第四章

角色動作預測

介紹完本系統的架構之後，本章將對角色動作預測的部分做更詳細的介紹。首先我們會對角色動作預測所要處理的問題做些簡單的說明。之後會詳細的介紹系統裡跟角色動作預測有關的元件，包括動作預測器及可行的動作樹。最後會說明我們如何利用這些元件來進行角色動作預測。

4.1 問題描述

角色動作預測的目的是要在角色組態空間裡，盡量搜尋出合法且可以被目前角色所達到的未來角色組態，讓系統之後可以根據這些預測結果來選擇反應的動作。但要在龐大的角色組態空間中進行搜尋，並不是一件容易的事。在有限制時間的情況下，如果只是進行盲目的搜尋，我們所能找到的組態數目可能會非常有限。所以為了增加搜尋的效率，我們會讓程式根據實際的需求來改變組態搜尋的優先順序，優先搜尋比較容易被使用到的組態，藉此減少許多不必要的計算。

圖 4.1是個簡單的動作預測例子，從圖裡可以看出角色目前正處於行進中的狀態，未來它可以選擇向前走、向左轉或向右轉的動作，角色動作預測的目的便是要判斷其中那些動作是合法且可以被目前角色達到的。在這個例子裡，因為右邊有牆壁阻擋，角色未來無法選擇右轉的動作，所以向前走和左轉便是我們這裡所希望預測出來的合法角色組態。其中如果使用者給角色的命令是往前，那麼我們將會優先搜尋向前走的動作。

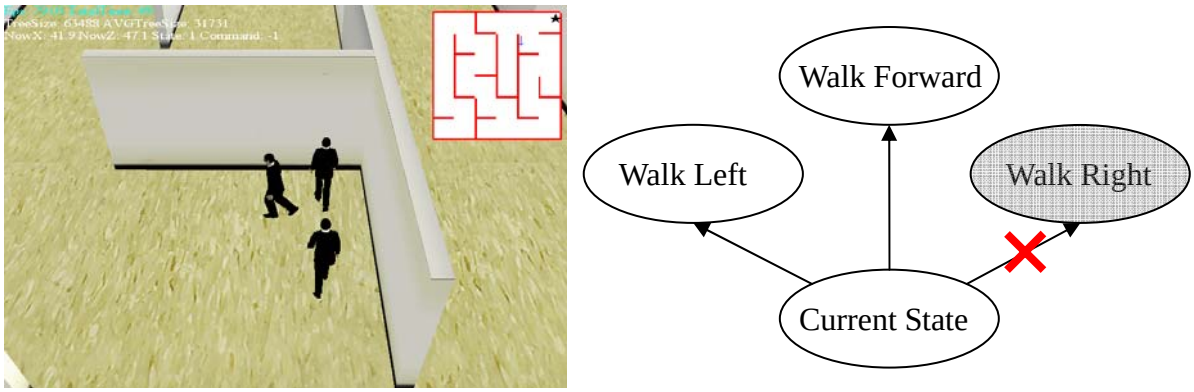


圖 4.1：角色動作預測

4.2 元件介紹

在簡單的描述了我們所要解決的問題後，接下來將對系統中角色動作預測部分的元件進行詳細的說明，包含了 4.2.1 節的動作預測器及 4.2.2 節的可行的動作樹。

4.2.1 動作預測器

動作預測器的功用是根據系統所輸入的角色組態，來推測出角色的下一個可行的組態。圖 4.2是動作預測器的示意圖，其中 Mi 是角色所正在進行的動作，我們可以根據角色的動作圖來推測出 Mi 的下一個可能動作片段，並計算出這些動作片段會對 Sj 造成的影響，藉此得到下一個角色組態的集合。

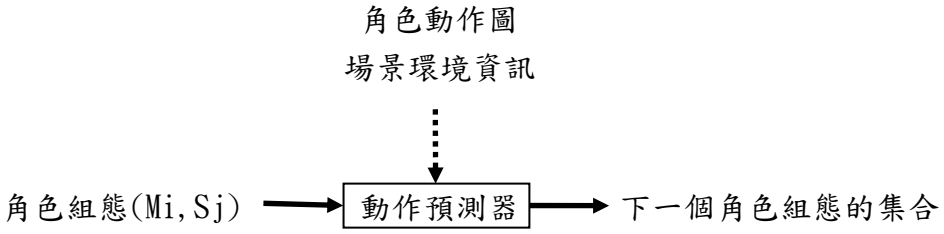
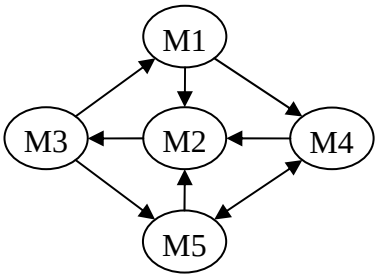


圖 4.2：動作預測器示意圖。

圖 4.3是動作預測器的執行範例，圖中的(a)是角色的動作圖，用來定義角色所有動作之間的連接關係。(b)是角色進行完這些動作時，動作會對角色及環境產生的影響，這裡以角色位置和場景時間的改變為例子。(c)是動作預測的執行範例，其中當輸入的組態

是(M1,S1)時，我們可以從(a)的動作圖裡看出角色的下一個動作可以是 M2 或 M4，從(b)的表裡可以計算出當角色進行完 M1 的動作時，它的狀態會從 S1 狀態轉變成 S2 狀態，因此它的輸出結果便是(M2,S2)和(M4,S2)，代表當角色組態為(M1,S1)時，角色的下一個組態可能為(M2,S2)或(M4,S2)。



(a) 動作圖

角色動作	動作位移	動作時間
M1	(0, 0, 2)	13
M2	(0, 2, 0)	8
M3	(0, -2, 0)	5
M4	(2, 0, 0)	10
M5	(2, 0, 2)	15

(b) 動作片段對角色狀態的影響表

輸入組態	輸出組態
(M1,S1)	(M2,S2)及(M4,S2)
(M2,S2)	(M3,S3)

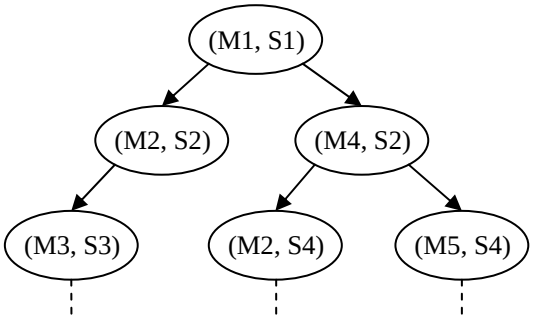
角色狀態	角色位置	系統時間
S1	(0, 0, 0)	0
S2	(0, 0, 2)	13
S3	(0, 2, 2)	21

(c) 輸入輸出範例

圖 4.3：動作預測器執行範例

4.2.2 可行動作樹

可行動作樹是我們在這篇論文裡所提出，一種用來儲存角色未來組態的樹狀資料結構。其中樹的根節點(root)代表目前的角色組態，其它節點則是根據根節點所預測出來的角色未來可能組態。圖 4.4是個簡單的例子，這個例子是以角色中心點所處的位置及當時的系統時間來表示 Sj。其中樹的根節點是(M1,S1)，代表角色所正在進行的動作為 M1，在場景裡的狀況為 S1。而從圖 4.3的動作圖裡可以看出角色的下一個動作可能是 M2 和 M4，且根據動作圖裡所定義的動作片段對角色狀態影響表，我們可以計算出當角色進行完 M1 的動作時，它在場景裡的狀況會從 S1 狀態轉變成 S2 狀態，因此根節點(M1,S1)的子節點便是(M2,S2)和(M4,S2)。之後可從節點(M2,S2)推論出(M3,S3)，或從(M4,S2)推論出(M2,S4)和(M5,S4)，以此類推下去。



角色狀態	角色位置	系統時間
S1	(0, 0, 0)	0
S2	(0, 0, 2)	13
S3	(0, 2, 2)	21
S4	(2, 0, 2)	23
S5	(2, 2, 2)	31

圖 4.4：可行的動作樹：根據圖 4.3裡的動作圖所建立的動作樹。

4.3 角色動作預測

在介紹完動作預測器及可行的動作樹後，接下來將介紹本系統如何利用這兩個元件來進行角色動作預測。我們首先會在 4.3.1 節說明整個角色動作預測的系統運作流程，之後會在 4.3.2 節和 4.3.3 節裡對流程中的可行的動作樹的擴展步驟進行說明。

4.3.1 運作流程

在角色動作預測的部分，程式首先會在系統進行初始化時，將所要預測角色的初始狀態設成可行的動作樹的根節點。之後在角色動作預測的步驟裡，利用這個根節點來擴展可行的動作樹。擴展的方法是從可行的動作樹裡取出一個優先權最高的葉節點，將這個節點輸入到動作預測器中，計算出它的子節點，並將結果回存到可行的動作樹裡。之後如果角色還有多餘的思考時間，便繼續重覆相同的步驟來繼續擴展可行的動作樹，直到角色沒有多餘的思考時間為止。

圖 4.5是我們所提出的角色動作預測演算法。其中的 Priority_Queue 是用來比較葉節點優先權的優先佇列，當這個佇列為空的時候，代表可行的動作樹裡已經沒有合法的葉節點可以被拿來繼續進行擴展了。因此便回傳 All_Collsion 訊息，並停止擴展可行的動作樹。演算法裡的 Predictor 則是前面所提到的動作預測器，用來計算輸入的葉節點的所有合法子節點。

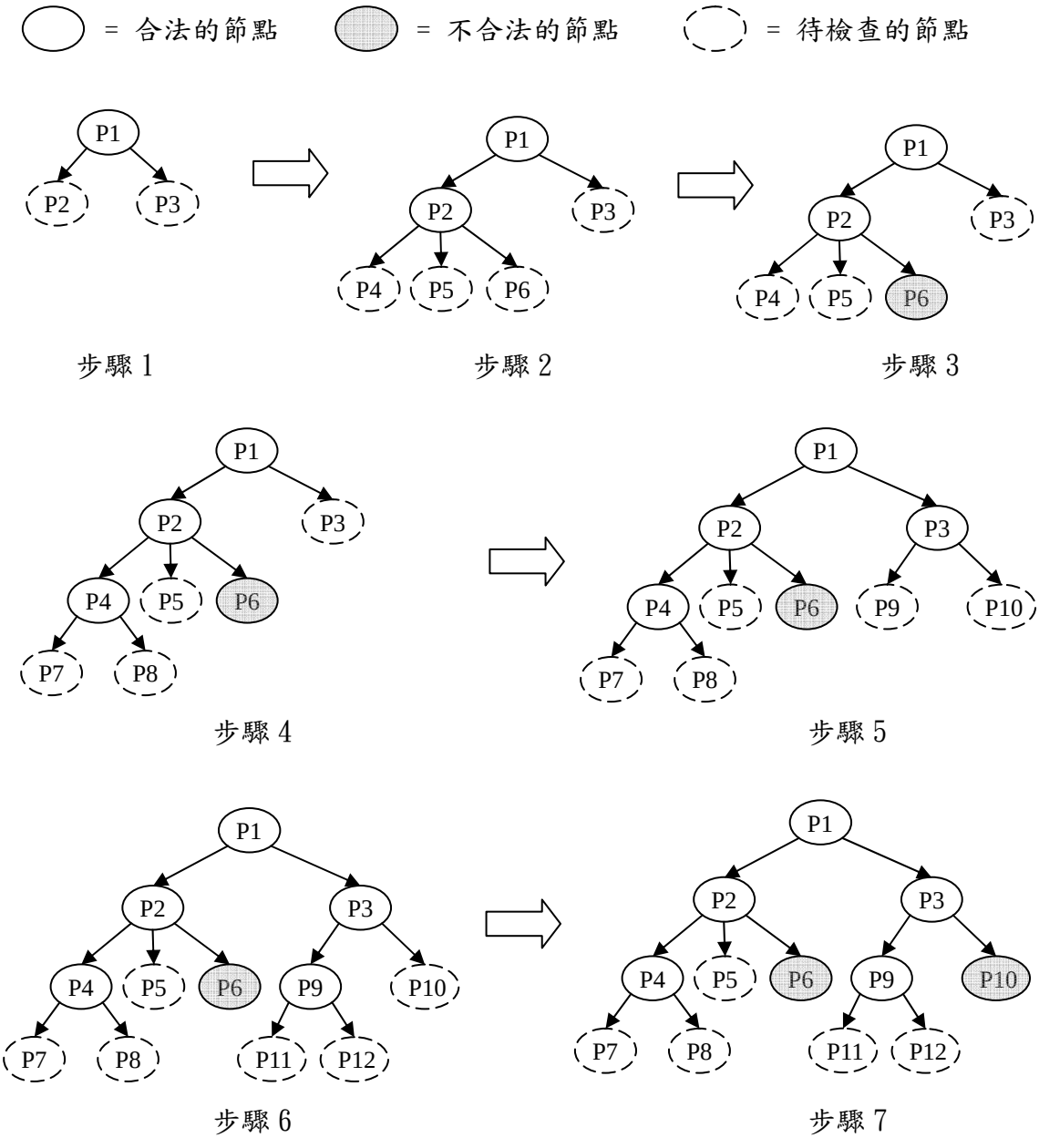
```
Algorithm: Predict_Config(FMT)

Input. A feasible motion tree FMT.
Output. Status of feasible motion tree, and updated FMT.

1.  // 進入角色思考迴圈
2.  while System.Status is IDLE
3.  begin
4.    // 檢查是否還有活路能繼續思考下去
5.    if FMT.Priority_Queue is Empty then
6.      return All_Collision;
7.    end if
8.    // 擴展可行的動作樹(FMT)
9.    s1 = FMT.Priority_Queue.pop();
10.   predict_result = Predictor.predict(s1);
11.   add all configurations in predict_result to FMT;
12    push all configurations in predict_result to FMT.Priority_Queue;
13. end begin
14. return Collision_Free;
```

圖 4.5：角色動作預測演算法

4.3.2 擴展可行動作樹的實作說明

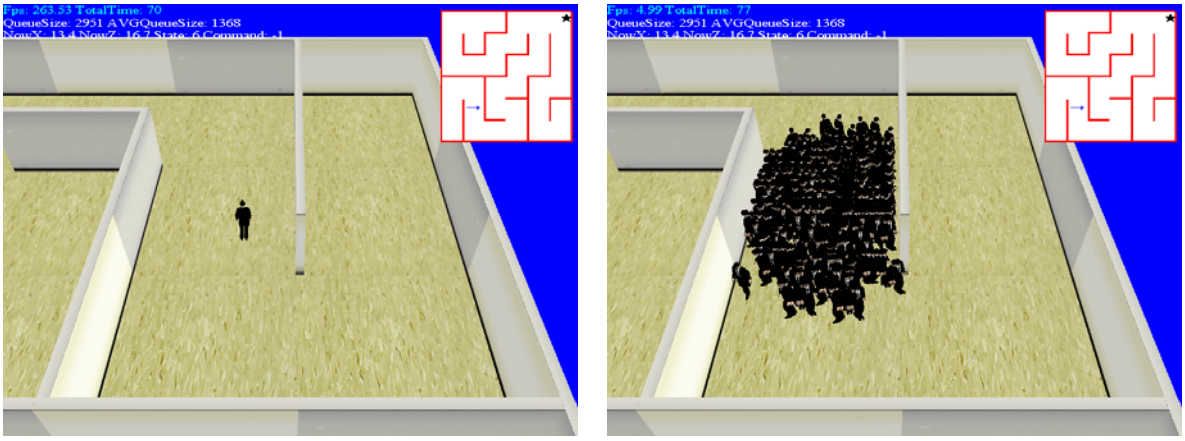
在動作樹擴展的實作裡，我們會把動作樹的節點分成三種類型。分別是合法節點、不合法節點以及等待檢查節點。合法節點是已經確認為不會跟場景產生碰撞的節點，這種節點因為未來有可能會被角色選擇到，所以它的子節點會被繼續擴展下去。不合法節點則是已確認為會跟場景產生碰撞的節點，這種節點未來不可能會被角色選擇到，所以它的子節點將不會被繼續擴展下去。待檢查節點是還沒被確認是否會跟場景產生碰撞的節點，這種節點在經過檢查後，可能會被轉變成合法或不合法的節點。

我們擴展動作樹的方法，是將角色的初始狀況設成根節點，並檢查這個節點是否為合法的節點。如果它是合法的節點，我們便根據動作圖及動作對角色的影響表，來擴展出它的子節點，並將它們設為待檢查節點。之後我們再從這些待檢查節點裡，挑選出一個最適合的節點來進行檢查，確認它是否為合法節點。如果這個節點也是合法節點，我們便繼續擴展出它的子節點，並將它們也設成待檢查節點。之後繼續從剩餘的待檢查節點裡，挑選出一個最適合的節點來進行檢查，並重覆相同的步驟下去。



節點	擴展優先權	節點	擴展優先權
P1	1.0	P7	0.648
P2	0.9	P8	0.567
P3	0.8	P9	0.76
P4	0.81	P10	0.72
P5	0.63	P11	0.684
P6	0.855	P12	0.608

圖 4.6：擴展動作樹的執行範例



(a) 角色目前的狀態　　(b) 擴展出來的動作樹

圖 4.7：動作樹的擴展實例

在擴展動作樹時，為了決定那一個待檢查節點是最適合的節點，我們會替動作樹中的所有節點都計算出一個擴展優先權(Expansion Priority)，用來代表程式擴展這個節點的優先程度。之後當系統在決定那一個待檢查節點是最適合的節點時，我們便直接根據這些待檢查節點裡的擴展優先權來進行選擇，其中數值最大的節點，便是最優先進行擴展的節點。圖 4.6是擴展動作樹的執行範例，其中 P1 是根節點，系統會根據這個根節點擴展出 P2 和 P3 節點。而經過系統計算的結果，因為 P2 的擴展優先權大於 P3 的擴展優先權，所以步驟 2 是優先擴展 P2 的子節點：P4、P5 和 P6 節點。在步驟 3，系統會檢查所有的葉節：P4、P5、P6 和 P3 節點，從中挑選出擴展優先權最高的 P6 節點。但經過檢查的結果，P6 被認定是不合法節點，所以系統不會繼續擴展它的子節點。之後在步驟 4 裡，系統會繼續檢查剩餘的待檢查節點：P4、P5 和 P3 節點，並從中挑出擴展優先權最高的 P4 節點，擴展出 P7 和 P8 節點。之後在步驟 5，繼續從剩餘待檢查節點：P7、P8、P5 和 P3 節點中挑選出 P3，並重覆相同的步驟繼續擴展下去。圖 4.7是角色在實際場景裡擴展動作樹的例子，其中左圖(a)是角色目前的狀態，代表動作樹的根節點，右圖(b)是程式根據這個根節點所擴展出來的動作樹。

4.3.3 擴展優先權的計算

擴展優先權的計算方式會隨著應用的改變而有所不同，它的設計結果往往也會直接影響到角色動作預測的品質。我們在本論文裡使用了五種計算方式來進行實驗，分別是寬度優先(breadth-first)、深度優先(depth-first)、優先權老化(aging gradually)和目標吻合度(goal-matching)。

- **寬度優先**

寬度優先是最簡單的動作樹擴展方式，它是根據節點所擴展出來的先後順序來決定節點的擴展優先程度，愈早擴展出來的節點就會有著愈高的擴展優先權。這種擴展方式有著計算量小及容易實作的優點，但因為擴展順序無法隨著目標而改變，因此有著缺乏彈性及品質不穩定的缺點。圖 4.8是個簡單的範例，其中左圖是動作樹在場景裡的擴展範例，右圖是這個範例的擴展順序示意圖，它的數字代表著節點的擴展順序。如圖所示，我們會先從根節點 1 擴展出節點 2、3、4 和 5。之再根據同深度的 2、3、4 和 5 節點，繼續擴展出 6、7、8、9、10、11、12、13 和 14 節點，接著重覆相同的步驟繼續擴展下去。從左圖的執行結果裡，我們可以看出這擴展策略的特性，是會讓動作樹幾乎圍繞著根節點來進行擴展。

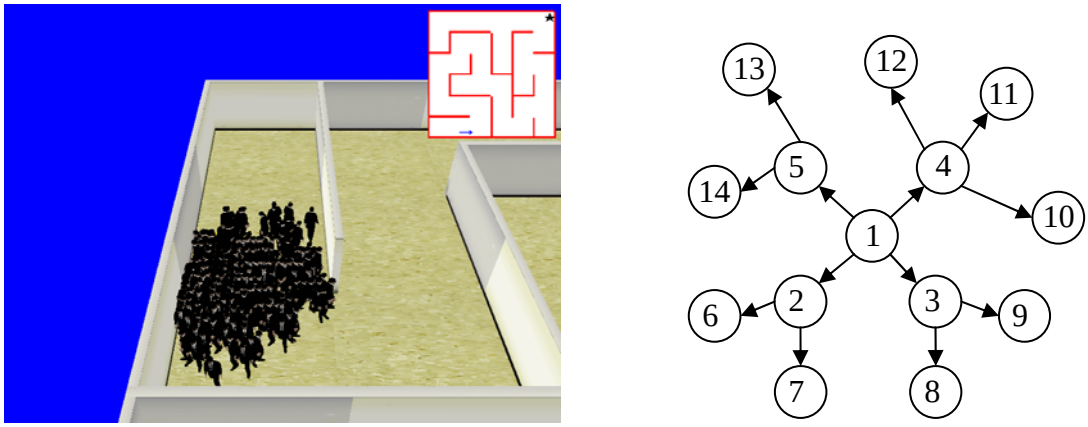


圖 4.8：寬度優先的擴展策略

- **深度優先**

深度優先是根據節點的深度來決定動作樹的擴展順序，愈深的節點就會有著愈高的擴展優先權。這種擴展方式除了優先考量的狀況不同以外，其它性質都跟寬度優先的擴展策略一樣，同樣也有著計算量小、容易實作、缺乏彈性及品質不穩定的優缺點。圖 4.9 是個簡單的範例。在這個範例裡，程式會先從根節點擴展出三個子節點，並從這些子節點裡選出最適合擴展的子節點 2。節點 2 再擴展出三個子節點，並從中選出節點 4。但因為節點 4 的子節點全部為不合法節點，所以系統會回到上一層改選擇節點 5 來進行擴展，之後繼續重覆相同的步驟擴展下去。從左圖的執行結果裡，我們可以看出這擴展策略的特性，是會讓動作樹往幾乎某個深度方向進行擴展。

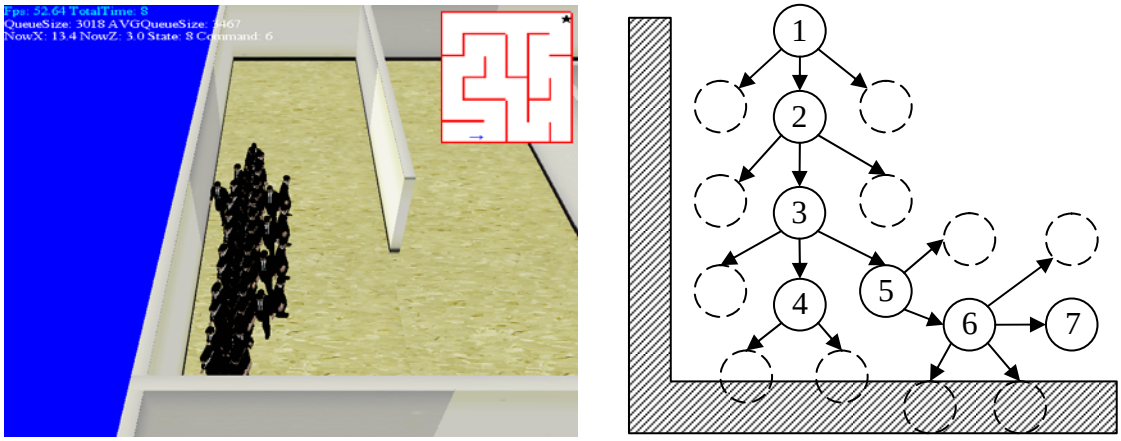


圖 4.9：深度優先的擴展策略

- **優先權老化**

優先權老化是一種跟深度優先完全相反的動作樹擴展策略，這種策略是希望角色在進行動作預測時，能夠不要思考得太遠，而是可以專注於時間上比較緊急的狀況。例如在閃避飛彈的問題裡，比起思考很遠的未來必須如何有效率的閃躲飛彈，專注於時間上比較近的未來可能反而會是比較重要的事。在這種方法裡，系統會利用其它擴展策略來

計算出節點的擴展優先權，並在之後以節點的深度為依據，將這個優先權乘上一個介於 0.0 到 1.0 之間的老化值，藉此計算出新的擴展優先權。圖 4.10是個簡單的範例。其中左圖是深度優先的動作樹擴展結果，右圖是搭配優先權老化後的擴展結果，圖中的數字代表節點的擴展優先權。

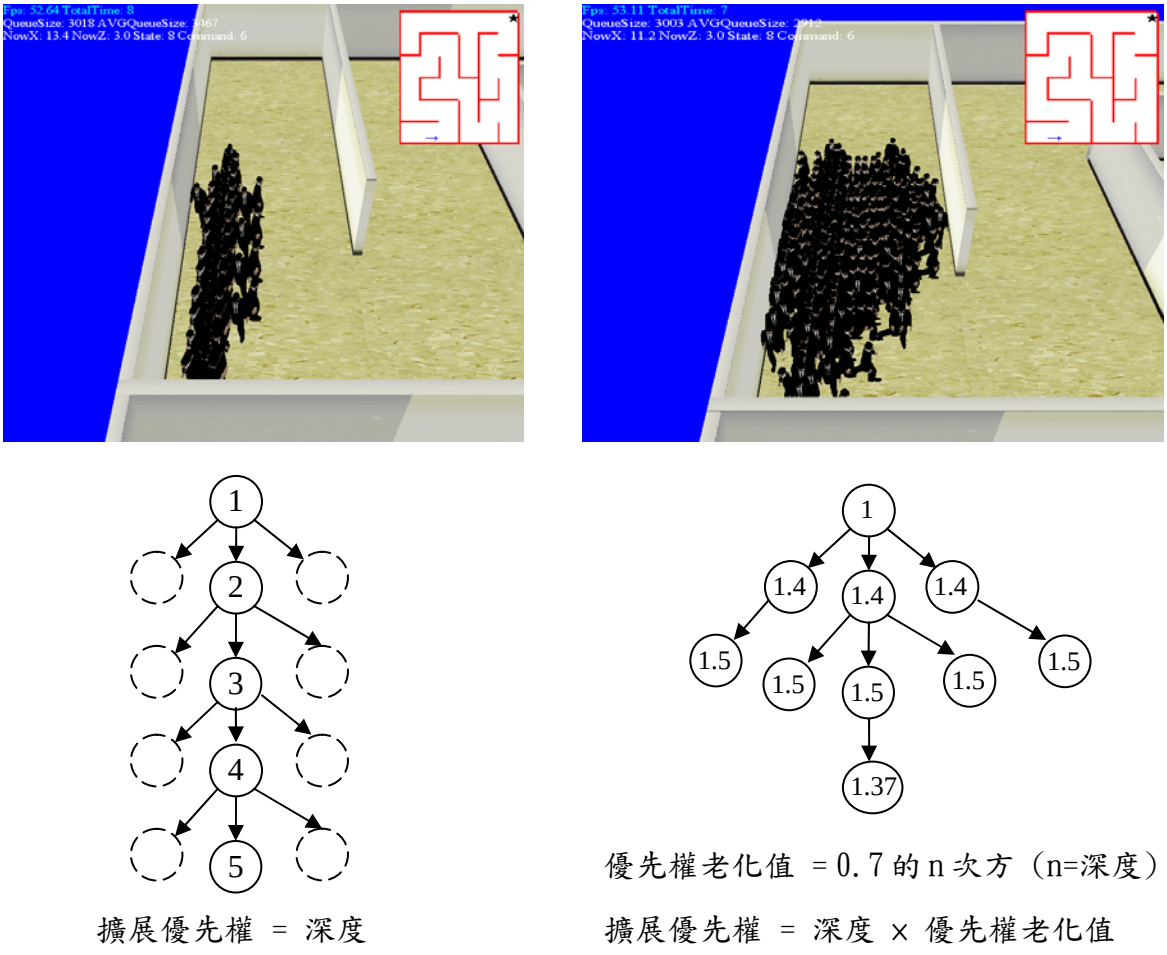


圖 4.10：優先權老化範例：(左)深度優先的擴展結果 (右)搭配優先權老化的結果

- **目標吻合度**

目標吻合度是根據節點組態與角色目標之間的差距來決定動作樹的擴展順序，愈符合角色目標的節點就會有著愈高的擴展優先權。例如在閃避子彈的問題裡，系統可能會以角色和子彈的距離來計算節點的優先權，距離愈大的節點就會有著愈高的優先權。這

種擴展方式能隨著目標來改變動作樹的擴展順序，所以有高彈性及品質穩定的優點。但因為必須設計節點組態與角色目標差距的計算函式，所以有著較不容易實作的缺點。

圖 4.11是根據使用者命令來計算目標吻合度的例子。在這個例子裡，我們希望角色能根據使用者命令的方向來進行移動，因此這裡的目標便是使用者命令所代表的方向向量 *v1*。其中 N1~N3 是可行動作樹裡的節點，Root 是代表角色目前狀態的根節點，*v2* 是角色從 Root 到 N3 的位置改變向量。在計算 N3 的擴展優先權時，系統會計算出 *v1* 和 *v2* 向量之間的夾角 *a*，取這個值的絕對值來當成目標的吻合度。它的值愈小，代表節點的狀態愈能符合使用者命令所要達到的目標，那麼節點就會有著愈高的擴展優先權。

$$\text{擴展優先權} = (\ \pi - |a|\ ) / \pi$$

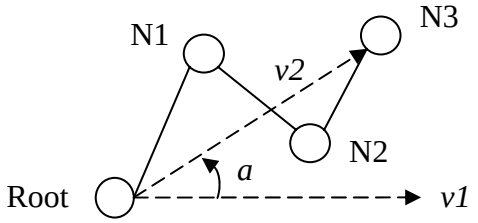


圖 4.11：目標吻合度的計算範例：根據使用者命令(*v1*)