path3	P1→P2→P5	2.53
Path4	P1→P2→P6	2.75
Path5	P1→P3→P9→P11	3.24
Path6	P1→P3→P9→P12	3.16
Path7	P1→P3→P10	2.52

(a) 所要分析的可行的動作樹

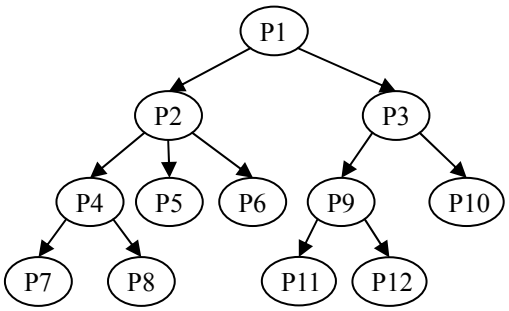
子樹	子樹所包含的路徑	子樹品質
Subtree1	Path1、Path2、Path3、Path4	2.98
Subtree2	Path5、Path6、Path7	2.97

(b) 分析出來的子樹及其品質

圖 5.6：搜尋最佳子樹：有二個子樹可以選擇，其中 Subtree1 為最佳子樹。

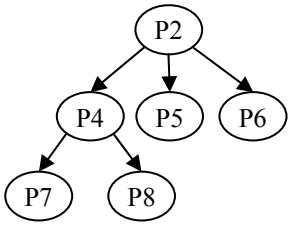
5.4 調整可行動作樹

當角色選擇了下一個反應動作後，為了符合角色目前的狀況，我們必須調整可行的動作樹裡的預測結果。調整的方法是直接以選擇的子樹來替換新的動作樹，並根據擴展優先權計算方式的不同，視情況修正動作樹節點的擴展優先權。圖 5.7是調整動作樹的範例，在這個範例裡，系統是利用目標吻合度搭配優先權老化的方式來計算節點的擴展優先權，所以系統所計算出來的擴展優先權會受到節點深度的影響而改變。圖中因為系統選了 P2 為角色的下一個動作，所以 P2 子樹就成了新的動作樹。在這個新的動作樹裡，因為 P2 變成新的根節點，所以它的擴展優先權被改成 1.0。而動作樹裡的所有節點也會因為深度的改變，而必須修正他們的擴展優先權。其中 P4 的深度原本為 2，它必須乘上的深度權重為 0.9。但在新的動作樹裡，它的深度變成了 1，所以必須乘上的深度權重也變為 0.95，因此節點的擴展優先權便從原本的 0.72 轉變成 0.76。



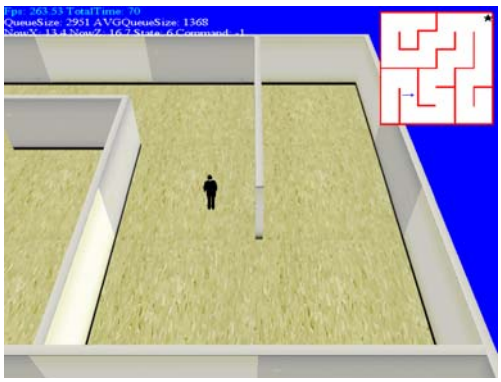
擴展優先權 = 目標吻合度 *(1-0.05x深度)

節點	擴展優先權	節點	擴展優先權
P1	1.0	P7	0.5 * 0.85 = 0.425
P2	0.9 * 0.95 = 0.855	P8	0.6 * 0.85 = 0.46
P3	0.8 * 0.95 = 0.76	P9	0.7 * 0.9 = 0.63
P4	0.8 * 0.9 = 0.72	P10	0.4 * 0.9 = 0.36
P5	0.6 * 0.9 = 0.54	P11	0.9 * 0.85 = 0.765
P6	0.5 * 0.9 = 0.45	P12	0.8 * 0.85 = 0.68

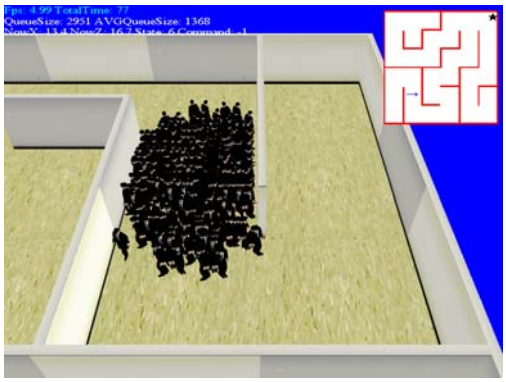


節點	擴展優先權	節點	擴展優先權
P2	1.0	P6	0.5 * 0.95 = 0.475
P4	0.8 * 0.95 = 0.76	P7	0.5 * 0.9 = 0.45
P5	0.6 * 0.95 = 0.57	P8	0.6 * 0.95 = 0.57

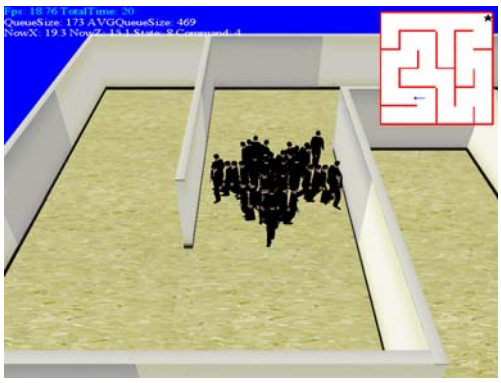
圖 5.7：調整可行動作樹的範例：角色選擇 P2 為下一個動作。



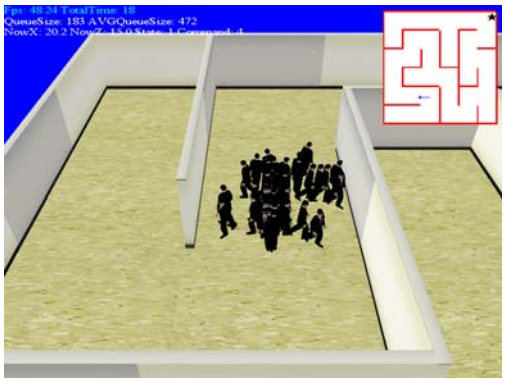
(a) 角色目前的狀態



(b) 角色目前的動作樹



(c) 角色選擇往前走之後的動作樹



(d) 角色選擇右轉後的動作樹

圖 5.8：動作樹調整的實例

5.5 修正可行動作樹

- **修正動作樹的原因**

在現實生活中，當我們在進行反應動作的思考時，我們通常很難將場景中的所有環境變動可能性給考慮進去，而是只能考慮其中某些較明顯且能夠被我們注意到的部分。其它比較不明顯的部分，只有當預測結果和現實有差距時，我們才會利用它們來進行思考結果的修正。例如在我們要穿越十字路口時，我們可能只會針對紅綠燈、路旁停止的車輛及附近的行人來進行動作的思考，進而判斷出往前走是安全的。但當我們正在往前走時，可能會突然有一輛車子闖紅燈衝了出來，這是我們當初在進行動作思考時所無法預料到的，因此我們必須根據這輛車子來修正我們之前的思考結果，進而做出後退、停下及往前快跑來閃躲車子的反應動作。

在我們的系統裡，為了模擬出以上的思考模式，我們設計了兩段式的思考流程。第一段思考流程是要根據場景中較明顯且能夠被角色注意到的環境資訊來進行思考，讓系統盡量預測出角色的未來可能動作及場景狀況。第二段思考流程則是要判斷之前的預測結果是否和場景的實際狀況有出入，並在判斷後對其進行修正，讓系統的預測結果可以符合目前場景中的實際狀況。這裡的第一段思考模式相當於我們之前在第四章所提到的角色動作預測，第二段思考模式則我們在這裡所要提到的動作樹修正。

- **修正動作樹的方法**

我們調整動作樹方法，是把原本合法的節點改成等待檢查的節點，並在之後的動作樹更新步驟裡，重新對這些等待檢查的節點進行檢查，確認這些節點是否還合法。在實作上，我們是在節點經過檢查並確認為合法後，將已經檢查過的環境資訊標註在這些節點上。之後當有新狀況被偵測到時，系統便根據這些標註來跳過已檢查的環境資訊，讓系統只根據新的狀況來檢查節點的合法性。

這裡我們用射擊遊戲的例子來說明我們的實作方法。在這個射擊遊戲裡，因為角色必須閃躲過場景中的所有飛彈，所以系統會根據飛彈和角色的碰撞情況來決定節點是否合法，如果節點所代表的角色動作會被飛彈擊中，便代表這個節點是不合法的。圖 5.9 是實作的範例步驟，其中節點裡的數字代表著節點已經檢查過的飛彈編號。例如編號為 2 的節點，代表標號值小於或等於 2 的所有飛彈都已經被檢查完畢，不須要再對它們進行重覆的檢查。在步驟 1 裡，飛彈編號的最大值為 2，其中標示為 1 的不合法節點，代表著當節點在檢查 1 號飛彈時，這個節點就已經被判斷為不合法的節點，所以不須要再檢查是否有跟 2 號飛彈產生碰撞。在步驟 2 裡，因為場景中有新的飛彈加入，所以我們將所有的合法節點都改成等待檢查的節點，之後再根據節點裡的擴展優先權來對動作樹進行擴展。其中步驟 2 到步驟 5 的節點碰撞檢查，都只有對新的 3 號飛彈進行檢查，並沒有重覆檢查 1 號飛彈和 2 號飛彈。步驟 6 的節點則是對所有的飛彈都進行了碰撞檢查，並在檢查完碰撞後，擴展出了新的等待檢查節點。

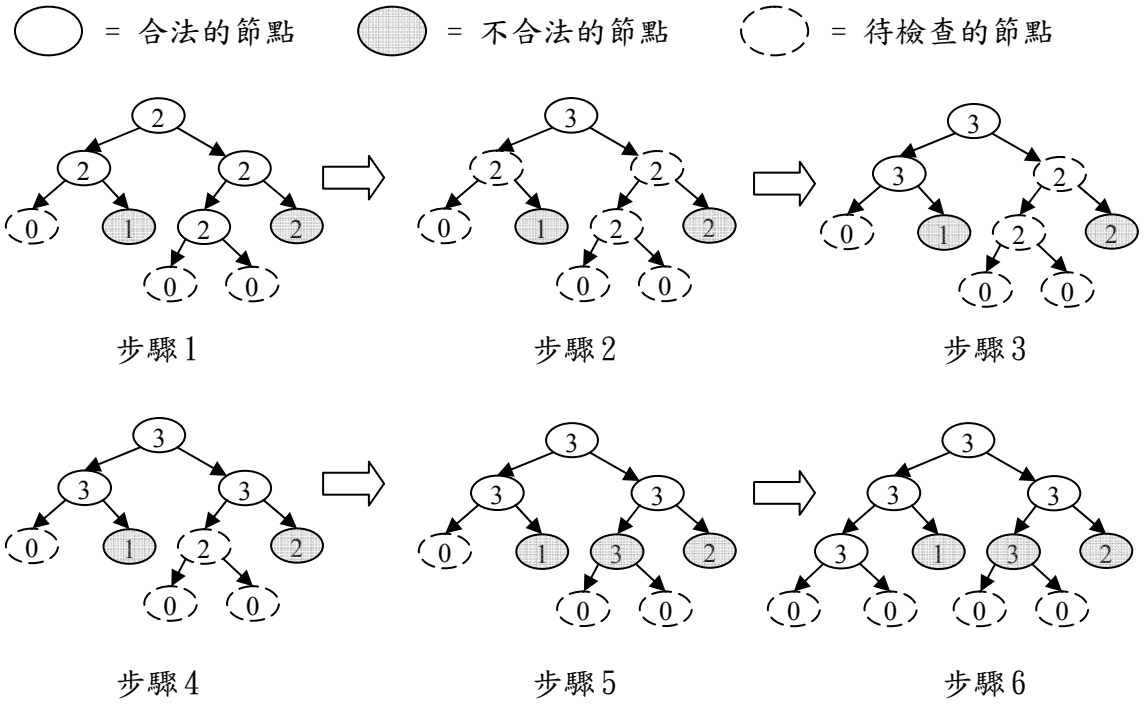


圖 5.9：修正可行動作樹的實作範例