人工智慧專題
BomberMan Agent

4108056018 資工四 王旻玄

4108056031 資工四 盧悅盛

4108056003 資工四 潘尚謙

目錄

一、 導論

動機與背景:

許多現代人會選擇以遊玩電子遊戲的方式來抒解壓力或是當做休閒娛樂，適度地從事電子遊戲可以為使用者帶來愉悅的心情，進而增進心理健康。除了放鬆之外，遊玩電子遊戲往往需要進行快速的思考，讓玩家加快在高壓情境下的決策速度，並提高決策的準確度以及增強同時追蹤多個資訊流的能力。

即時競技遊戲需要在有限的時間之下做出抉擇並用手指進行操作，處了考驗大腦的思考速度之外，手的協調性以及反射性更是重要，除此之外，與人競爭所獲得的勝利更刺激了腦內多巴胺的分泌，但是真人之間的競技往往需要雙方時間上的配合，只要缺少人就無法進行遊戲，因此我們想以 Bomberman 炸彈人遊戲為範例，打造一個具有人工智慧的電腦對手，讓玩家能與電腦進行對戰，克服真人時間上的限制，並提升玩家的遊玩水平。

文獻回顧:

基於[1]文獻，我們可以使用深度學習的技術來打造我們的電腦對手，我們將針對以下幾種方式進行說明。

Deep reinforcement learning

強化學習會讓電腦與不斷變化的環境持續互動，重複嘗試及出錯，試圖把獎賞最大化，並記錄其對應狀態和動作。再使用深度學習（如：人工神經網路…等）去近似其結果，期望最終能得到適應良好的電腦對手。

Q-learning

強化學習方法之一。用表格來儲存所有狀態對應行動的獎賞，透過不斷遞迴來找到各自最大的獎賞值，在特定條件底下，Q-learning 可以找到最佳解。

Long short-term memory (LSTM)

將時間也考慮進去，把過去所做的決定也做為決策條件之一。主要有 3 個階段：1.將較不重要的資料遺忘 2.將決策記起來，越重要就記越多 3.決定是否輸出。

Imitation learning & Behavior cloning

模仿學習和強化學習不同，並沒有獎賞的機制，而是從專家提供的範例中學習，使用的原因大多是：1.獎賞不好定義 2.人定義的獎賞可能會造成不好（或無法預期…等）的結果。

BC 就是複製專家範例中所做的行為，但可能會遇到範例沒有的問題，這時就把資料給專家詢問該怎麼做，下次訓練時就可以有新結果 (dataset aggregation)。

除了深度學習之外我們也找到了，以演算法為基底，並且依據環境進行決策的人工智慧[2]，以下將根據文獻進行探討。

State Evaluation: 用來評估未來的狀態，篩掉不好的選擇專注在好的狀態。

1. estimated_bombs(gamestate):

   預估破壞箱子所得到的分數。在模擬中，每破壞一個箱子我們加上 r^d，d 為未來炸彈會引爆的數量。

2. is_survivable(gamestate, player):

   如果玩家可以生存回傳 true，假設對手沒有放新的炸彈。用 BFS 預估未來的 8 輪。

3. can_kill(gamestate, player, enemy):

   如果對手會在兩輪內被消失回傳 true。用窮舉法計算。

4. 利用上述 1~3 我們計算未來狀態得分數，分數為下述之總和
    - 每破壞 1 個箱子 1 分
    - 0.9 * min(5, range) + 0.4 * range
    - 3.4 * min(2, extra_bombs) + 1.7 * min(4, extra_bombs) + 0.7 * extra_bombs
    - Estimated_bombs(state), r = 0.95
    - 如果剩下的箱子超過 20 則 -0.04 * distance_to_center 不然 -0.1 * average_distance_to_boxes
    - 如果玩家死亡 -1000 分

Single-Player Monte Carlo Tree Search:

為避免計算複雜度，我們假定在初始遊戲 state，對手不再改變位置而且不再放置炸彈，藉由這樣的設定我們可以大幅度的減少 children 的數量，但是 agent 無法在接下來的回合預測即將到來的炸彈和即將被炸開得箱子。為避免被敵人攻擊，用窮舉法算出未來兩輪會不會被其他炸彈炸到，如果會則不到那個位置。

根據遊戲，我們可以預測自己深度為 12 對手的深度為 9，但是太深的預測對手可能不會如預測行動因此並不可靠，故我們會限制深度。

Beam Search:

增加強以下功能已擴展 vanilla Beam Search schema

1. Zobrist hashing (ZH):
    - 用 Zobrist hashing 這個 hash function 計算每一個 state 以移除重複的 states。

2. Opponent prediction (OP):

- 我們如之前所述的方法預測敵人未來的動作

3. Local beams (LB):

- 把 states 根據玩家位置聚集在一起且限制每一個位置最多可以存 local_beam_width 最好的 states。這樣可以在保有演算法貪婪性的同時維持多樣性和提升搜尋品質。同樣的，每個層級仍限制最多儲存 beam_width states。

4. First move pruning (FMP):

首先去掉不可能存活的初始 state。此外，如果存在一個行動使得對手死亡，我們去除這個以外的。這些計算可以用 is_survivable 和 can_kill function 達成。有一些罕見情形是玩家把對手連同敵人殺死，根據規則如果玩家死亡勝者為破壞較多香子的人。當玩家可以自殺且殺死敵人時，我們計算各個玩家可以得到的分數，如果我們可以贏，我們執行這個行為。

二、 實作方法

我們使用 Unity 這個遊戲引擎來進行遊戲的製作，我們將以 3D 的方式來建置遊戲環境，並限制玩家只能夠在 X-Y 的平面上進行移動，玩家的操作方式可以利用方向鍵來進行上下左右的移動，並且利用 Space 空白鍵來放置炸彈，並且炸彈會在 3 秒之後以十字的形式進行爆炸。

我們的遊玩畫面如**圖 1** 所示，將會有玩家(紅色)，以及 AI(黃色)兩個人物，玩家一開始會在地圖的左上角，而 AI 會在地圖的右下角，地圖上的黑色正方體與長方體為牆壁，玩家不可穿越，也不可透過炸彈破壞，而橘色正方體則是道具箱，玩家不可穿越但是可以透過炸彈破壞，破壞之後有機率掉落升級道具。

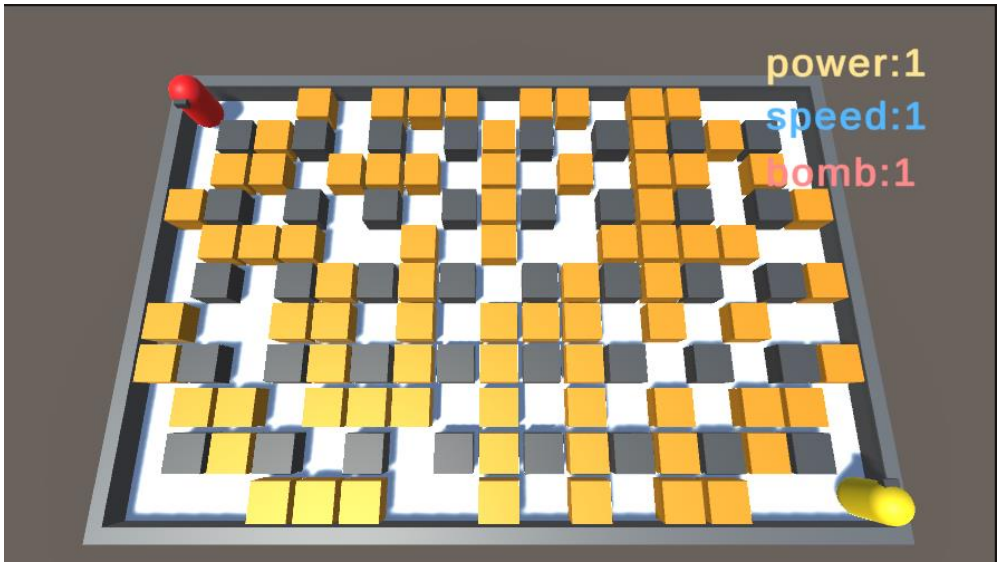

圖 1: 遊玩場景

掉落的道具可分為紅色，可用來提升放置的炸彈數量，最高可以一次放置 10 顆。藍色，提升行走速度，最高速度可以是一開始的 200%。黃色，用來提升炸彈的威力，炸彈會以十字形進行爆炸，一開始會擴散一格，最高能夠提升到擴散 10 格，掉落物如**圖 2** 所示。

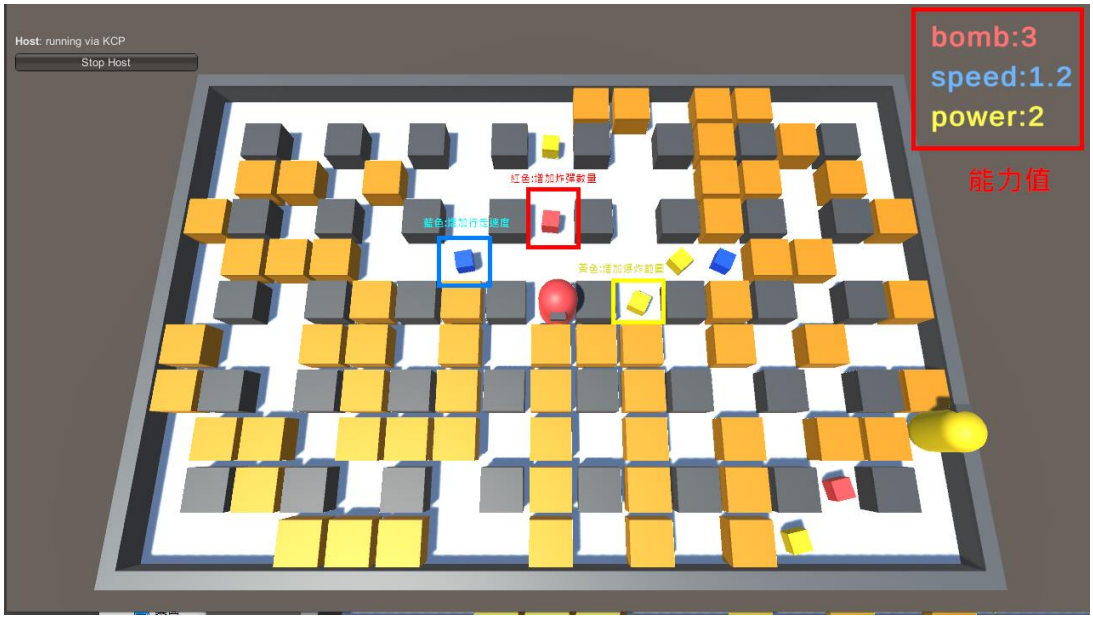

圖 2: 對應能力值與道具

炸彈本身具有碰撞體，因此玩家無法走過穿過炸彈，但是還沒離開放置炸彈的格子之前可以在該格子中進行移動，爆炸時只會炸到第一個碰撞的牆壁或是道具箱如**圖 3** 與**圖 4** 所示。

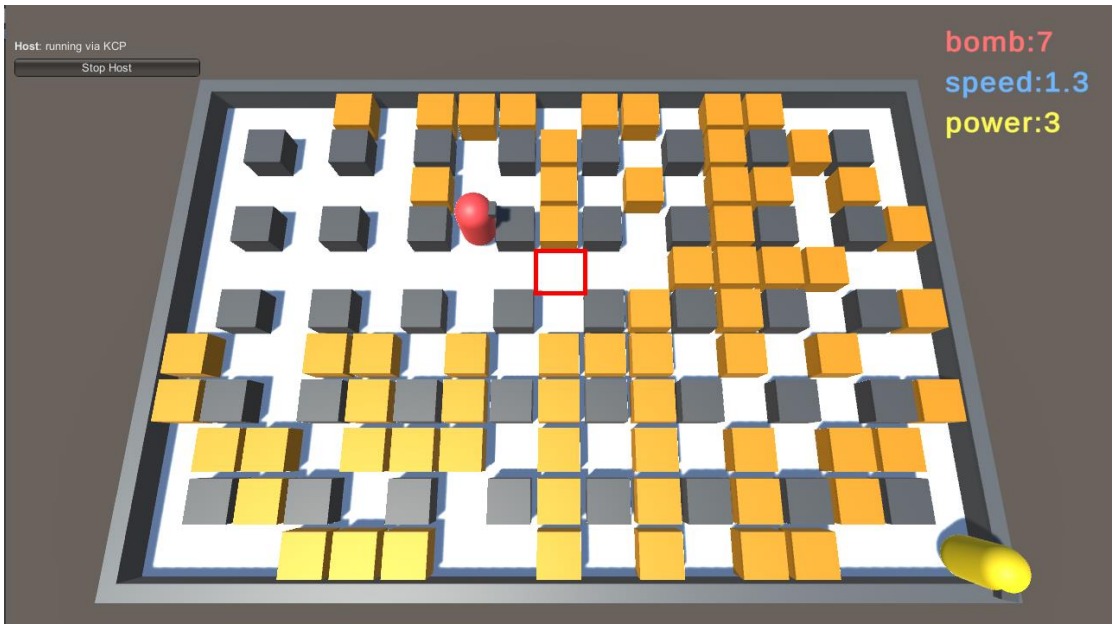

圖 3: 在該點放置炸彈

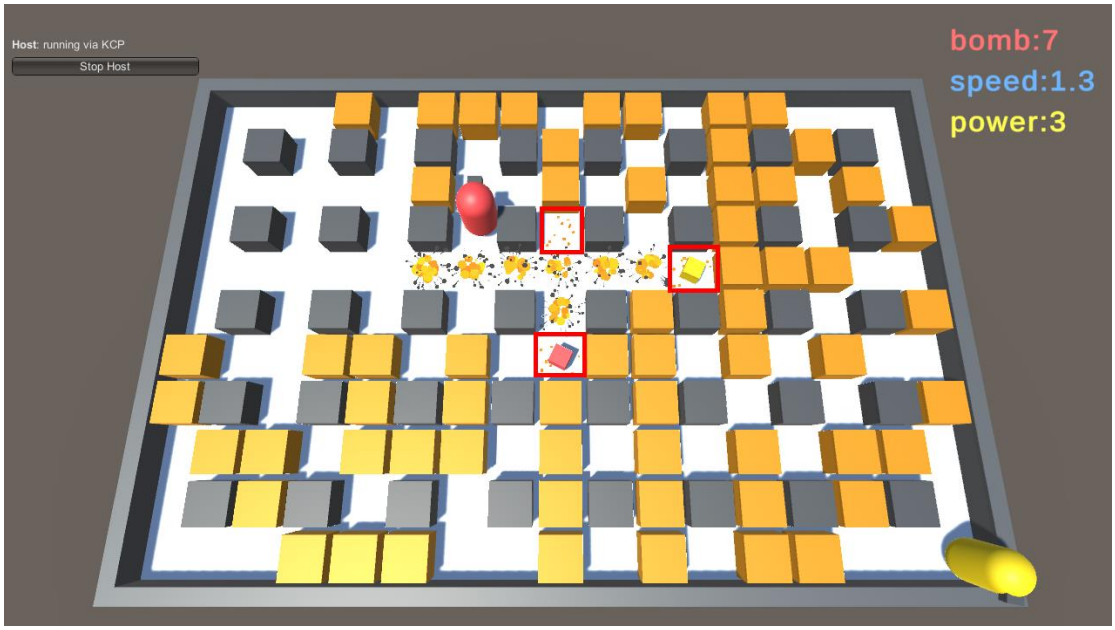

圖 4: 爆炸的範圍與掉落道具

一個炸彈的爆炸會影響另一個炸彈的爆炸，我們稱這個機制為連鎖爆炸，假設兩個炸彈的爆炸時間不一樣，其中一個比較早爆炸，且這個炸彈的爆炸範圍涵蓋另一個炸彈，那麼這兩個炸彈會同時爆炸，**圖 5** 中 3 顆炸彈的爆炸時間都不相同，越紅的炸彈代表越快要爆炸，由於左下角的炸彈會影響右下角的炸彈，而右下角的炸彈又會影響右上角的炸彈，因此結果是同時爆炸，如**圖 6**。

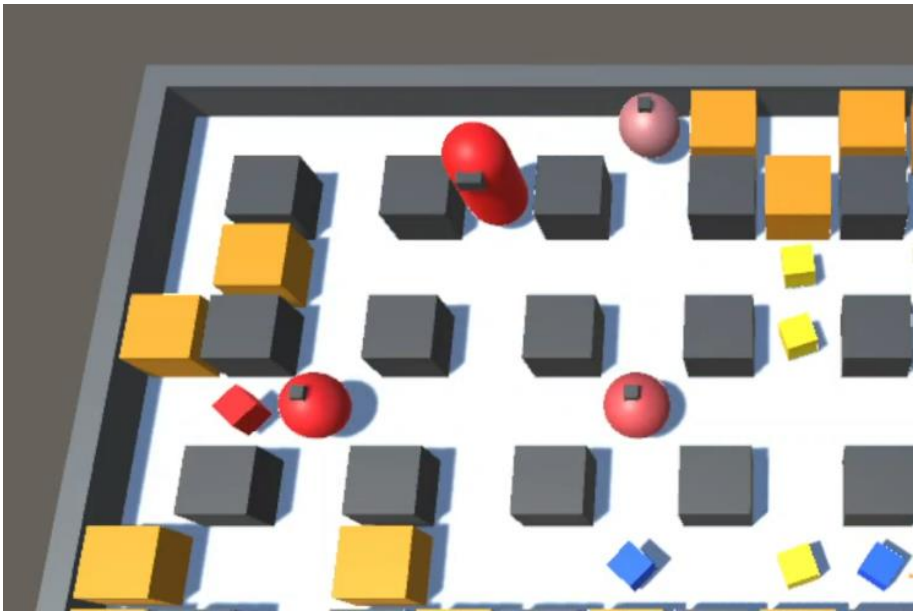

圖 5: 3 顆不同爆炸時間的炸彈

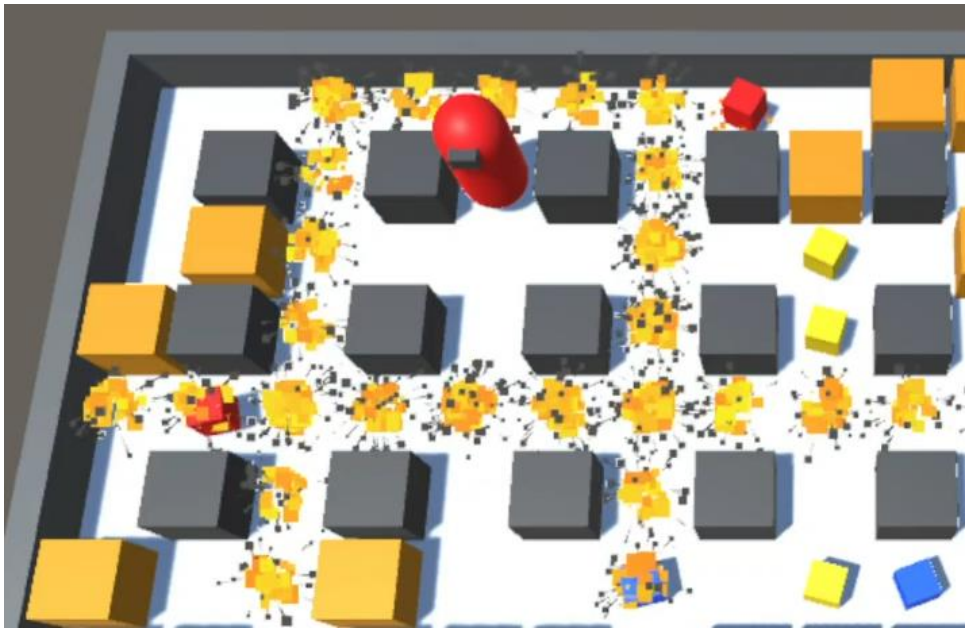

圖 6: 同時爆炸

我們開發出一個演算法能夠讓 AI 能夠尋找出安全的路徑並移動，讓 AI 盡可能的存活，以此與玩家進行對打，這個演算法分為兩個部分 Phase-1 與 Phase2。

Phase-1:

當玩家與 AI 之間還沒有連接的通路之前，我們把這個狀態稱作 Phase-1，如**圖 7**，玩家在左上角有一塊可以移動的區塊，而 AI 在右下角也有一塊可以移動的區塊，這兩個區塊並沒有通路。

在 Phase-1 時，雙方會盡可能的炸掉箱子來取得升級道具，AI 尋找一個安全的位置並炸掉箱子，之後再尋找另一條路徑，當箱子越來越少時，玩家與 AI 之間就會產生通路，進入 Phase-2。

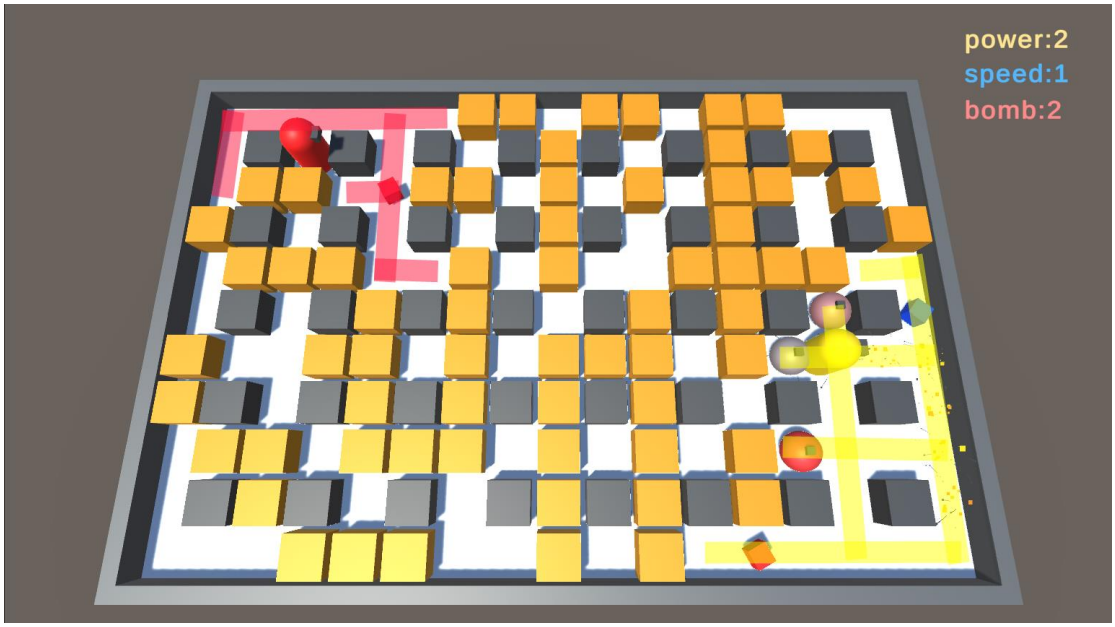

圖 7: 雙方可移動的範圍

我們將介紹 AI 如何找到一條安全的路徑，我們先來介紹一下 DangerZone()這個 Function，呼叫 DangerZone()時，輸入是現在的場景，是一個 11x17 的 2D-Array，而輸出是一個 3D-array，第一個維度是第 N 步的狀況，而後面則是 11x17 的 2d-array，代表每一步移動時的狀況。

這個 3D-array 回傳的長度，會根據 AI 的移動速度而有所變化，我們測量出來當 AI 的速度是 1.0 的時候，移動一格所需的時間是 0.23 秒，AI 可以透過吃道具來提升移動速度，最高可以到 2.0，此時移動一格所需的時間變為 0.115 秒，我們假設 AI 現在的速度是 1.5，移動一格所需的時間是(0.23/1.5) = 0.153 秒，在 3 秒內可以移動 [3/0.153] = 20 格，因此回傳的長度是 [20][11][17]，會限制在 3 秒是因為炸彈在 3 秒後會爆炸。

這個 3D-array 回傳的長度，會根據 AI 的移動速度而有所變化，我們測量出來當 AI 的速度是 1.0 的時候，移動一格所需的時間是 0.23 秒，AI 可以透過吃道具來提升移動速度，最高可以到 2.0，此時移動一格所需的時間變為 0.115 秒，我們假設 AI 現在的速度是 1.5，移動一格所需的時間是(0.23/1.5) = 0.153 秒，在 3 秒內可以移動 [3/0.153] = 20 格，因此回傳的長度是 [20][11][17]，會限制在 3 秒是因為炸彈在 3 秒後會爆炸。

我們定義陣列中的值為 0~4，意義為以下：

- 0 - 空格
- 1 - 箱子
- 2 - 牆壁
- 3 - 炸彈
- 4 - 危險

我們以**圖 8** 為目前的地圖狀態，**圖 9** 則是轉化為陣列中的值，代表 [0][11][17]，其中陣列中還會記錄這個炸彈還有幾秒鐘會爆炸。

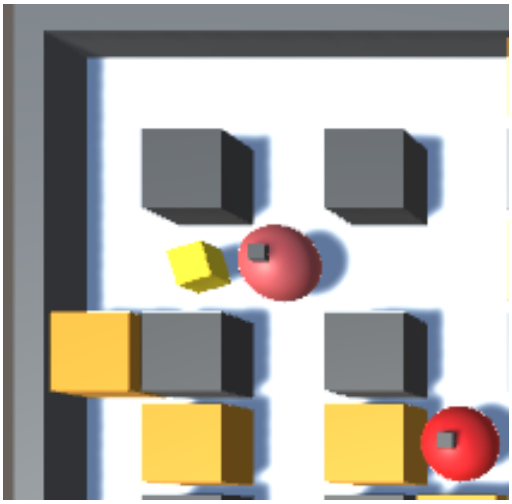

圖 8: 地圖狀態

0	0	0	0	0
0	2	0	2	0
0	0	3	0	0
1	2	0	2	0
0	1	0	1	3

1.1

0.55

圖 9: [0][11][17]陣列

我們假設 AI 移動一格的時間是 0.15 秒，那麼走 3 格後，也就是 0.45 秒之後陣列會變成**圖 10**，代表[3][11][17]的狀態。

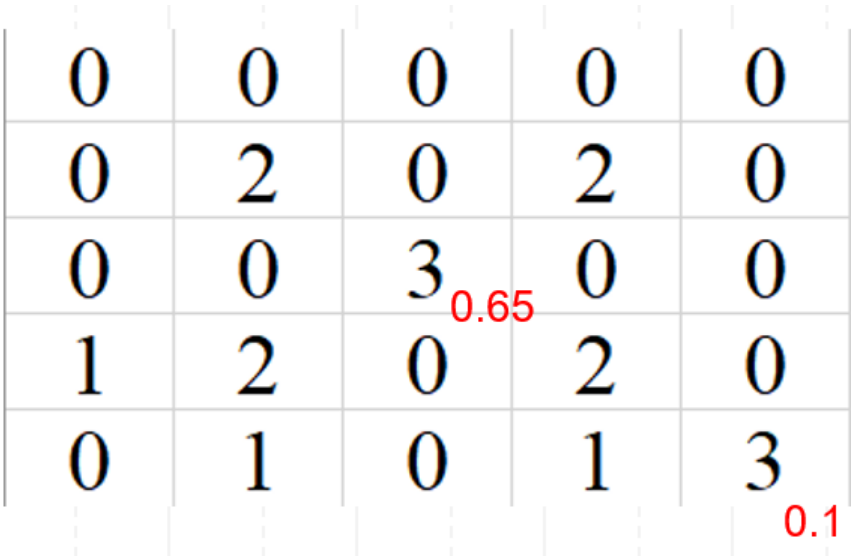

圖 10: [3][11][17]陣列

當 AI 移動到第 4 步時，右下角的炸彈爆炸，狀態變成圖 11，能看出右下角的炸彈爆炸，周圍十字延伸兩格的格子在這個時間點是不安全的。

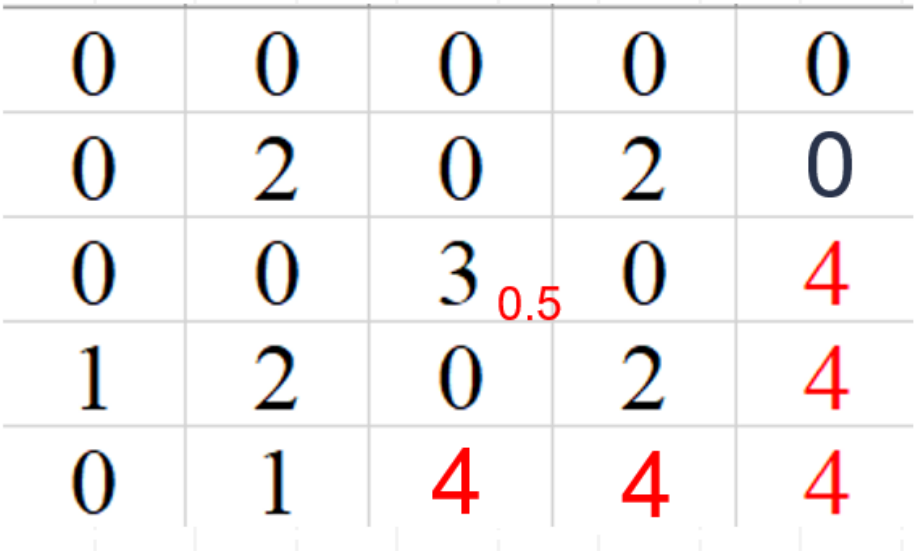

圖 11: [4][11][17]陣列

當 AI 移動到第 5 步時，右下角的爆炸結束，狀態變成**圖 12**，原本在第 4 步危險的格子因為爆炸結束，又回歸成安全狀態，而中間那顆炸彈則會在第 8 步爆炸。

0	0	0	0	0
0	2	0	2	0
0	0	3 0.35	0	0
1	2	0	2	0
0	1	0	0	0

圖 12: [5][11][17]陣列

在 Phase-1 的狀態下，AI 與玩家被隔開，因此，區域中的所有炸彈都是 AI 所放置的，也就是說所有的情況都是可以預測的，遊戲一開始 AI 會先隨機往上或往左移動到箱子旁邊，如**圖 13**。

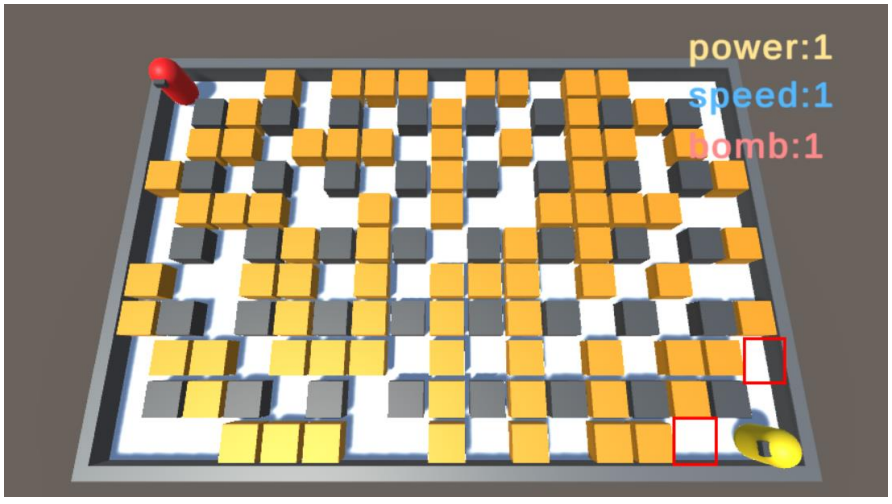

圖 13: AI 先往上或往左移動到箱子旁

之後 AI 會使用 BFS 找出最近的箱子旁邊，作為可能的目的地，移動之前會先模擬將炸彈放置在目前的位置，並利用 DangerZone()計算是否有條絕對安全的路徑到目的地，如果沒有，就會計算下一條路徑是否為絕對安全，如果有則會放置炸彈後移動，如果都沒有絕對安全的路徑或是身上沒有炸彈，則會留在原地，等炸彈爆炸後必定會出現絕對安全的路徑。

因此，AI 所停留的格子必定是安全的，而行走的路徑也絕對是安全的，所以 AI 不會在 Phase-1 被自己炸死，當停留在某一格時才會進行選擇路徑的計算。

絕對安全保證 Agent 移動到目的地之間的路徑是安全的，且停留在目的地等到 3 秒，都是安全的，能夠保證 3 秒後的未來 Agent 不會被炸死，Phase-1 的流程圖在**圖 14**。

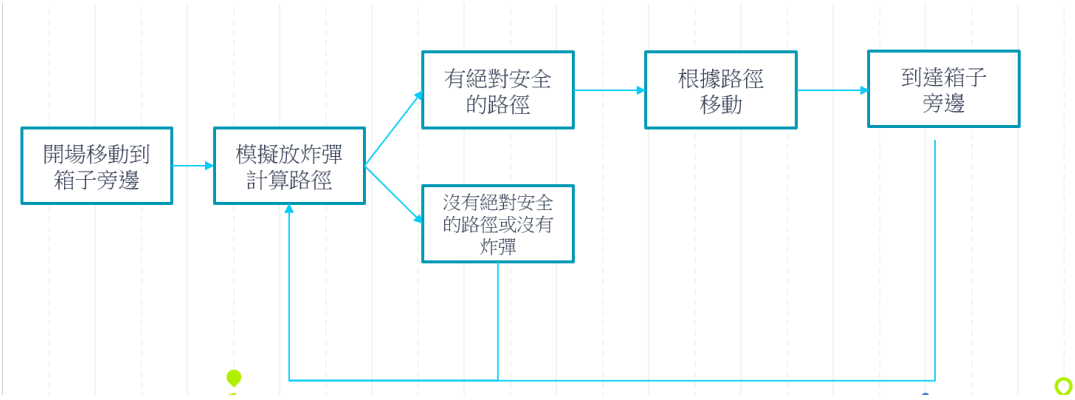

圖 14: Phase-1 流程圖

Phase-2:

當玩家與 AI 有通路存在時，則進入 Phase-2，如**圖 15**。

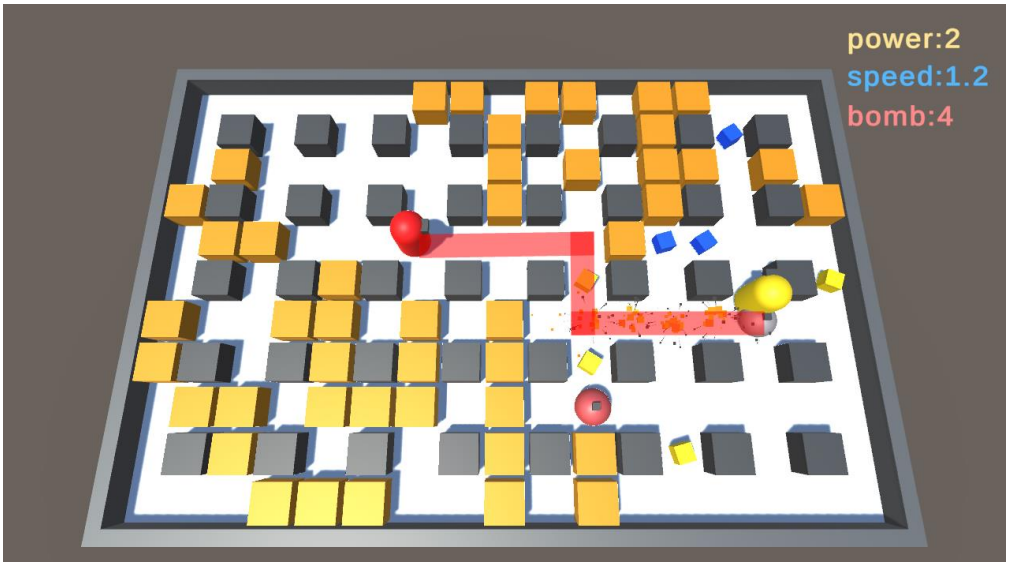

圖 15: 玩家與 AI 有路徑存在

Phase-2 的策略我們設計的較為保守，AI 會以保命為優先，會一直尋找一條安全的路徑，並在不影響該路徑的情況下去放置炸彈。AI 會先使用 BFS 來算出到所有點的最短路徑，如**圖 16**。

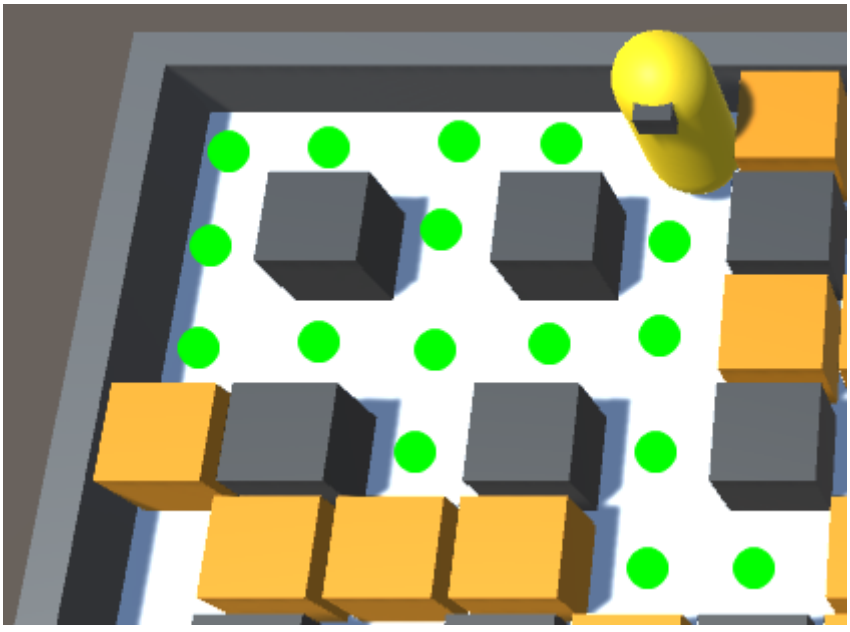

圖 16: AI 可到的所有點

再來我們會使用上述介紹的 DangerZone()來判斷這些路徑是否為相對安全，相對安全代表 AI 能夠安全的走完這條路徑，但不能夠保證走完路徑後是安全的，因為玩家也會放炸彈，因此沒有絕對安全的情況，AI 無法完全預估未來，因此 DangerZone()的判斷只會持續到 AI 到達目的地，不會將整個 3 秒都判斷完全。

之後會從相對安全的路徑中選出一條最長的路徑，並根據其長度選擇差距在 3 以內的相對安全路徑，給予 AI 隨機性，如果都沒有相對安全的路徑，AI 會先停留在原地。

當 AI 在行走路徑時，每一幀都會進行檢查，會看下一個行走的路徑是否被炸彈擋住，如**圖 17**，並且每一幀也會去呼叫 DangerZone()，看剩餘的路徑是否能夠安全的抵達目的地，如果任一情況發生，代表這條路徑是不安全的，因此會依照上面的邏輯重新計算一條相對安全的路徑。

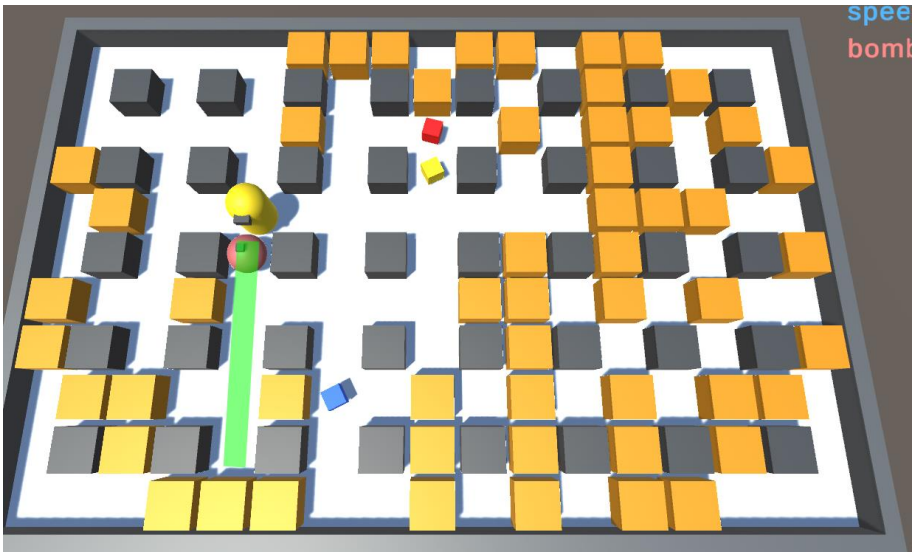

圖 17: AI 的下一步被擋住

AI 行走的路徑可分為 intermediate 與 destination，如**圖 18**，藍色的格子是 intermediate，紅色的格子是 destination。

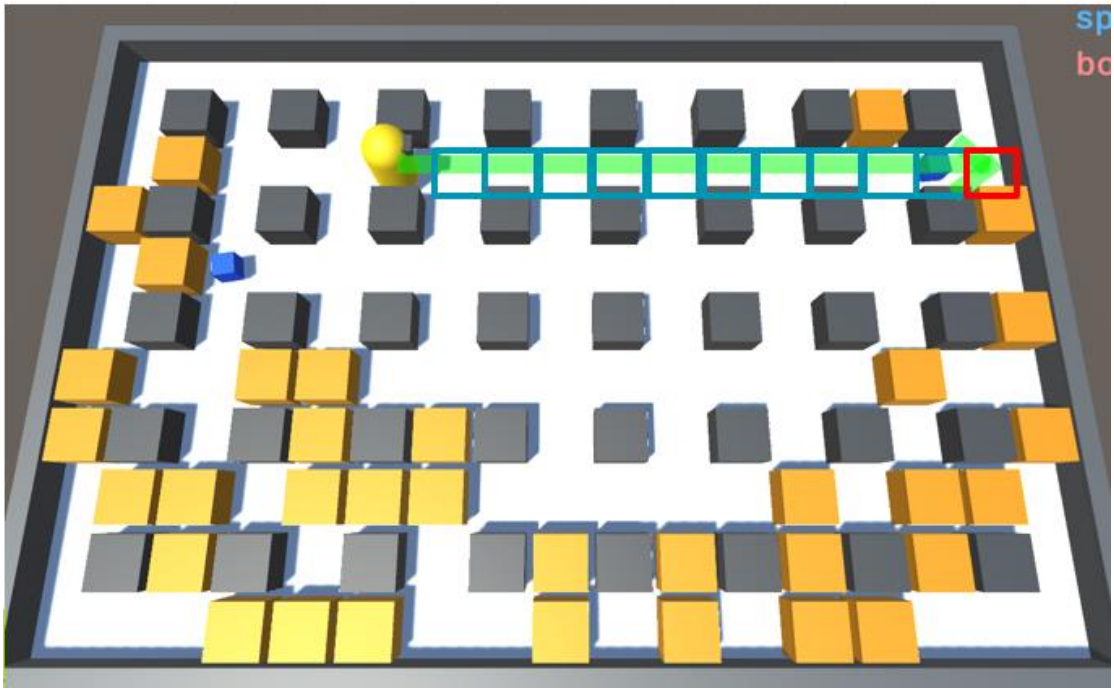

圖 18: intermediate 與 destination

當 AI 走到 destination 時，會尋找下一條相對安全的路徑，而走到 intermediate 時，AI 則可能會放置炸彈，我們設定 AI 有 50%的機率會放炸彈，但放置炸彈的前提是身上必須有炸彈，並且會利用 DangerZone()模擬放置炸彈後是否能夠安全的把剩餘路徑給走完，再來就是距離 Player 要 6 個單位以上才能放炸彈，減少卡住自己的機會。

三、 研究結果

AI 在 Phase-1 中能夠以極高的效率來炸掉箱子，並且在 Phase-2 中能夠一直找到安全的路徑來躲避玩家的攻擊，我們邀請了 5 個受試者進行 10 次的測試，**圖 19** 為測試的紀錄，人類的平均勝率是 **24%**，而 AI 的平均勝率是 **76%**。

（遊玩畫面可看簡報，執行檔附在檔案中）

我們分析人類能夠獲勝的關鍵，其中一個關鍵是強化能力值的道具要盡可能吃滿，尤其是速度，只要人類的速度夠高，就能夠運用速度優勢來卡住 AI，測試中人類若要勝利，通常遊戲時間會較久，需要一直嘗試卡住 AI。

勝率											
	第1局	第2局	第3局	第4局	第5局	第6局	第7局	第8局	第9局	第10局	勝率
測試者1	輸	輸	輸	勝	輸	輸	勝	輸	勝	輸	30%
測試者2	輸	輸	輸	輸	勝	平	輸	輸	輸	勝	20%
測試者3	輸	勝	輸	輸	輸	輸	輸	輸	輸	輸	10%
測試者4	輸	輸	輸	勝	輸	輸	輸	勝	輸	輸	20%
測試者5	輸	輸	平	勝	輸	勝	勝	輸	勝	輸	40%

圖 19：勝率紀錄

我們也讓兩個 AI 進行 100 場的互相對戰，其結果在**圖 20**，可看出其勝率接近於 50%。

	勝場	勝率
紅色	47	47%
黃色	54	53%

圖 20：AI 互打勝率

四、 結論

我們所開發的 BomberMan AI 能夠以高效率的方式去炸掉箱子，並且能夠得知哪一條路徑與哪一個點是安全的，進而移動，某些場景對於人類來說是危險的而不敢移動，但對於 AI 來說，他能夠準確判斷該點是否危險，而做出風險較高的操作。

運算複雜度方面，我們圍繞著 DangerZone()這個 Function 進行計算，他的複雜度會跟據 AI 3 秒內能走的格子數量而改變，最高會是 27x11x17，其複雜度極低，能夠有效找出安全的路徑。

目前 AI 的設計比較保守，主要都是在存活之餘嘗試去放炸彈，往後可以針對進攻方面來去進行延伸，可以偵測幾個能夠卡住玩家的 case 進行操作。

撰寫該程式的途中，比較困難的地方是演算法的設計，需要想出一個合理且運算複雜度低的方式，由於遊戲擁有隨機性，每次遊玩時掉落的強化道具都不一樣，難以去 Debug，幸好最終有將此專案做出來。

五、 參考文獻

[1] Goulart, Í., Paes, A., & Clua, E. (2019, November). Learning how to play bomberman with deep reinforcement and imitation learning. In Joint International Conference on Entertainment Computing and Serious Games (pp. 121-133). Springer, Cham.
[2] Kowalczyk, D., Kowalski, J., Obrzut, H., Maras, M., Kosakowski, S., & Miernik, R. (2022). Developing a Successful Bomberman Agent. arXiv preprint arXiv:2203.09608.

[3] Juliani, A., Berges, V. P., Teng, E., Cohen, A., Harper, J., Elion, C., ... & Lange, D. (2018). Unity: A general platform for intelligent agents. arXiv preprint arXiv:1809.02627.
[4] [Unity-ML Agent 筆記]完全從零開始的機器學習 , https://medium.com/ericzhan-publication/unity-ml-agent%E7%AD%86%E8%A8%98-%E5%AE%8C%E5%85%A8%E5%BE%9E%E9%9B%B6%E9%96%8B%E5%A7%8B%E7%9A%84%E6%A9%9F%E5%99%A8%E5%AD%B8%E7%BF%92-01-39472fccc7be, 2022/10/31
[5] Unity ML-Agents Toolkit Document, https://github.com/Unity-Technologies/ml-agents , 2022/10/31