1 摘要

The Conference on Games (CoG) 自 2019 年由 Computational Intelligence and Games (CIG) 發展而來，是一個討論和遊戲相關一切的國際會議，參加者不乏來自學界與遊戲產業。會議內容的範圍很廣，其中遊戲 AI (AI for Playing Games) 更是熱門的主題、競賽之一。

Legends of Code and Magic (LoCM) 爲 CoG 其中一款遊戲，其玩家爲程式撰寫的虛擬玩家。它是一款簡化版的爐石戰記 (一款卡牌遊戲)。規則爲一對一的回合制，玩家的目標是從卡池中組成牌組，並透過牌來攻擊對方。只要將對方的血量扣光，即獲得勝利。

由於牌的能力豐富且多樣，有些也會有克制關係。組牌時要考慮多方因素，玩牌時要依據情境操作，所以要找出遊戲的最佳策略並非易事。

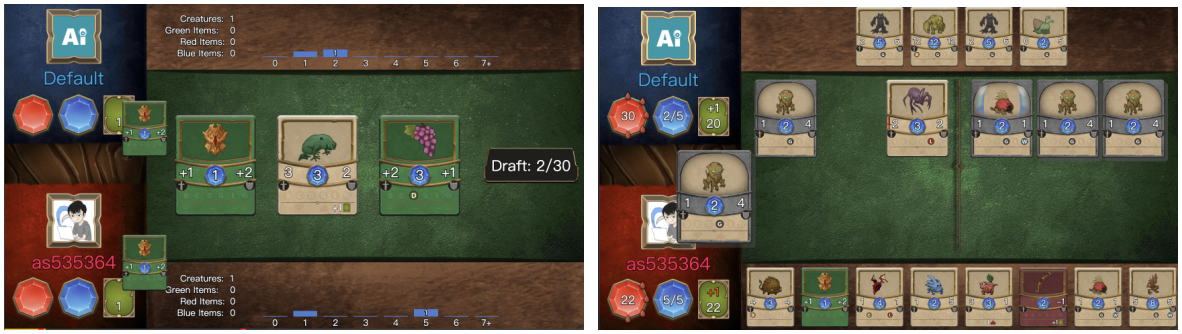
本專題將基於人工智慧技術設計虛擬玩家 (agent)，以組出牌組和玩牌爲兩個研究目標，並運用基因規劃法 (GP)、蒙地卡羅樹搜尋來完成其最佳策略。我們會透過 agent 來參加比賽或與其他公開的 agent 作戰來進行學習，並不斷的自我強化判斷能力，期望可以打敗歷屆高手寫的 agent。

2 研究動機與研究問題

爐石戰記是一款很有名的遊戲，有七千多萬的玩家，因此 LoCM 作爲一款類似的卡牌遊戲，相對於爐石戰記略爲簡化，卡牌種類也不會時常作平衡性調整，但複雜度仍相當高，具有研究價值。我們的研究也助於對於此種卡牌遊戲的了解，可以幫助設計類似的卡牌遊戲或是平衡遊戲強度，也可以研究 AI 產生出來的策略，了解在卡牌遊戲中獲勝的方法。

LoCM 是一個 Imperfect-information 的卡牌遊戲，也就是說有些資訊是不透明的，像是對手玩家的手牌。相對於 Imperfect-information 的是 Perfect-information 常見的例子爲圍棋、象棋，所有的資訊都在場上爲所有玩家知道。LoCM 所有卡牌效果都是不隨機的，因此所有的不確定因素只來自抽卡的順序以及對方手裡未知的卡牌。所以良好的策略還是有很大的效果，這也是我們的研究方向。另外，與爐石戰記不同，LoCM 的卡牌是從卡池中由 AI 挑選出來的，因此無法使用人類玩家預選好的排組。這也使 LoCM 在這方面比爐石戰記有更高的策略性，具有研究價值。

LoCM 的遊戲目的是使用生物或物品牌攻擊，直到雙方有一方生命值歸 0 爲止。共分爲兩個階段：抽牌階段如圖 1(i) 與戰鬥階段如圖 1(ii)



(i) 抽牌階段　　(ii) 戰鬥階段

圖. 1: 階段

- 在抽牌階段，雙方玩家皆要組成 30 張牌的牌組，雙方抽牌的選擇是相同的

- 抽牌階段結束後，雙方的牌組會隨機弄亂
- 在戰鬥階段，盤面會被一分爲二，雙方可將手牌打在屬於他的那一半
- 雙方玩家皆有 30 滴血，有些牌可以加血
- 要降低對手的血量，玩家需透過牌向對手攻擊

遊戲是回合制的，就如圍棋一樣，玩家要操作完才能換下個玩家。在戰鬥階段，玩家的每一回合可做兩件事：使用牌、攻擊。

- 使用牌：使用手牌，可出一張以上。手牌分物品牌跟生物牌。物品牌使用完就會消失，生物牌使用後則會留在盤面上。每一回合可供使用的法力有限，而出牌會消耗法力。如何選出最好的牌來出，是需要研究的問題。
- 攻擊：即用自己盤面上的生物牌攻擊對方盤面上的生物牌或對方本身。但每張牌只能攻擊一個。如何選出最好的目標攻擊，也同樣是需要研究的問題。

LoCM 的卡片能力十分豐富且多樣性高，也有許多不同的屬性。如生物牌有些可阻擋敵方攻擊我方本體、有些可一擊必殺、有些可在使用時多抽牌等。物品牌有些可移除生物牌能力、有些可攻擊生物牌、有些可回血等。如此多的能力，使得遊戲複雜度很高，有很大的研究空間。

3 文獻回顧與探討

3.1 研究議題

要如何透過 AI 在遊戲中取勝，一直是個複雜的問題。適合訓練遊戲 agent 的演算法有很多種，其中 [1] 就是以蒙地卡羅樹爲主，GP 爲輔的混合演算法來製作。

3.2 基因規劃法

演化演算法（Evolutionary algorithm, EA）是一個演算法的集合，其中有各種不同的實現方法，但在這些演算法背後有著相同的精神 [2]：在一個環境中產生一個族群有著許多不同的個體，這些個體之間會互相競爭環境中有限的資源，無法在環境中生存的個體將遭到淘汰，而存活的個體將會繁衍或突變。經過數個世代的演化，將使得整個族群中的個體愈加適應該環境。

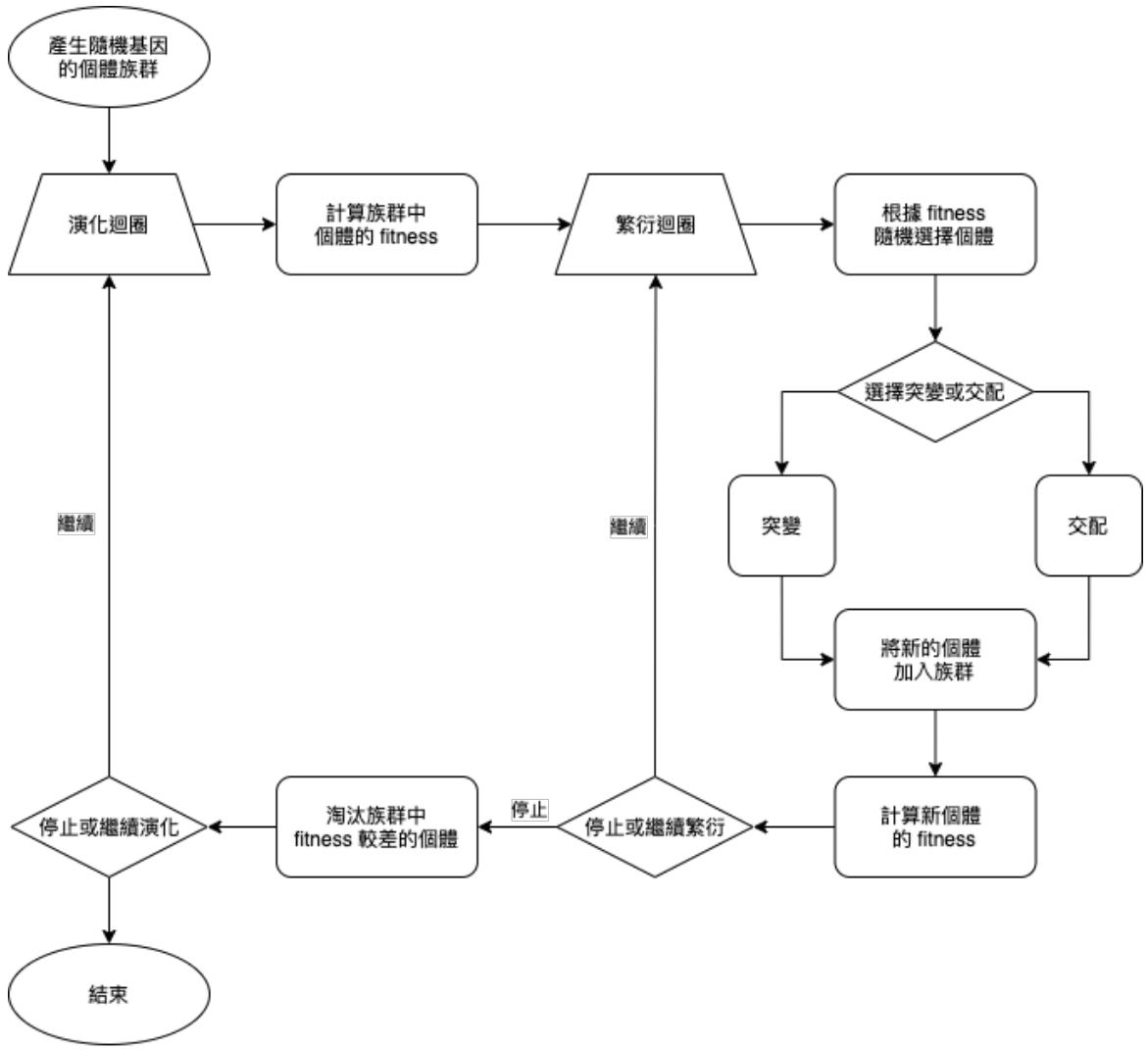


圖. 2: Evolutionary algorithm

這類（EA）的演算法擅長在一個問題的候選解集合中尋找最佳的近似解。其中一種實現方法爲基因規劃法（Genetic programming, GP），在 1990 年由美國史丹佛大學教授 Koza 提出 [3] [4] 。早期 GP 有 Cramer（1985）、BEAGLE（1986）、Koza（1992）等等的實現方法。以 Koza 的方法來說，一個 GP 由以下的元素所組成：

- **The terminal set:** 一個變數與常數（運算元）的集合。在樹的編碼中可以視爲一個葉節點的值。
- **The function set:** 一個函式（運算子）的集合，這些函式的輸入來自 terminal set，可以有不一樣的參數數量（如：一元運算子，NOT。三元運算子：if-then-else）。在樹的編碼中可以視爲非葉節點。
- **The fitness function:** 一個用來評斷個體好壞的函式。fitness 越大的個體，代表其基因越優秀，越容易在演化的過程中保留下其特徵。
- **The control parameters:** 一些 GP 算法中可調整的參數。如族群的大小、交配與突變的機率等等。

- **The termination criterion:** 代表演算法何時可以結束的終止條件。他可以是一個固定的演化次數，也可以是最小可容許的 fitness 值 (當大於時即可結束)。

GP 可以被應用在解決 [5] 符號回歸 (Symbolic regression)、[6] 排程 (Scheduling) 最佳化、控制策略 (Control strategy) 最佳化等問題。本專題同樣可以使用 GP 讓演算法自我學習，嘗試找出最佳的遊戲策略。

3.3 蒙地卡羅樹搜尋

但是只用 GP 是不夠的，因爲場上可能有一些隱藏資訊可能過幾個回合後才顯現出來。以象棋爲例，象棋中的「兌子」，就需要一些回合後才能看出某個走步的優劣。因此我們需要一些樹搜尋演算法像是 Minimax 、蒙地卡羅樹搜尋 (Monte Carlo tree search, MCTS)，樹搜尋演算法可以基於雙方操作迭代推算出過幾步的情況，這是 GP 做不到的。不過由於此類卡牌遊戲的分支因子 (Branching factor) 較大，LoCM 的思考時間也不是那麼充裕，因此我們深入研究 MCTS (通常搜尋較小的遊戲空間) 而不是 Minimax (通常搜尋較大的遊戲空間)。同時 MCTS 也用於圍棋 [7] [8]、六貫棋 [9] [10] 等。

Monte Carlo Tree Search (MCTS) 主要分爲四個階段：

1. Selection: 從當前已經展開的搜尋狀態樹中找到一條從根節點到葉節點的路徑。
2. Expansion: 把前一步驟中找到的葉節點依據可以使用的操作在下面增加一個或多個子節點。
3. Simulation: 把前一步驟中 expand 出的節點作模擬，直到遊戲結束爲止，模擬所使用的操作是預先設計好的，或是隨機的選擇當下可採取的動作。
4. Back-propagated: 模擬結束後把結果一路由葉節點紀錄回根節點。

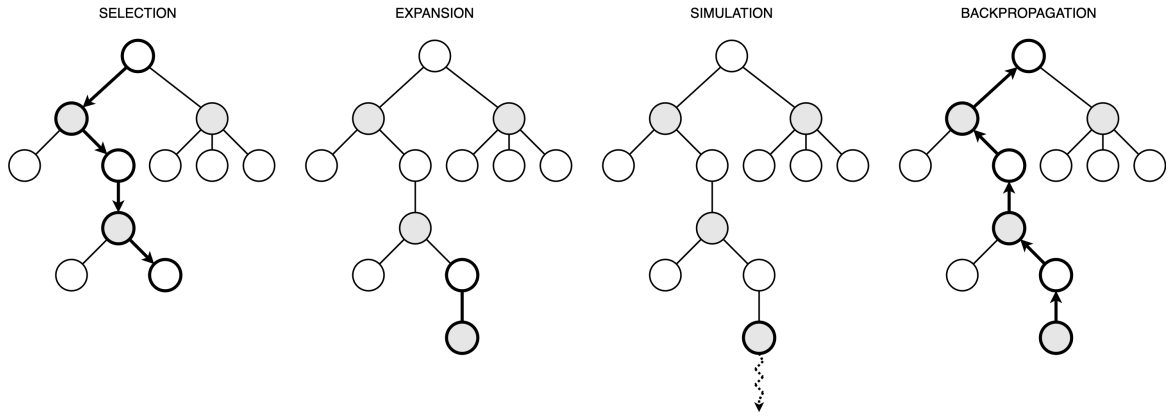


圖. 3: MCTS

對於 LoCM 而言，在 Selection 以及 Simulation 的階段都有改善的空間： 在 [1] 爲了解決有時會因爲勝率的關係沒選到潛在可能較好的狀態，因此引入了 upper confidence bound (UCB) 的概念，UCB 讓現在模擬結果中表現較差的節點也有在 Selection 中被選到的機會，UCB 定義如下。

$$\frac{Q(w)}{N(w)} + c\sqrt{\frac{2lnN(v)}{N(w)}}$$

- c 是常數，用來調整開發項 (左項) 與探勘項 (右項) 間的權重。
- $N(w)$ 代表 w 被拜訪的次數。
- $Q(w)$ 代表 w 勝利的次數。
- $N(w)$ 以及 $Q(w)$ 是在 Back-propagated 的階段紀錄的。

在 Simulation 中，需要模擬雙方操作，因此知道對手手牌，從而得知對手可能採取的行爲也非常重要，在 [1] 中，也利用場面資訊推斷對手可能的牌組，進而能使 Simulation 能更貼近眞實的情況。

3.4 混合演算法

GP 產生的策略，即使已經最佳化，可能依舊勝率不高。爲了解決這個問題，可以搭配蒙地卡羅樹搜尋一起使用，即混合演算法。

混合演算法是指以蒙地卡羅樹搜尋 (MCTS) 爲主，GP 爲輔的演算法。一般的 MCTS 在 Simulation 的階段時，會使用隨機的方式來模擬，但是遊戲中隨機的成分 (從牌組抽到的牌) 和不透明的遊戲資訊 (對手手牌)，會導致其無法準確的模擬對手。也就是說，假設在某個操作下，會導致對玩家不利的結果，但是因爲遊戲模擬的隨機性過高，使遊戲模擬的結果較佳，會讓 MCTS 誤以爲這個操作比較好，導致樹往錯誤的方向發展。這將會影響最終的勝率。爲了解決這種狀況，可以使用大量的模擬次數，讓隨機產生的誤差變小。這會導致所需時間上升。但由於比賽時每一回合的可用時間都有限制，MCTS 的迭代次數和模擬次數都無法太多，所以提高模擬次數不可行。另一個方法是提升 Simulation 階段的準確度以及效率，這可以透過使用勝率較高或執行效率較高的 AI 來達成。勝率較高的 AI 較不會亂使用牌。[1] [11] [12] 皆使用 GP 產生出了 rule-based 的策略，利用 GP 產生的策略進行模擬，可以改進準確性以及效率的問題。這樣的 MCTS 勝率，能比只用 GP 的更上一層樓。

4 研究方法及步驟

LoCM 這款遊戲主要分爲兩個階段：選牌以及戰鬥階段。主要的研究方向有如何使用 GP 在選牌、戰鬥階段中，分別產生出最佳的 rule-based 策略。以及研究如何使用 GP 產生的結果搭配 MCTS 使用 (混合演算法)。

4.1 使用 GP 產生策略

我們透過遊戲的盤面資訊來組成染色體，每條染色體都是一個啓發式函式，也是一個數學公式。每個 agent 皆有四條染色體，分別用來預測不同行動的分數。agent 在行動前，會根據分數做出最好的動作。以下是 agent 的四條染色體：

1. 選牌階段策略的染色體編碼

2. 戰鬥階段使用物品牌策略的染色體編碼

3. 戰鬥階段使用生物牌策略的染色體編碼

4. 攻擊敵人策略的染色體編碼

選牌階段的每一個回合，agent 必須從隨機的三張牌中挑選一張。挑牌不只需要考慮單一一張牌的能力、消耗法力。要考慮牌組整體的平均消耗法力，也要考慮當前已選擇的牌的屬性，使得牌組中的牌可以互相配合。

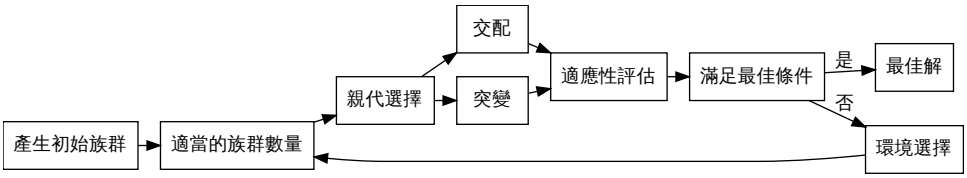
agent 在使用卡牌 (物品牌或生物牌) 時，必須消耗相對應的法力。而每一回合的法力有限，因此它必須決定哪些牌要優先打出。所以當 agent 要使用手牌時，它會根據第一和第二條染色體，來算出手上每一張手牌的分數，並根據分數由小到大，逐一打出，直至法力用盡。

已經被使用、放在場上的生物牌，每一回合可以各選定一個攻擊目標，這個目標可以是敵方的生物牌或敵方的本身。agent 會使用第四條染色體計算每一個敵方目標的分數，根據該分數選擇攻擊的目標。假設我方場上有 n 張生物牌，那麼敵方的每一個目標，皆會被打上 n 個分數，用來表示該目標對於我方 n 張生物牌各自不同的分數。

與 [11] 的啓發式函式不同 (其只包含計算手牌的分數)，本專題的四個啓發式函式包含了遊戲中的每一個決策點，能更全面的演化出最佳策略，因此得出的勝率也較高。

此外，分析演化得出的最佳染色體，可以瞭解哪些盤面資訊在遊戲中扮演重要的角色。

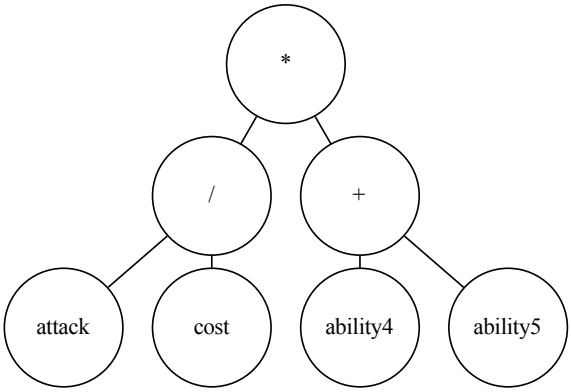
以下是我們的 GP 之流程圖，每跑完一次迴圈算是完成一個世代。



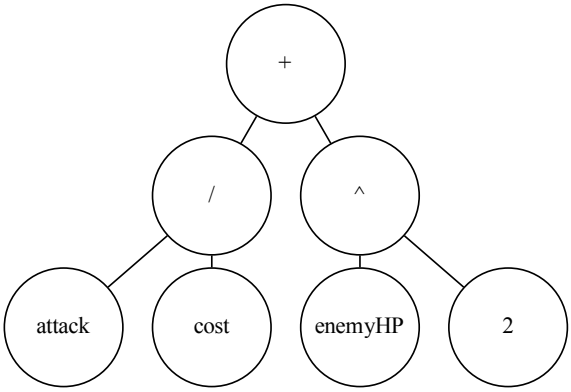
4.1.1 編碼

編碼決定了染色體的組成。在這裡，我們的染色體是由終端集合跟函式集合的元素所構成的。終端集合包含盤面的相關資訊及常數，函式集合則包含兩個輸入或一個輸入的數學操作。整條染色體會變成一個數學公式，在不同情況下，輸出一個不同的數值，用來表示分數。以下是四個染色體的例子：

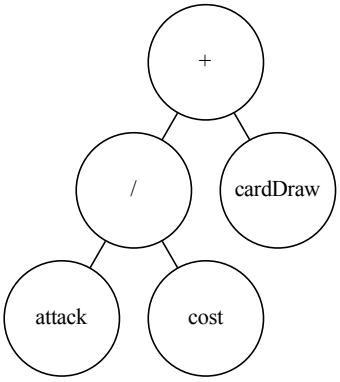
1. 選牌階段策略的染色體編碼



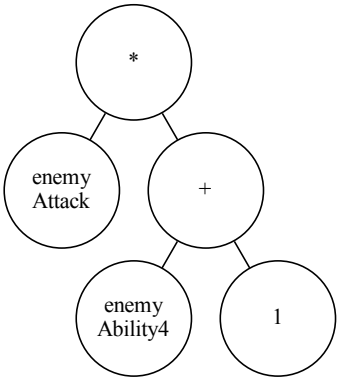
2. 戰鬥階段使用物品牌策略的染色體編碼



3. 戰鬥階段使用生物牌策略的染色體編碼



4. 攻擊敵人策略的染色體編碼



函式集合(運算子)　以下運算子四條染色體皆有。

+	-	*	/	^	sqrt	log	exp	if-else

終端集合(運算元)　以下是四條染色體的運算元，注意每條染色體皆有常數這個額外運算元。每個運算元皆有一個代號。除了常數外，其他運算元都是變數，所以會在不同的情況下有不同的數值。

cost	attack	defense	ability
playerHP	enemyHP	cardDraw	enemyTotalHP
ownTotalHP	enemyTotalAttack	ownTotalAttack	enemyAttack
enemyDefense	enemyAbility		

4.1.2 解碼

每一條染色體皆是一條數學式，用於計算每個操作的分數。以上方四個編碼樹爲例，解碼後分別爲：

1. 選牌階段策略：attack 爲卡牌的攻擊力，cost 爲使用卡牌所需的法力。其比值可以視爲一張牌的「基本價值」。ability4（Guard）可以防止對手直接攻擊我方玩家，ability5（Lethal）則有一擊必殺的效果。當某張牌具有某項能力時，該 ability 值爲 1 否則爲 0 。將一張牌的基本價值與兩個能力一起考慮，可以作爲選牌策略的基準。

$$\frac{\text{attack}}{\text{cost}} \times (\text{ability4} + \text{ability5})$$

2. 戰鬥階段使用物品牌策略：enemyHP 爲使用該卡牌時，可以向對手造成的傷害量。因爲遊戲目標是將對手血量歸零，考慮其傷害量的平方，可以作爲優先使用某物品牌的依據。

$$\frac{\text{attack}}{\text{cost}} + \text{enemyHP}^2$$

3. 戰鬥階段使用生物牌策略：戰鬥階段須從手牌中選擇卡牌使用；而一般情況下，每回合只能從牌組中抽一張牌至手牌中。當手牌越多，代表選擇越多。cardDraw 爲使用該卡牌時，下一回合可以額外抽取的牌的數量，考慮其值可以增加勝利的機會。

$$\frac{\text{attack}}{\text{cost}} + \text{cardDraw}$$

4. 攻擊敵人策略：enemyAttack 爲對手該張卡牌的攻擊力。enemyAbility4 代表對手該張卡牌是否具有 Guard 的能力。有該能力時其值爲 1 否則爲 0 。綜合兩者可以視爲一張對手卡牌的威脅程度，作爲優先攻擊的依據。

$$\text{enemyAttack} \times (\text{enemyAbility4} + 1)$$

將這四個啓發式函式放入我們的 agent 中，即可進行遊戲。藉由放進不同的基因（啓發式函式），來產生不同的遊戲策略與 baseline 進行對戰。baseline 是官方提供的 agent ，實力不算太強，可以做爲我們短期內 GP 的訓練對象。agent 與 baseline 的遊戲策略分別如下：

agent

1. 抽牌階段
   - 依照上述由 GP 產生的啓發式函示挑選牌卡。
2. 戰鬥階段
   (a) 依照上述由 GP 產生的啓發式函示對生物牌、物品牌進行評分、使用，直到法力用盡爲止。
   (b) 使用由 GP 產生的攻擊敵人策略計算分數並依序攻擊。

baseline

1. 抽牌階段

   - 優先選第一張牌
   - 但如果第二或第三張牌有 Guard (必須優先被攻擊) 的屬性時，優先選那張牌。

2. 戰鬥階段

   (a) 依照手牌順序，如果法力足夠就召喚該張牌。

   (b) 如果場上生物可以攻擊，則優先攻擊對方場上具有 Guard 的生物，否則攻擊對方玩家。

   (c) 綠色物品 (具有增益效果) 卡片：優先使用在我方第一個在場上的生物。

   (d) 紅色物品 (具有損益效果) 卡片：優先使用在對方第一個在場上的生物。

   (e) 藍色物品卡片 (具有損益效果): 優先使用在對方玩家身上。

4.1.3 適應性評估

適應性評估能決定染色體的好壞。評估最直接的方法就是以勝率計算，畢竟我們的最終目標就是要訓練出勝率高的 agent 。但是由於一場遊戲就要花不少時間 (0.5 秒)，所以無法比太多場。勝率也會常常重複。而且剛開始在訓練時，勝率往往會在零附近。以上兩個因素，導致 agent 的評估分不出好壞。爲了解決這個問題，我們採用的是每場比賽結束時的血量差來評估。這樣結果會更細，即使每場都輸，血量也會有差距。評估出來就能有差異。

原本我們是使用隨機的種子碼來做爲遊戲中的隨機產生法，因爲正式比賽中也不會有固定的種子碼，這樣可以符合正式比賽的環境，訓練出來的解果也會更貼近所需。但由於遊戲的隨機性太高，導致 agent 無法穩定的演化，假如有個 agent 的實力較好，但在比賽時運氣較差，那在演化的過程中就會被淘汰。爲了解決這個問題，我們採用固定的種子碼來演化，這樣每個 agent 在比賽中的隨機影響都會相同，他們能在完全一樣的環境下競爭。

我們會對每個 agent 進行 n 次比賽，n 會隨著演化越來越高，如此才能得到越來越準的評估。因爲遊戲具有隨機性，所以需要很多場以消弭隨機性的影響。這在演化後期遇到瓶頸時，能做出準確的判斷。

評估的方法是遊戲結束時雙方玩家的血量差異之平均，像是如果遊戲結束時，我方獲得勝利，剩下 20 滴血，而敵方輸了，被扣到剩 -1 滴血，那此場的數值即爲 21。

公式：

$$\sum_{i=1}^{n}(P_i - E_i)/n$$

- P_i 爲第 i 場的玩家剩餘血量
- E_i 爲第 i 場的敵方剩餘血量
- n 爲遊戲場數，公式爲：$\min(\text{generation} + 5,\ 20)$，generation 爲目前進行到哪個世代

4.1.4 初始化族群、選擇、交配與突變

- 初始化族群：隨機生成，深度爲三層的樹，由函式集合與終端集合內隨機挑選的元素構成。
- 選擇的方法：直接選取最好的 50% agents ，剩下的淘汰掉
- 突變的方法：隨機選取染色體中的一個節點，並對他進行改變
- 交配的方法：選取兩個染色體，隨機挑選各一顆子樹，互相交換後接上去

4.1.5 交配與突變前後示意圖：

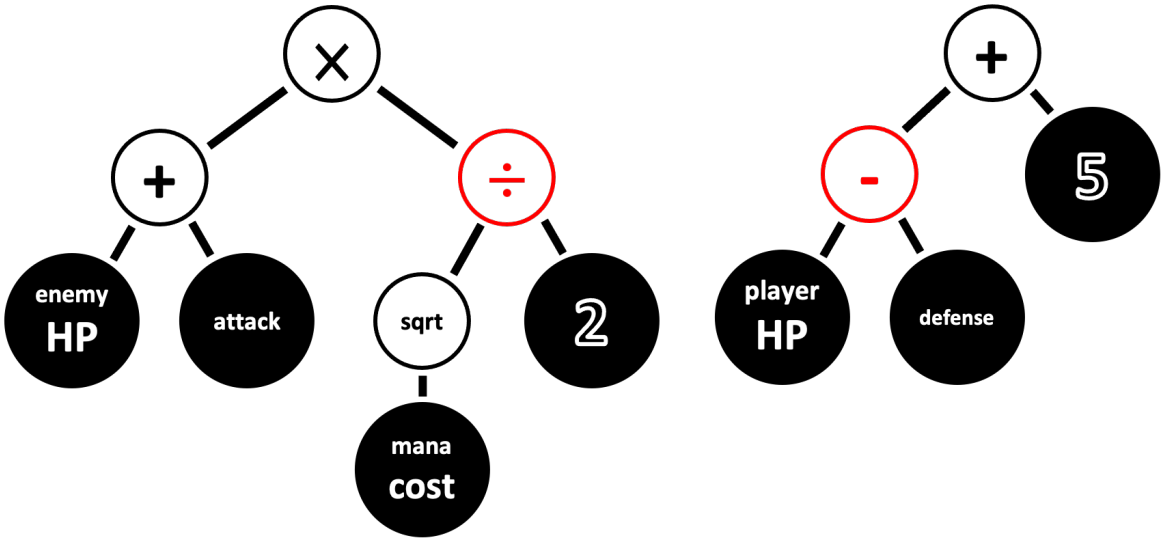


圖. 4: 交配前

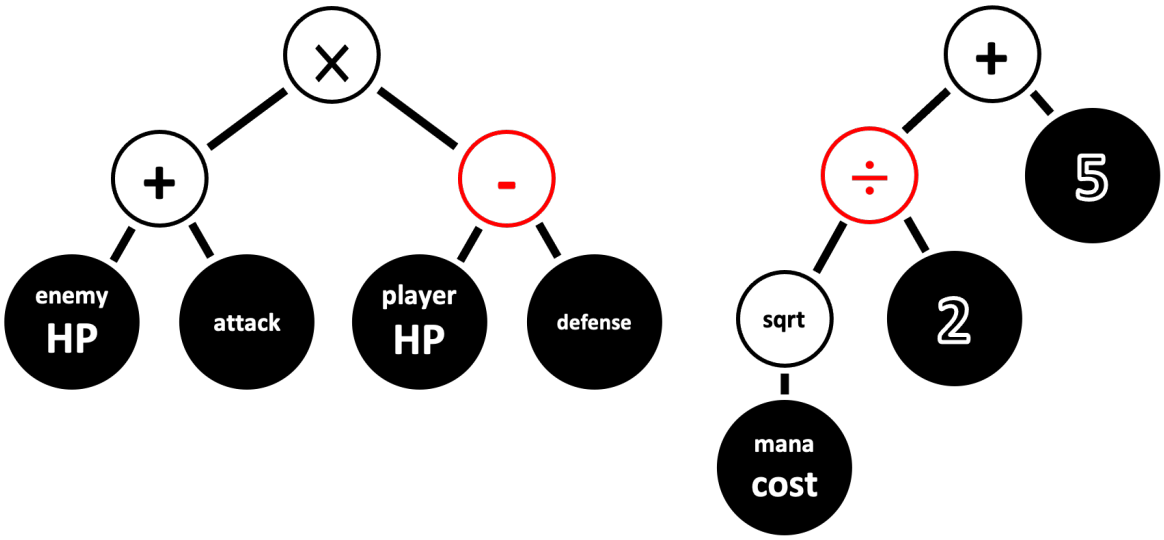
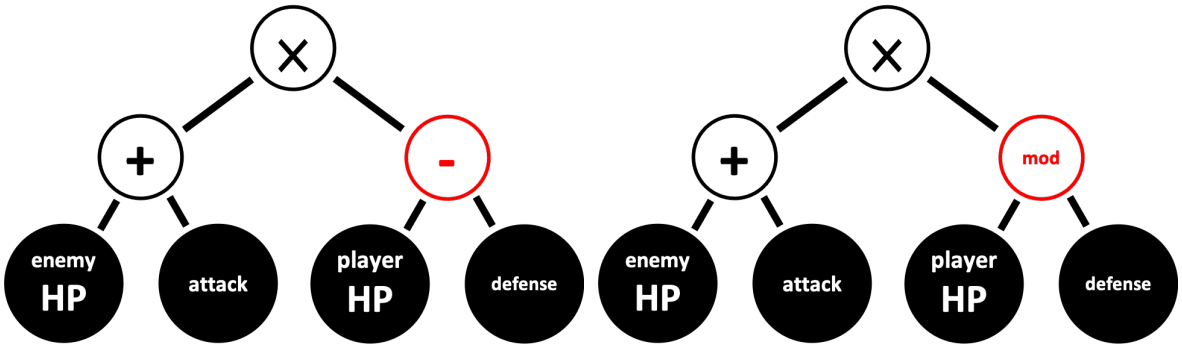


圖. 5: 交配後



(i) 突變前　　(ii) 突變後

圖. 6: 突變

4.2 使用蒙地卡羅樹搜尋

單純使用 GP 有其侷限性，像是無法模擬場上狀況直到遊戲終止，可能會忽略一些需要過幾手才會顯現的資訊。因此我們也需要像是 MCTS 或是 Min-Max 的演算法。但因爲 LoCM 每手思考時間短至 200ms ，因此我們需要在 MCTS 的基礎上做一些調整，如 [1] 以及我們前面所說的 GP 來輔助 Simulation 的階段，或是如 [1] 在 Simulation 加入一些與遊戲有關的 domain knowledge 增加準確度以及速度。

另一個提高 MCTS 效率的方式，是使用漸進式剪枝，他會將成效較低的操作逐漸剪去，被剪去的操作無法成爲 Selection。這樣可以將時間集中在較好的操作上

我們也可以預測對手的行爲，像是透過他使用過哪些牌，來預測他目前有哪些手牌。這可以幫助我們更準確地預測結果。

5 預期結果

透過 GP 長時間的訓練，最佳化出一個完整的 rule-based agent，並搭配 MCTS 使用。使用它來參加正式的比賽、並獲得佳績。

同時研究 GP 演化出來的策略，了解在卡牌遊戲中什麼因素會導致遊戲的勝負。並對於如何平衡卡牌遊戲，有一套完整的標準。

6 參考文獻

[1] A. Santos, P. A. Santos, and F. S. Melo, “Monte carlo tree search experiments in hearthstone,” in *2017 IEEE Conference on Computational Intelligence and Games (CIG)*, pp. 272–279, IEEE, 2017.

[2] A. E. Eiben, J. E. Smith, *et al.*, *Introduction to evolutionary computing*, vol. 53. Springer, 2003.

[3] J. R. Koza, *Genetic programming: A paradigm for genetically breeding populations of computer programs to solve problems*, vol. 34. Stanford University, Department of Computer Science Stanford, CA, 1990.

[4] J. R. Koza and J. R. Koza, *Genetic programming: on the programming of computers by means of natural selection*, vol. 1. MIT press, 1992.

[5] D. A. Augusto and H. J. Barbosa, "Symbolic regression via genetic programming," in *Proceedings. Vol. 1. Sixth Brazilian Symposium on Neural Networks*, pp. 173–178, IEEE, 2000.

[6] S. Nguyen, M. Zhang, M. Johnston, and K. C. Tan, "Automatic design of scheduling policies for dynamic multi-objective job shop scheduling via cooperative coevolution genetic programming," *IEEE Transactions on Evolutionary Computation*, vol. 18, no. 2, pp. 193–208, 2013.

[7] R. Coulom, "The monte-carlo revolution in go," in *The Japanese-French Frontiers of Science Symposium (JFFoS 2008), Roscoff, France*, vol. 115, 2009.

[8] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of monte carlo tree search methods," *IEEE Transactions on Computational Intelligence and AI in games*, vol. 4, no. 1, pp. 1–43, 2012.

[9] B. Arneson, R. B. Hayward, and P. Henderson, "Monte carlo tree search in hex," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 2, no. 4, pp. 251–258, 2010.

[10] B. Arneson, R. Hayward, and P. Henderson, "Mohex wins hex tournament," *ICGA journal*, vol. 32, no. 2, p. 114, 2009.

[11] H.-C. Chia, T.-S. Yeh, and T.-C. Chiang, "Designing card game strategies with genetic programming and monte-carlo tree search: A case study of hearthstone," in *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 2351–2358, IEEE, 2020.

[12] A. M. Alhejali and S. M. Lucas, "Using genetic programming to evolve heuristics for a monte carlo tree search ms pac-man agent," in *2013 IEEE Conference on Computational Inteligence in Games (CIG)*, pp. 1–8, IEEE, 2013.

7 需要指導教授指導的內容

指導教授研究專長爲啓發式演算法，且研究室學生亦在進行相關之研究，希望藉由教授多年來的研究經驗及與本專題有關的相關背景知識，提供實作本專題的指導和建議。