# Group Game Project #2

Team name:前物聯網無念蓉

Group23

109550100 陳宇駿 109550085 陳欣妤 109550106 涂圓緣

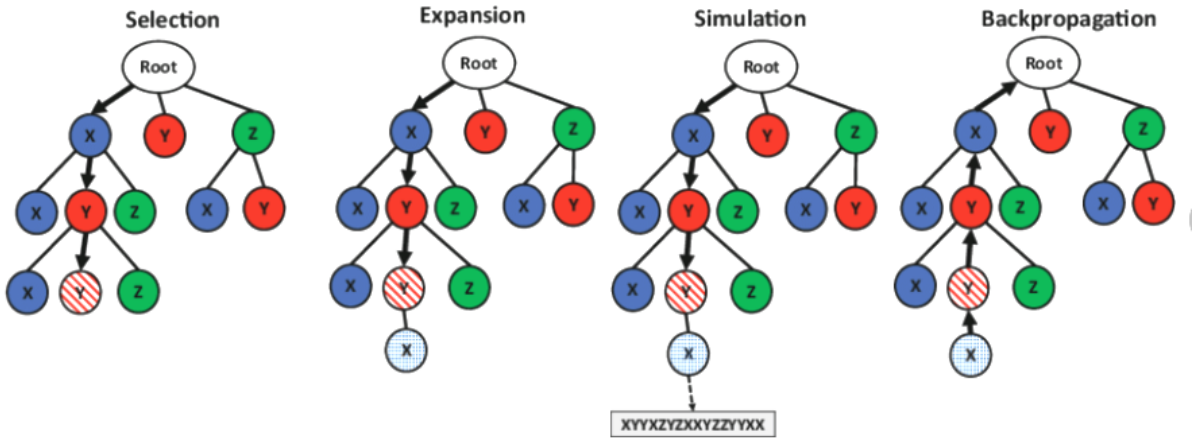
## How our game agents work

在對戰過程中，我們的agent程式負責生成指令，獲取初始地圖狀態（包括羊群位置、數量和移動方向），根據當前輪到哪位玩家以及尚未被佔用的位置，從目前狀態中擴展節點。模擬對戰時，它從生成的節點中選擇最佳的節點作為下一步的移動，並記錄當前的遊戲狀態以及整個樹狀結構（節點和模擬次數）。

我們採用MCTS進行樹搜索，將模擬對戰的計算過程以樹狀結構記錄，從根節點開始依據子節點的對戰結果將每個節點的好壞向上傳遞，以影響每個中間節點的評估。對於Battle Sheep而言，若要計算每個節點的勝率，將導致整個樹狀結構過於龐大，同時也會超出時間限制，因此，我們利用MCTS來平衡搜索範圍以及尋找具有最高勝率的節點。

## Algorithm：MCTS

主要分成 selection、expansion、simulation、backpropagation 四大步驟




fig(1). MCTS search

1. Selection: 根據Upper Confidence Bound值選擇要進行擴展的child node。
2. Expansion: 擴展節點，生成新的子節點。
3. Simulation (rollout): 從選擇到的子節點去模擬遊戲，直到遊戲結束。

4. Backpropagation: 根據模擬的結果來由下往上更新路徑上各個node的value。

以下將介紹 agent中的重要的class以及其function的作用。

- **Game Interaction**
  - **flip_board (state)**
    - 將遊戲地圖和羊的分佈狀態進行行和列的對調，創建了新的地圖狀態和羊狀態，目的是為了後續的處理和分析提供方便。
  - **flip_pos(pos)**
    - 將位置座標進行行和列對調。目的是為了後續的處理和分析提供方便。
  - **flip_action(action)**
    - 翻轉動作的方向，具體的映射關係存儲在dir_table字典中。
    - 例如：將方向2對映轉為方向4。目的是為了後續的處理和分析提供方便。
  - **possible_dir()**
    - 返回可能的移動方向清單。
  - **dir_value(dir)**
    - 返回每個方向對應的行和列的偏移量。
  - **check_valid_init(mapStat, init_pos)**
    - init_pos 使用 (x, y) 表示，先檢查mapStat的init_pos是否已經有羊存在或是此位置為障礙物，若有羊或是有障礙物則返回 False表示該位置無效。
    - 將mapStat周圍填充一層 0，確保對init_pos的窗口不超出地圖範圍，在extended_map從init_pos取 3x3 的window，檢查window 中是否有 -1，若有則返回 True表示init_pos周圍存在障礙物。
  - **on_board(x, y, board_size)**
    - 判斷座標是否在棋盤上
  - **get_init_pos(mapStat, board_size=12)**
    - surroundings是一個列表，包含棋盤格子周圍的相對座標。
    - 通過迭代棋盤上的所有可能位置，計算每個位置的有效性並找到最佳的初始位置。在迭代過程中，對於每個可能的初始位置，檢查其周圍是否有可用的位置。如果找到的有效的周圍位置數量大於

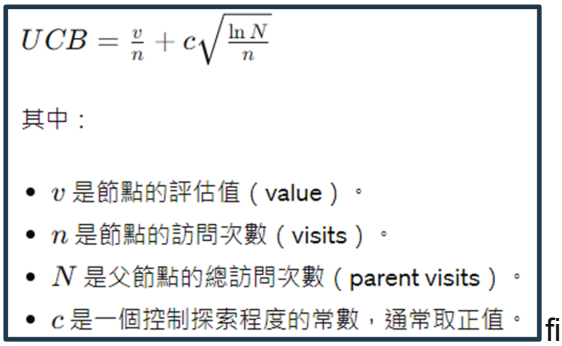
之前找到的最大數量，則更新最佳的初始位置。最終返回最佳的初始位置。

- **probe_direction(x, y, dir, mapStat)**
  - 用 dir_value 函式獲取方向，在給定的方向上不斷移動，直到遇到邊界或障礙物，停止運行並返回最後所在位置的座標。
- **apply_action(state, action)**
  - 將動作應用到當前的狀態上，並返回新的狀態。
  - 從動作中提取出位置座標 、移動的羊數量 m 和移動方向 dir，使用 probe_direction 函式來找到在給定方向上的最終位置，更新地圖狀態（將原位置的玩家ID替換為新位置的玩家ID）、羊的狀態（減去原位置的羊數量並在新位置設置新的羊數量）、玩家ID以便下一個玩家進行動作。
- **get_possible_actions(state)**
  - 獲取給定狀態下可能的動作列表，作可行的羊移動動作。
  - 從狀態中提取出玩家ID、地圖狀態和羊狀態。然後，遍歷地圖狀態，尋找所有可分割的羊群，並將這些羊群的座標添加到 splitable_sheep 列表中。
  - 對於每個可分割的羊群，它遍歷所有可能的移動方向，每個移動方向使用 probe_direction 函式來探索該方向上的最終位置，若最終位置與起始位置相同，則表示無法移動，跳過該方向，如果新的位置與原始位置不同，則將一半的羊群數量 添加到動作列表中，並返回所有可能的動作列表。
- **get_territory(mapStat)**：
  - 取得每位玩家的領地大小的地圖。
- **get_territory(mapStat)**
  - 取得每位玩家的領地大小的地圖。。
- **get_connected_regions(mapStat, playerID)**
  - 取得玩家的所有區域大小。
- **get_player_score(state)**
  - 取得玩家的得分，依照玩家所屬區域的大小計算得分。
- **get_winner(state)**
  - 獲取遊戲的贏家，根據得分最高的玩家決定贏家。

- **get_winning_team(state)**
  - 根據兩隊的總得分來決定獲勝的團隊。
- **mock_sheep_stat(state)**
  - 產生模擬的羊分配狀態，用於特定的遊戲設定，建了一個字典 sheepDict，用於存儲每個玩家的羊群數量分配情況。對於每個玩家，它將16隻羊平均分配到他們的領地上。
  - 遍歷地圖的每個格子，對於每個不是當前玩家的其他玩家的格子，從相應的 sheepDict 中取出一隻羊，並將它放置在這個格子上，使每個玩家的領地將擁有他們分配到的羊群數量。
  - 返回了一個模擬的羊群狀態 mockSheepStat，其中包含了根據模擬情況重新分佈的羊群數量。
- 其他輔助方法
  - 如計算領地(dfs)、是否為葉節點、檢查遊戲結束等。

這些方法提供了遊戲狀態的操作和處理，以及遊戲邏輯的輔助功能，例如判斷遊戲結束、計算玩家分數等。

- **MCTS**：Handles MCTS 的整個樹狀結構。
  - **select (self, node)**
    - 使用了 lambda 函數對每個子節點計算其UCB值，並選擇具有最大UCB值的子節點，這樣的設計使得算法能夠在探索和利用之間取得平衡，從而提高搜索效率和性能。以下是UCB formula，我們取2當作程式中的常數c值。



fig(2). UCB formula

    - 採用 UCB (Upper Confidence Bound) 當作 utility function，這個 formula 可以讓 MCTS 在平均較高勝率的 node 間移動，同時探索

少數可能 node之間的平衡。

- **expand (self, node)**
  - 在上一個步驟select 選出某個 node 之後，針對這個 node 去擴展它的 children，即根據當前節點的狀態，生成可能的動作並將其應用到狀態上，以創建新的子節點。
  - 作法：調用get_possible_actions獲取當前節點的狀態下所有可能的動作，對每個可能的動作用apply_action函式以將該動作應用到當前節點的狀態上，得到新的狀態，使用這個新狀態和動作來創建一個新的節點，將其添加為當前節點的子節點，最後再將新創建的子節點添加到當前節點的子節點中。
  - 策略：如果集中在進攻的話，很快就會被敵方擋住去路，反之，若只拼命的防守，可能會忽略敵方某個特別厲害的對手，因此我們會探索所有的 legal move。
- **simulate (self, state)**
  - 針對所有已經 expand 出來的 node 進行simulation，模擬我方和其他玩家輪流隨機對戰，直到分出勝負，結束後得到的結果代表這個 node 好壞的估值，類似evaluation value。
  - 方法：使用while迴圈迭代，每次先獲取當前狀態下的所有可能動作，再檢查是否有可行的動作。如果沒有可行的動作，修改當前狀態以便讓下一位玩家行動，並繼續迭代；如果有可行的動作，則隨機選擇一個動作並應用於當前狀態上，以模擬遊戲的進行。
  - 遊戲結束後，它會檢查遊戲的ID，如果是4（game_4）的話，代表進行的是隊伍比賽，程式會先確定該玩家所屬的隊伍，調用game_interaction的get_winning_team函式來獲取贏得遊戲的隊伍，並且比較該玩家所屬的隊伍與獲勝的隊伍是否相同；如果遊戲ID不是4（game_1/2/3），則代表進行的是個人比賽，程式會直接調用game_interaction的get_winner函式來獲取贏家，去比較贏家的玩家ID與該玩家的ID是否相同。
  - 根據 Monte Carlo 法的精神，隨機取樣使總模擬次數夠大，就可以

趨近真正的值。

- **backpropagate (self, node, value)**
  - 持續前面 simulate 的步驟一直到遊戲結束，並將勝負結果作為reward回傳，往 search tree 的 root node 逐一更新。
  - 將該節點的visits加1，表示此節點已被訪問；將該節點的value加上value值（這裡的value通常是模擬遊戲後獲得的結果），用於計算節點的平均值。
  - 目的：更新節點的訪問次數和值，以便後續的節點選擇和後向傳播過程能夠利用這些資訊進行合理的決策。

## Experiments and experience

- **Experiment 1**: **simulate**次數
  - 在 MCTS 中，simulate 次數影響著算法的搜尋深度和品質，simulate 次數越多，MCTS 能更好地估計每個動作的value和選擇最佳動作。透過增加 simulate 次數，可以提高 MCTS 算法的準確性和可靠性。
  - simulate 次數的增加也會增加算法的計算時間，尤其是在遊戲空間較大或複雜的情況下。因此，需要根據具體情況來平衡搜索深度和計算成本，以使算法在保證效果的情況下具有合理的計算效率。
    - **game_1/3/4的iterations為300；game_2的空間比較大，所需時間更多，故iterations 只有 100。**
- **Experiment 2: Modify parameter**
  - 目標：找到在UCB算法中，最佳參數c
  - 參數c控制了對exploitation和exploration的權衡，影響了算法在樹搜索中進行探詢的程度。較大的c值會促使算法更多地進行探索，而較小的c值則會傾向於選擇已知效果較好的節點進行利用。
  - UCB 算式左方的w/n 是 exploitation：MCTS 會根據 w/n 來選擇具有最高

平均收益的節點，以達到最大化收益的目的。

- UCB 算式算式右邊的部分是 exploration：總模擬次數 N 越大，或 node 本身被模擬次數 n 越 小，exploration 佔的比重越大。
- Exploitation 利用，我們可以利用我們所知的最佳選擇；Exploration 探索，我們冒著一些風險來收集關於未知選項的資訊。
  - 最好的長期策略可能涉及短期犧牲。
- 通常參數c的選擇是基於經驗和問題的性質而定的，如果問題的解空間很大或者有很多不確定性，就需要較大的c值來促進更多的探索。相反，如果問題的解空間相對較小或者已知的信息較多，就可以選擇較小的c值，更多地依靠利用已知信息。
  - 當常數參數**c為2時能獲得最大的 long-term reward**。

## Similarities and differences in the different game settings

- game_1與game_2 非常相似，主要區別：
  - InitPos 和 GetStep 方法中的 GameInteraction Object的Constructor參數值改為 15，這個參數是用來指定遊戲棋盤的大小。
- game_1/3/4的iterations為300；game_2的空間比較大，所需時間更多，故 iterations 只有 100。
- game_3和其他的差別主要對sheepStat的處理方式不同。
  - 在game_1/2/4版本中，GetStep()直接使用Game Server提供的 sheepStat。
  - 而在game_3中，GetStep()首先使用 GameInteraction().mock_sheep_stat 方法，傳入了 (playerID, mapStat, sheepStat) 作為參數，根據mapStat來將16隻羊平均分散在那些羊群狀態不可見的玩家的領地上，並且返回一個模擬的sheepStat。
- Game_4
  - MCTS的simulation的reward改成：當遊戲的setting ID是4的話，代表這是隊伍比賽，MCTS().simulation()會先確定該玩家所屬的隊伍自己那個

team贏了就return 1，否則return -1。

## Contributions of individual team members

**- code:** 陳宇駿 **109550100**、陳欣妤 **109550085**

**- report:** 涂圓緣 **109550106**