國立交通大學

資訊科學與工程研究所

碩 士 論 文

平行蒙地卡羅樹狀搜尋之軟體框架

Software Framework for Parallel Monte Carlo Tree Search

研究生： 廖挺富

指導教授：吳毅成 教授

中 華 民 國 一百零二 年 八 月

平行蒙地卡羅樹狀搜尋之軟體框架

Software Framework for Parallel Monte Carlo Tree Search

研 究 生：廖挺富　　Student： Ting-Fu Liao

指導教授：吳毅成　　Advisor： I-Chen Wu

國 立 交 通 大 學

資 訊 科 學 與 工 程 研 究 所

碩 士 論 文

A Thesis

Submitted to Institute of Computer Science and Engineering

College of Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer Science

August 2012

Hsinchu, Taiwan, Republic of China

中華民國一百零二年八月

平行蒙地卡羅樹狀搜尋之軟體框架

研究生： 廖挺富　　指導教授：吳毅成

國立交通大學資訊科學與工程研究所

摘要

在本篇論文中，我們設計出了一個軟體框架，能協助遊戲人工智慧的開發者，快速開發基於平行化蒙地卡羅樹狀搜尋的人工智慧。這個框架實作了與系統相關的細節，能夠使開發者能夠快速地建立起人工智慧，並專注在遊戲相關的專家知識強化上。

這個框架支援在共享記憶體架構以及分散式系統的平行，經測試可在 Windows 系統以及 Linux 系統執行，並且已套用至雙人完整資訊(perfect information)遊戲、雙人不完整資訊(imperfect information)遊戲以及單人遊戲上。我們的實驗使用圍棋版本的實作；在共享記憶體平行化中，使用 48 核心可以達到 36.08 倍的模擬次數，與單核心版本對戰，使用 12 核心可以達到 90%以上勝率。在分散式平行化下，我們比較單一台機器與 8 台機器各使用 36 核心，分別與 Fuego 對戰，勝率從 23%提升至 68%。除圍棋外，亦套用至禁圍棋、暗棋等遊戲上；其中圍棋利用此平行框架，同時使用 576 核心進行運算，在十七屆電腦奧林匹亞電腦對局競賽中，獲得第四名。

Software Framework for Parallel Monte Carlo Tree Search

Student: Ting-Fu Liao Advisor： I-Chen Wu

Institute of Computer Science and Information Engineering

National Chiao Tung University

Abstract

In this thesis, we design a software framework for developing computer games based on parallel Monte-Carlo tree search. This framework hides the game-independent details from developers, so that developers can concentrate on improving heuristics related to game-specific knowledge.

This framework supports parallelization in both shared-memory and distributed systems, and cross-platform in both Windows and Linux operating systems. In our experiments, we use the implementation of Go. In shared-memory parallelization, the speedup is 36.08 for 48 cores, and the winning rate against the single core version is more than 90% for 12 cores. In distributed-memory system, the winning rate for one machine with 36 cores against Fuego is 23%, while that for 8 machines each with 36 cores is 68%, about 45% higher. In addition to Go, we also applied this framework to another two-player perfect-information games, NoGo, a two-player

imperfect-information game, Dark chess, and a single player game, 8 puzzle. The one for Go using a total of 576 cores won the 4th place in the 17th Computer Olympiad, held in Yokohama, Japan, 2013.

誌謝

這篇論文能夠完成，首先要感謝我的指導教授吳毅成教授以及李素瑛教授，沒有老師的建議與指導，絕對無法完成這篇論文；同時也感謝口試委員許舜欽教授、留忠賢教授以及林秉宏博士對本篇論文的缺失提出精闢的建議。

感謝我的家人，因為有你們的支持，才能讓我沒有後顧之憂的完成學業。

感謝所上以及財團法人聯詠科技教育基金會提供的獎學金，讓我在研究生生涯的兩年可以不用擔心生計，專注在研究上。

感謝實驗室的學長、學弟以及同學們，在研究上互相砥礪與協助。感謝吳東穎、高振綱、張元耀與連育賢維護實驗室的機器，讓實驗能夠順利完成。感謝楊秉恩的仙人掌以及魏經軒將六子棋移至我的手機，讓我紓解研究的壓力。感謝張傑閔和我一起去游泳及健身房。感謝圍棋組的同伴，左存道講解 HappyGo 的各項細節，讓我能快速的理解開發多年的程式，並和何庭築一起將 HappyGo 移植到軟體框架上，楊秉恩用工作層級演算法建立開局庫，讓圍棋 AI 變得更強。

感謝使用這個軟體框架的開發者，提供功能面的寶貴意見。並感謝國家高速網路與計算中心提供穩定的機器，讓各項實驗可以順利執行。

此論文獻給所有支持我的老師、家人、朋友們，

民國一百零二年九月　於　新竹交大工程三館電腦遊戲與智慧實驗室

目次

表目錄

圖目錄

第 一 章 緒論

本章節會在第 1.1 節介紹這篇論文的研究動機，並於第 1.2 節列舉這篇論文能達到的目標，最後第 1.3 節介紹整篇論文的架構。

1.1 研究動機

蒙地卡羅樹狀搜尋(Mnte-Carlo Tree Search，以下簡稱 MCTS)自 2006 年被成功地應用在電腦圍棋後[9]，開始被廣泛地應用在各類桌上遊戲上[1][18][21]，以及其他非桌上遊戲的人工智慧如 Pacman[17]、general game playing[12]。同時也被用於各類最佳化問題如零工式工廠排程問題(Flexible Job Shop Scheduling Problem)[20]、火力機組發電排程問題(Unit Commitment Problem)[7]等。

會影響基於 MCTS 的遊戲人工智慧的強度，有兩類因素。一是跟該遊戲相關的，專家知識的加入，或是各種機器學習的方法如 Simulation Balance[15]、Minorization Maximization[16]，以及一些啟發式的方法如 All Moves As First、Rapid Action Value Estimation[14]以及 Last Good Reply[10]等。

另一類就是與系統相關，改進 MCTS 本身的架構，如良好的記憶體控管、高效率的樹狀結構、以及將 MCTS 平行化[6]；然而在架構設計與實作上，需要對系統有足夠的了解，否則無法獲得預期的效能改進。因此若能提供一個良好的 MCTS 軟體框架，讓遊戲人工智慧開發者能夠專注於遊戲相關知識的實作。

本篇論文設計一個平行化的蒙地卡羅樹狀搜尋之軟體框架;使各類遊戲開發者能透過該框架建立其人工智慧程式。

1.2 研究目標及成果

本論文設計一個基於平行化 MCTS 的軟體框架，讓遊戲人工智慧開發者能夠容易地建立其遊戲人工智慧，並專注在遊戲相關設計如強化專家知識；並提供下列特性：

- 支援有限人數的遊戲
- 支援完整資訊以及非完整遊戲
- 支援 Windows 以及 Linux 作業系統
- 支援共享記憶體架構的無鎖樹平行
- 支援分散式系統的根平行

目前應用本軟體框架的程式如下：

- 圍棋：雙人完整資訊遊戲，在十七屆電腦奧林匹亞電腦對局競賽中獲得第四名。
- 禁圍棋：雙人完整資訊遊戲，在十七屆電腦奧林匹亞電腦對局競賽中獲得第一名。
- 暗棋：雙人不完整資訊遊戲，在 2013 TCGA 電腦對局競賽中獲得第四名。

- 8 puzzle：實作單人遊戲，證明本軟體框架可以應用在單人遊戲上。

1.3 論文組織

在第一章中，我們介紹了本篇論文的研究動機以及要完成的目標，第二章內會介紹相關的背景知識，第三章將會說明我們如何設計這個軟體框架，並在第四章利用一些例子說明如何快速的使用這個框架，最後在第五章下結論，並說明未來研究的方向。

第 二 章 背景

在這個章節，我們會介紹一些本篇論文用到的相關知識。2.1 節介紹 MCTS 的基本概念及應用。2.2 節介紹平行化 MCTS 的各種做法。2.3 節介紹先前其他人做的成果。2.4 節介紹我們在實作這個軟體框架所用到的函式庫。

2.1 蒙地卡羅樹狀搜尋

蒙地卡羅方法(Monte Carlo Method)是一種基於大數法則的方法，利用隨機取樣以統計逼近實際結果；在 1949 年由 Metropolis 等人於提出[19]，應用於在核子武器中，中子在反應爐中的傳輸過程，提出後被廣泛應用在許多領域上。於 1993 年被應用在電腦圍棋上[5]，但並未獲得突破性的進展。

2006 年，Crazy Stone[9]將蒙地卡羅方法應用在樹狀搜尋上，獲得電腦奧林匹亞電腦對局競賽九路圍棋金牌，開始了在蒙地卡羅樹狀搜尋(以下簡稱 MCTS)在電腦對局上的研究。隔年，Mogo[13]將 Upper Confidence Bound (UCB)的概念應用在蒙地卡羅搜尋樹上，稱之為 Upper Confidence Bound Applied to Trees (UCT)[4]，於當年的電腦奧林匹亞十九路圍棋競賽中，獲得全勝的紀錄。

2.1.1 蒙地卡羅樹狀搜尋流程

蒙地卡羅樹狀搜尋是一種最佳優先搜尋(Best-first search)[8]，其演算法包含下列四個階段：選擇(selection)、展開(expansion)、模擬(playout)以及更新(update)，這四個步驟執行一次稱為一個 Simulation，藉由重複進行 Simulation 來進行搜尋。

1. 選擇：從根節點開始，根據先前的資訊選擇一最好的子節點，直到葉節點為止。
2. 展開：根據選擇的結果，從被選擇的節點展開一新的子節點。
3. 模擬：從展開的節點開始，根據規則隨機選步，直到模擬結束。
4. 更新：根據模擬結果，更新選點路徑的資訊。

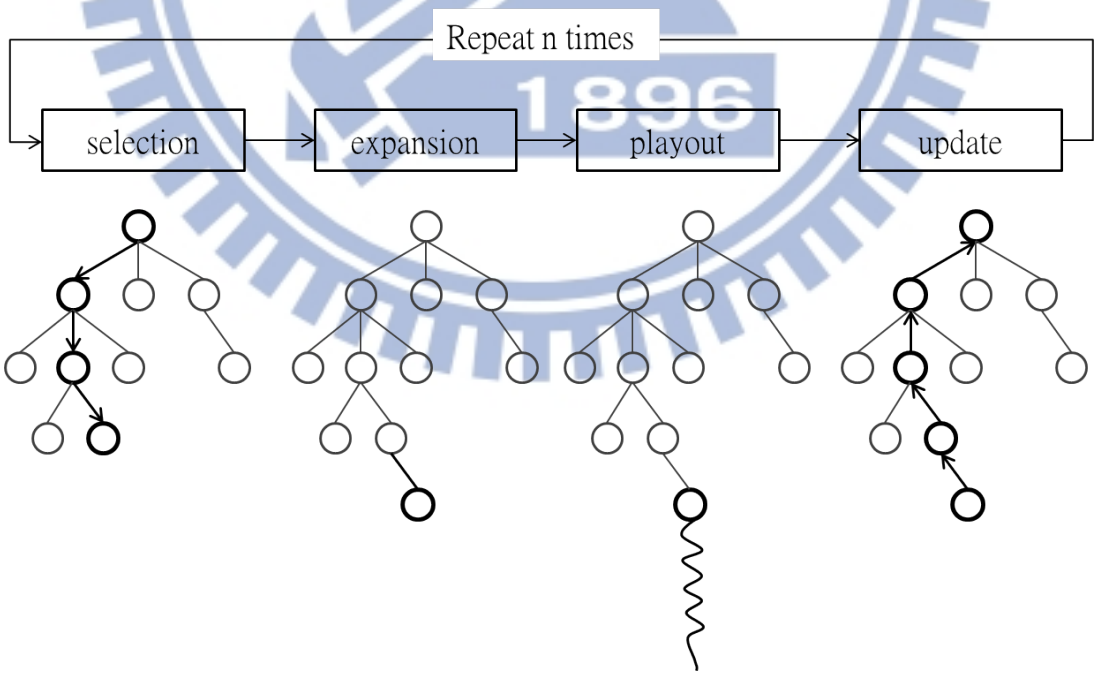


圖 1. 蒙地卡羅樹狀搜尋

2.1.2 Upper Confidence Bound Applied to Trees

在 MCTS 中，我們必須要平衡探勘(Exploration)與開發(Exploitation)，Upper Confidence Bound (UCB)即是用於平衡這兩者的公式。在 MCTS 的選擇階段，選擇根據 UCB 公式計算出分數最佳的子節點，其公式如下：

$$UCB_i = X_i + C \times \sqrt{\frac{\log N}{N_i}} \quad (1)$$

- UCB_i：子節點i的 UCB 值
- X_i：子節點i的目前的平均分數
- C：UCB 常數，用於調整探勘與開發的比例
- N：父節點目前的拜訪次數
- N_i：子節點i的目前的拜訪次數

在這個公式中，利用X_i項引導 MCTS 選擇目前認為較好的子節點；同時利用$\sqrt{\frac{\log N}{N_i}}$項，當父節點的拜訪次數相當多，而子節點$i$的目前的拜訪次數相對少時，該項數值會增加，藉此避免好的子節點被埋沒。

若將 UCB 公式應用在樹狀搜尋，則稱為 Upper Confidence Bound Applied to Trees (UCT)。

2.2 平行化蒙地卡羅樹狀搜尋

平行化 MCTS 的方法[6]主要有三種：葉平行(leaf parallelization)、根

平行(root parallelization)以及樹平行(tree parallelization)，其概念圖如圖2。

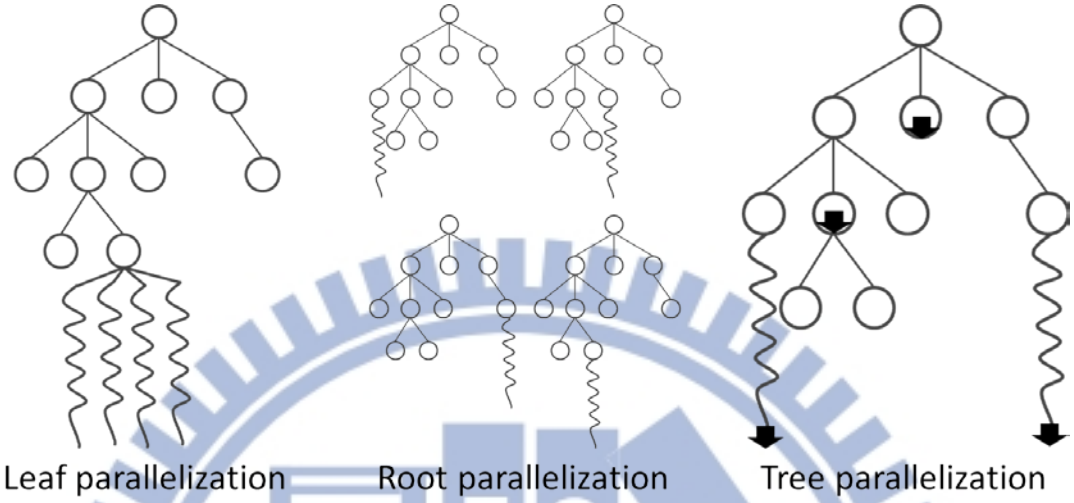


圖 2. 三種平行化 MCTS 的方法

2.2.1 葉平行

葉平行是平行化 MCTS 最簡單的方法之一，基本想法是利用多個執行緒(thread)進行模擬，用以增加模擬的準確度；其優點在於不需要使用互斥鎖(mutex lock)。然而，由於葉平行只對模擬階段進行平行化，因此若有其中一個的執行緒花費較多時間進行模擬，其他已經模擬完畢的執行緒仍需等待，此時會浪費大量的運算資源；故平行化效果並不好，現在已較少人使用葉平行。

2.2.2 根平行

相較於葉平行只對模擬階段進行平行化，根平行的做法是讓每個執行緒都擁有一棵獨立的搜尋樹；同時因為每棵搜尋樹是獨立的，故根平行

在實作上也不需要使用互斥鎖。但相對的，搜尋樹獨立也代表著每棵樹之間的資訊並不會被共用，因此每棵搜尋樹可能會得到相當類似的結果。目前多用在分散式系統的平行化上。

2.2.3 樹平行

類似於根平行，每個執行緒獨立的進行搜尋，但所有執行緒共用同一棵搜尋樹；根據其保護資料的方式分為三種：對整棵搜尋樹使用全域鎖(global lock)、對每個節點使用區域鎖(local locks)以及不使用鎖(no lock)。概念圖如圖 3。

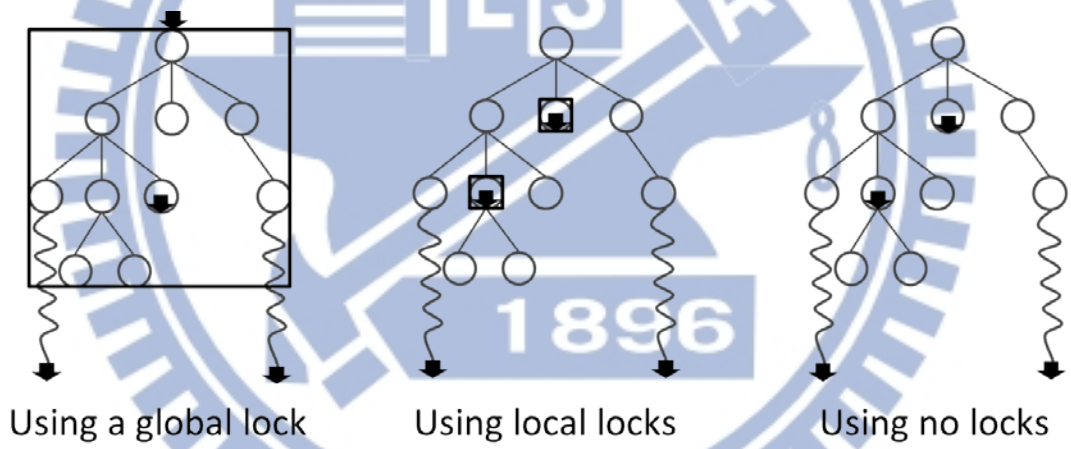


圖 3. 三種樹平行的方法

- 使用全域鎖

  利用全域鎖在選擇、展點以及更新階段時，確保當下只有一個執行緒在存取該搜尋樹；在搜尋樹被鎖定時，其他執行緒仍可以進行模擬。在搜尋樹被鎖的狀態下，若有其他執行緒需要存取搜尋樹，則需要等待搜尋樹被解鎖。

- 使用區域鎖

相較於全域鎖鎖定整個搜尋樹，使用區域鎖只鎖定正在處理的節點，藉此允許多個執行緒同時存取搜尋樹的不同部分，增加平行度。

- 不使用鎖

Martin Muller 在 2009 年提出了一種不需要互斥鎖的方式[11]，可以有效的提升平行化 MCTS 的效能。其做法預先從系統取得大量的空間用於儲存節點資訊，並利用兩個變數(子節點的數量以及第一個子節點的位置)維持搜尋樹結構。

在展開階段，展開父節點時，從預先取得的節點陣列中，分配一段連續的空間給子節點，將資訊寫入父節點時，先寫入子節點的數量，然後才寫入第一個子節點的位置；若有多個執行緒同時展開同一個節點，則僅有最後寫入的會被使用，這種狀況下會造成一點記憶體損失，但由於是自行管理預先從系統取得的空間，因此不會造成記憶體洩漏(memory leak)。

在選擇階段，大部分的操作皆只會讀取搜尋樹的資料，僅須注意若指向第一個子節點的指標為空時，則視為沒有子節點。

在更新階段，直接將更新的資訊寫入；若多個執行緒同時更新，會造成一部分的更新資料沒有正確的更新進節點，但在實作上這種狀況發生的情形相當少。

2.3 先前的研究

在這個章節會介紹其他過去有實作過平行化 MCTS 的軟體，包含一個相當優良的軟體框架 Fuego，以及一個也是使用平行化 MCTS 的圍棋程式 Pachi。

2.3.1 Fuego

Fuego[11]是一個開放原始碼的軟體框架，用於開發雙人完全資訊的遊戲人工智慧，提供 Alpha-Beta 以及平行化 MCTS 兩種演算法。在記憶體管理上，Fuego 讓每一個執行緒有一個獨立的陣列，藉此讓每個執行緒可以獨立的進行展開。

2.3.2 Pachi

Pachi[2]是一個開放原始碼的圍棋程式，實作了平行化的 MCTS。大部分實作與 Fuego 相似。但是在記憶體管理上，Pachi 讓所有執行緒共用一個預先取得的陣列；在展開階段，利用 GCC 提供的擴充指令，進行原子操作(atomic operation)，確保每個執行緒展開子節點時，都是使用不同的記憶體位置。

2.4 Boost libraries

Boost libraries[3]是一個跨平台的 C++函式庫集合，可支援大部分常見的平台如：Linux、BSD、Windows 等。我們選用了下列這些的函式庫，

作為跨平台之用途。

- Boost.Thread

  Boost.Thread 是一個跨平台的多執行緒函式庫。我們使用這個函式庫，讓我們的軟體框架能夠在各種平台都有相同的實作。

- Asio

  Boost.Asio 是一個用於網路以及低階輸入輸出操作的跨平台函式庫，同時也提供非同步(asynchronous)的輸入輸出操作。我們在分散式系統平行化使用這個函式庫，做為跨平台的網路溝通以及非同步輸入輸出的實作。

第 三 章 框架設計

本章節將會說明本篇論文的框架設計，包含共享記憶體的平行化以及分散式系統的平行化。圖 4 為本軟體框架的系統架構圖，包含軟體框架提供的模組，以及開發者需要實作的模組。

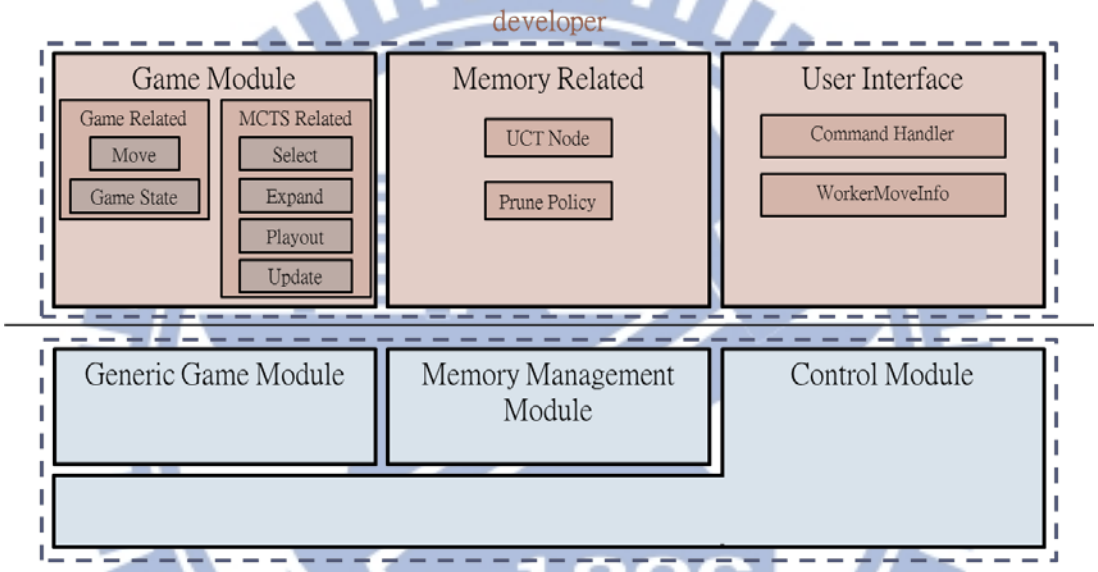


圖 4. 系統架構圖

3.1 通用遊戲模組(Generic Game Module)

通用遊戲模組負責連接開發者提供的遊戲模組。遊戲模組包含兩個部份：其一是與遊戲相關，另一則包含 MCTS 的四個步驟。

3.1.1 遊戲相關

與遊戲相關的部分包含兩個類別：Move 以及 GameState，開發者必須正確的實作這兩個類別，藉此才能與這個軟體框架溝通。

- Move：用於表達在該遊戲中，怎樣叫做一個動作。例如在圍棋中，黑方或白方下在某個位置就是一個 Move。由於這個類別會被用於參數傳遞，開發者在設計上必須盡可能的降低它的大小，以避免效能的損失。
- GameState：用於表達在該遊戲中的狀態。例如在圍棋中，那些位置有黑子或白子，以及一些圍棋相關的規則，開發者都必須在這個類別內實作。同時，為了與我們的軟體框架溝通，這個類別必須實作列於表 1 的方法。

reset()	重置整個遊戲狀態。
backup()	備份遊戲狀態，以便在 Simulation 結束後能快速地回復狀態。
rollback()	將遊戲狀態回復至前一次備份。
play(Move)	改變遊戲狀態，此處的 Move 即是開發者要實作的另一項類別。

表 1. GameState 需實作的方法列表

3.1.2 MCTS 相關

MCTS 的四個步驟被封裝進兩個類別中：UctAccessor 以及 PlayoutAgent，主要差別在於該步驟是否能存取 UCT，開發者可根據其需求複寫部分的方法。

- UctAccessor：包含能存取 UCT 的三個階段，選擇、展開、更新。其方法如表 2。

select(root)	傳入根節點，回傳最佳的葉節點。預設實作為從根節點根據 UCB 公式選擇小孩直到葉節點。
expand(node)	傳入被選到的葉節點，展開該節點的所有子節點。
update()	根據模擬結果更新選擇路徑上的資訊。

表 2. UctAccessor 可複寫的方法列表

- PlayoutAgent：包含模擬階段的行為，在這個類別中，無法存取 UCT。其方法如表 3。

run()	從當下遊戲狀態模擬至結束狀態。

表 3. PlayoutAgent 可複寫的方法列表

3.2 記憶體管理模組(Memory Management Module)

記憶體管理模組負責維護 UCT 結構、節點管理、並且提供兩個用於分配節點空間的類別。

3.2.1 UCT 結構

在本軟體框架中，我們依據 2.2.3 提及的無鎖樹平行方式來實作。節點只包含維持樹狀結構以及執行 MCTS 必要的欄位。若開發者需要擴充每個節點紀錄的資訊，須繼承基底節點類別後，再加入需要的欄位。惟須注意的是，節點大小會對實際記憶體用量有顯著影響，開發者須避免增加不必要的欄位。

3.2.2 節點管理

在節點管理上，我們使用了分頁的概念，每次向系統索取一定數量的節點，稱之為一個分頁(page)，管理方式如圖 5。同時，我們用一個 32 位元的整數來表達節點的位置，並封裝成 `NodePtr` 類別；藉此降低在 64 位元系統中，維持樹狀結構所需消耗的記憶體。`NodePtr` 格式如圖 6，利用前 n 個高位元，表達該節點是在第幾個分頁，並利用末 m 個低位元，表達該節點在分頁中的位置。

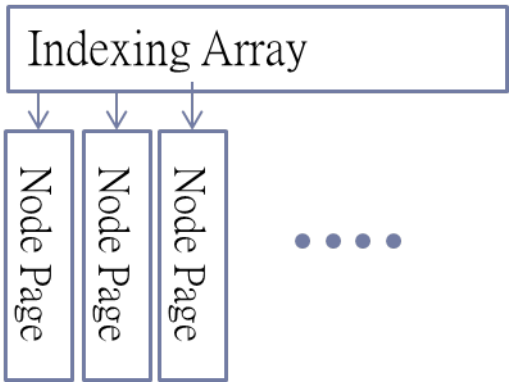


圖 5. 分頁管理示意圖

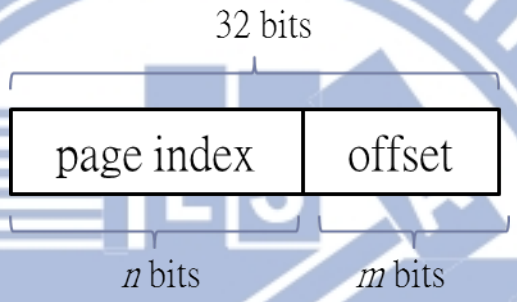


圖 6. NodePtr 格式

在實際使用上，NodePtr 必須轉換成原生的指標，才能取得該節點內的資料，轉換過程如表 4：

```
procedure resolve_address(node)
1. offset ← right m bits of node.index
2. index  ← node.index >> m
3. page ← GetPage(index)
4. return page.GetNode(offset)
```

表 4. NodePtr 轉換流程

首先從 NodePtr 中，利用位元運算子取得該節點在第幾個分頁，以及為該分頁中第幾個節點；取得該分頁後，再從分頁中取得節點實際位置。

3.2.3 節點分配

在節點分配上，我們提供了兩個類別來協助節點分配。一個分頁分配者(Page Allocator)以及每個執行緒有自己的節點分配者(Node Allocator)。

分頁分配者，負責向作業系統取得分頁，維護一個索引陣列以便能從 NodePtr 轉換成原生指標，並提供分頁給各個執行緒的節點分配者，流程如表 5。

```
procedure allocate_page()
1. if MAX_PAGE pages exceeded then
2.    return null
3. end if
4. if not PA.has_free_page() then
5.    allocate new page from system
6. end if
7. page ← PA.consume_page()
8. return page
```

表 5. 向分頁分配者取得分頁流程

分頁分配者在提供分頁時，首先確認是否使用的分頁已超過 MAX_PAGE，若使用數量已超過，回傳 null 並修剪搜尋樹，釋放無用的空間；接著檢查當下是否有空的分頁可以使用，若沒有則向系統取得新的分頁；最後將一個空的分頁回傳給節點分配者。

每個執行緒有一個獨立的節點分配者。負責向分頁分配者取得獨佔的分頁，並從該分頁中提供節點空間給展開時使用，流程如表 6。

```
procedure allocate_nodes(n)
1. if not page.has_enough_nodes(n) then
2.     allocate new page from Page Allocator
3. end if
4. first ← page.consume_nodes(n)
5. return first
```

表 6. 向節點分配者取得節點流程

在展開階段，向節點分配者取得節點時，首先檢查是否該分頁有足夠的節點數，若否則像分頁分配者取得新的分頁；其後標記使用 n 個節點並將第一個節點的位置回傳。

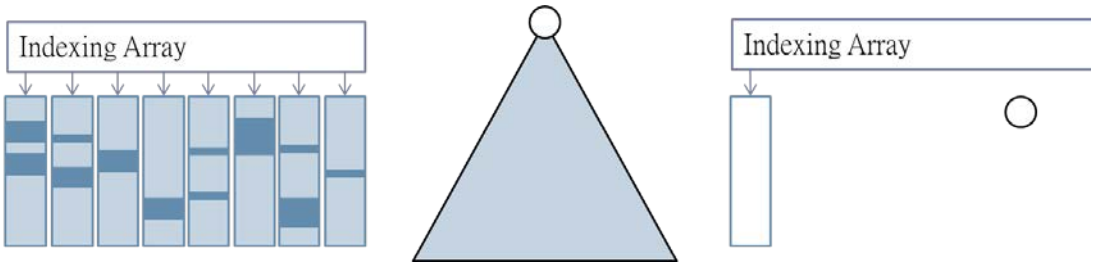
3.2.4 修剪搜尋樹

當分頁用盡時，我們會修剪搜尋樹，將較不重要的節點或是因同時展開而浪費的節點回收，然後再繼續進行搜尋。在進行修剪時，開發者可以透過實作 PrunePolicy 來決定哪些節點應該要被留下，其原型如下。

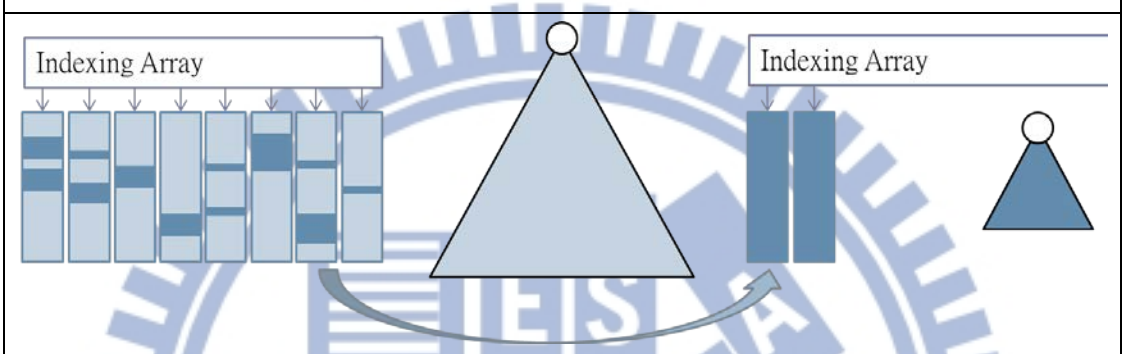
- `bool prunePolicy(parent, keeplist);`

  傳入父節點(parent)，若該節點需被設為未展開，則回傳 true；否則回傳 false，並在 keeplist 內紀錄要保留的子節點。

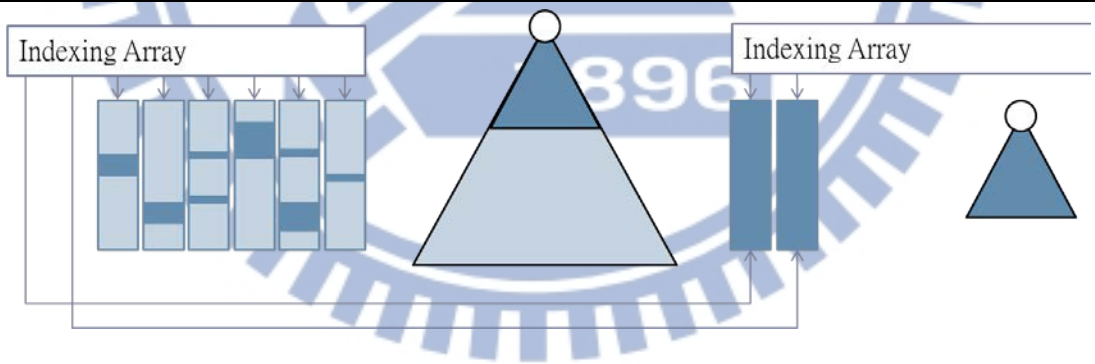
軟體框架修剪搜尋樹的流程如下表 7；在下列示意圖中，深色區塊表示原本的搜尋樹上，我們想要保留的部分。



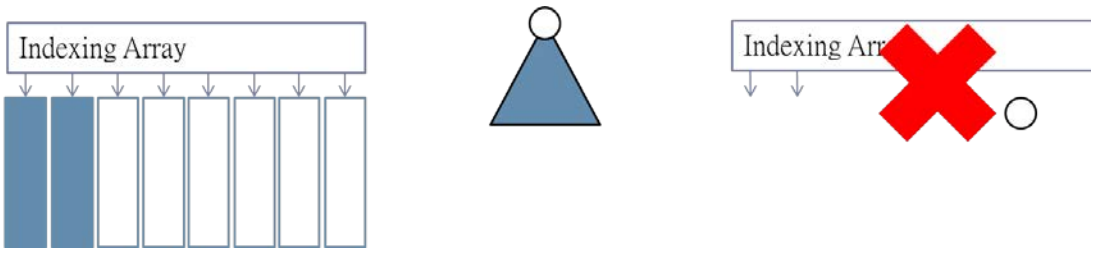
1. 建立一個暫存用的索引陣列，並取得一個新的分頁，複製根節點至該分頁。



2. 遞迴的根據 `PrunePolicy` 從原本的搜尋樹複製節點至新的分頁上，在過程中可能會需要更多的新分頁。



3. 用新分頁取代原本索引陣列的部分分頁。



4. 移除暫存的索引陣列，並清空剩餘的分頁。

表 7 修剪搜尋樹示意圖

3.3 控制模組(Control Module)

控制模組提供一組應用程式介面(Application Programming Interface)讓開發者實作使用者介面，以及控制執行緒進行 MCTS 的運算。其概念圖如圖 7。

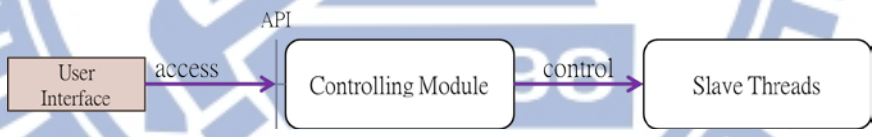


圖 7. 控制模組概念圖

控制模組提供的 API 包含兩部分：透過表 8 所列舉的 API，可以實作基本的遊戲互動。若需要實作背景思考功能，則需使用表 9 所列的 API。

`void newGame()`	清空盤面，並開始一場新的遊戲
`bool play(Move)`	下一手棋，對盤面進行更改
`Move genMove()`	用 MCTS 進行搜索，產生一手棋
`State& getState()`	取得當下的遊戲狀態
`void modifyState(Modifier)`	改變當下的遊戲狀態

表 8. 提供遊戲互動的 API

`void ponderStart()`	開始進行背景思考
`void ponderStop()`	停止背景思考
`void ponderCheckpage()`	在背景思考時，檢查是否分頁已經被使用完畢
`string ponderReport(Serializer)`	在不停止背景思考的狀態下，透過 Serializer 取得 UCT 的相關資訊。其原型為：`string serialize(root)`

表 9. 提供背景思考功能的 API

在這個軟體框架中，採用主從式架構，由一個主要執行緒(Master thread)控制多個從屬執行緒(Slave thread)。主要執行緒的工作包含處理 API、修剪搜尋樹以及控制從屬執行緒。從屬執行緒則是透過開發者提供的遊戲模組，進行 MCTS。

接下來會使用幾個情境，說明主從執行緒之間的互動。

1. 初始化。如圖 8 所示，在初始化時，主要執行緒會先將從屬執行緒產生出來，接著開始進行必要的初始化，最後等待所有從屬執行緒皆完成初始化；從屬執行緒在被產生後，立刻開始初始化自己，完成初始化之後通知主要執行緒。

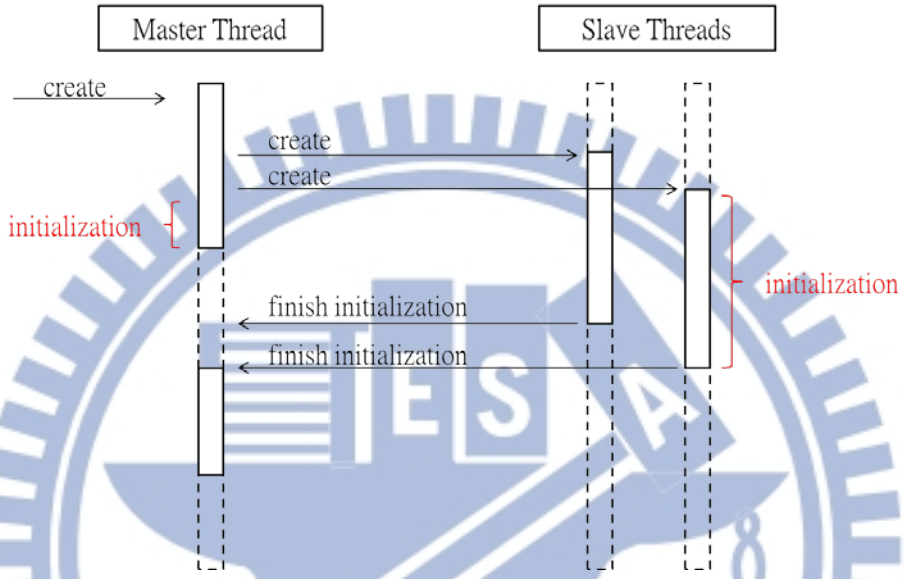


圖 8. 執行緒初始化

2. 產生棋步。如圖 9 所示，主要執行緒收到產生棋步的指令後，讓從屬執行緒開始進行運算，然後等待從屬執行緒運算完畢，選擇最佳的一步回應。而從屬執行緒根據設定的運算限制(如時間限制)，開始進行 MCTS，直到超過該限制。

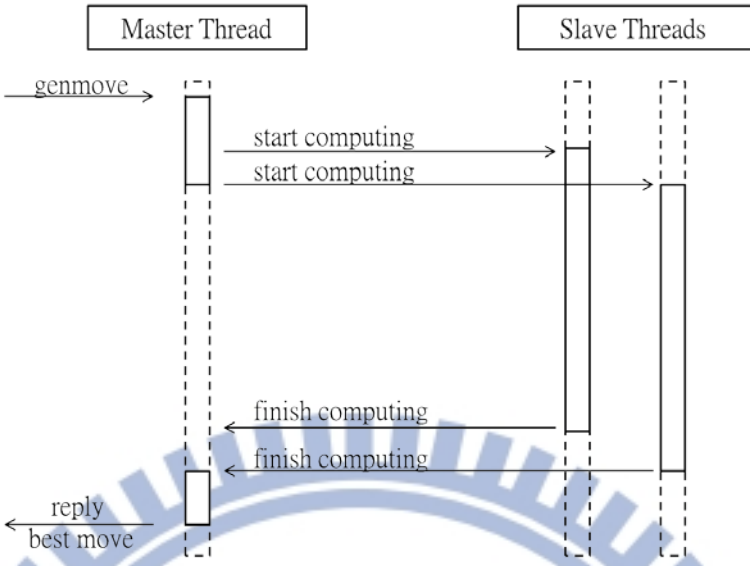


圖 9. 產生棋步流程

3. 修剪搜尋樹。如圖 10 所示，修剪搜尋樹發生在從屬執行緒思考過程中，當從屬執行緒發現所有的分頁都已經被使用完畢，則會暫停搜尋並將控制權交還給主要執行緒，待主要執行緒修剪完搜尋樹後，再繼續進行運算。

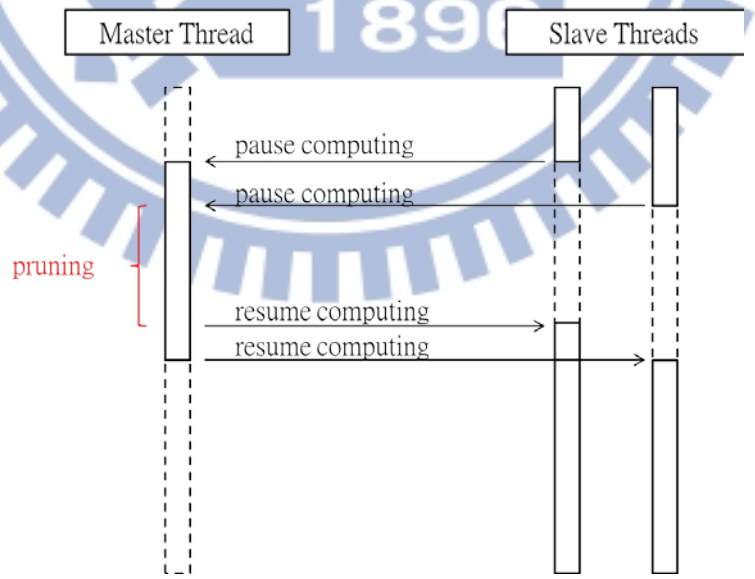


圖 10. 修剪搜尋樹

4. 背景思考。背景思考運作上是讓從屬執行緒在主要執行緒與使用者溝通時(含使用者思考時間)進行運算；如圖 11 所示，主要執行緒讓從屬執行緒開始思考之後，繼續和使用者溝通，直到收到會改變當下狀態的指令，才讓從屬執行緒停止背景思考。

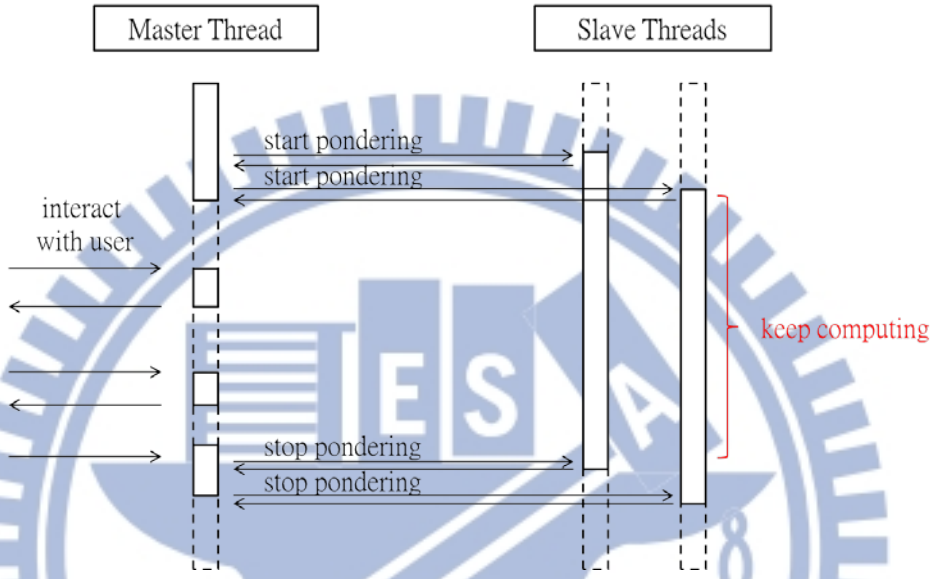


圖 11. 背景思考流程圖

3.4 分散式系統平行化(Distributed System Parallelization)

本節將詳細說明我們在分散式系統上的平行化作法。圖 12 為我們在分散式系統的平行化架構。分為伺服器端和工作者端兩部分：伺服器端負責和使用者溝通，並且管理後端的工作者；工作者則是實際進行運算的部分。

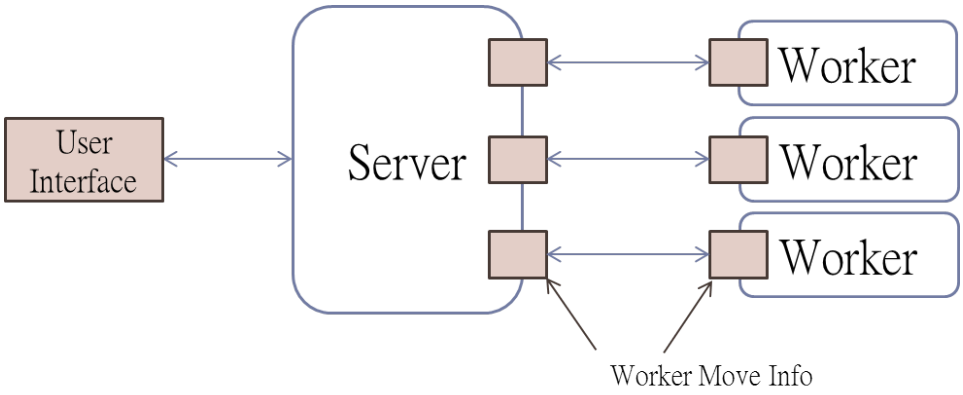


圖 12. 分散式系統平行化架構

3.4.1 協定(Protocol)

我們在伺服器端和工作者端之間，設計了以單行為基礎的協定來溝通。所有的指令以及回應都被限制在一行內。

由伺服器端送至工作者端的指令包含下面兩種：

- “clear\n”：清空盤面，開始新的遊戲。
- “play <move>\n”：下一子，其中<move>由開發者實作的CommandHandler產生

對於伺服器端送出的指令，工作者收到後需要回傳“=\n”作為回應，以確保伺服器端和工作者端看到的遊戲狀態是相同的。

另一方面，工作者端會主動定期向伺服器端回報當下搜尋的狀態，其格式為 “R <move_info>\n”，其中<move_info>由開發者實作的WorkerMoveInfo產生。

WorkerMoveInfo：用於操作工作者端搜尋狀態的類別。這個類別必須提供下列方法：

- string serialize (NodePtr)：將搜尋狀態(通常是 UCT 第一層節點的資訊)序列化成字串，產生送給伺服器端的報告。
- void parse (string)：解析工作者端傳來的資訊並儲存之。
- void merge (WorkerMoveInfo)：與其他的 WorkerMoveInfo 合併。

3.4.2 伺服器端(Server)

在分散式系統平行化中，我們假設使用者用標準輸入輸出與程式溝通，伺服器端僅負責與使用者溝通以及控制工作者；因此伺服器端所需要能非同步的處理三種事件：標準輸入有新的指令、工作者有新的訊息以及有新的工作者加入。

為了減少開發者實作非同步式輸入的困難，我們將其包裝進軟體框架中，開發者只需要實作處理指令的部分。

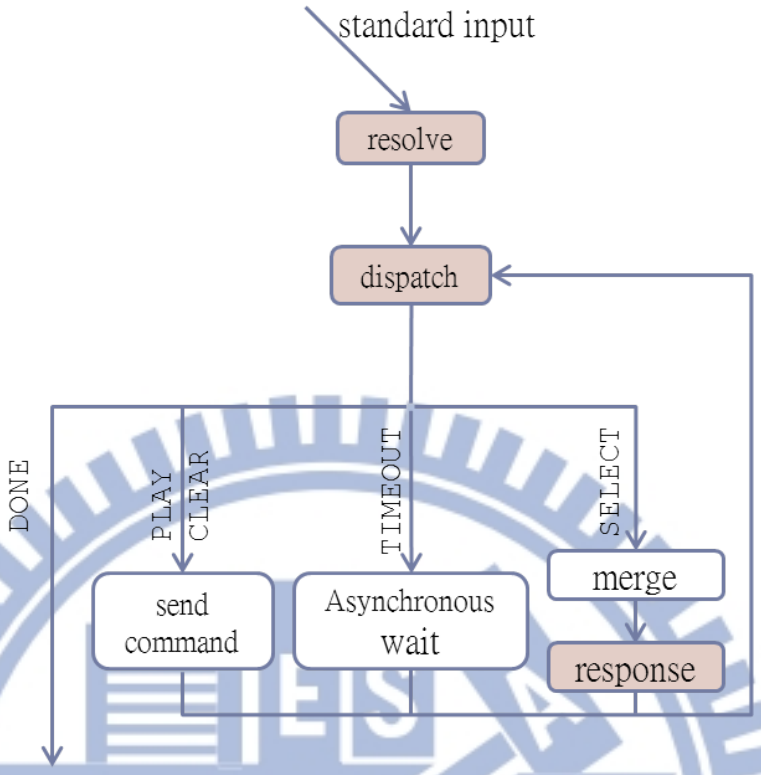


圖 13. 指令處理模型

圖 13 為我們處理指令的模型，開發者需實作 `CommandHandler`，作為事件的發送者，需實作的方法如下：

- `void resolve(string)`：解讀收到的指令，選取適當的有限狀態機(Finite State Machine)來處理指令。
- `Event dispatch()`：根據當下的狀態(指令處理的進度)來發送下一個事件給軟體框架，事件列表如表 10。
- `void response(WorkerMoveInfo)`：根據從各個工作者提供的搜尋狀態合併後的資訊，回應給使用者訊息。

事件名稱	後續行為
DONE	指令完成，接受下一項指令。
WAIT	等待一定時間後繼續處理這項指令。
PLAY	傳送“play <move>”給工作者端。
CLEAR	傳送“clear”給工作者端。
SELECT	合併工作者回報的搜尋狀態，交由 response 回應使用者。

表 10. 事件列表

如圖 14 所示，以產生一步為例。(1) 收到指令後進入 resolve()，選定需要的狀態機；(2) 等待一段時間讓工作者進行運算；(3) 合併工作者回報的搜尋狀態；(4) 選擇最好的一步並回應；(5) 通知工作者選擇的步；(6) 指令完成。

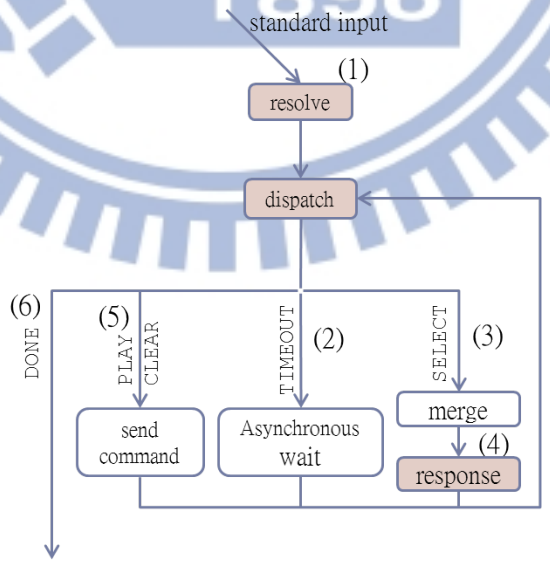


圖 14. 使用 CommandHandler 處理產生步指令

3.4.3 工作者端(Worker)

工作者端被啟動連上伺服器後，進入背景思考模式，除此之外只接受伺服器端的指令以及定時回報搜尋狀態。引此工作者端需要能非同步處力的事件有二：收到伺服器的指令時，暫停思考，待處理完指令後回復思考；定時利用 `WorkerMoveInfo` 將搜尋狀態序列化後傳送給伺服器端。

接下來使用幾個情境，說明伺服器端以及工作者端的互動。

1. 清除盤面，如圖 15。當玩家開始新遊戲時，伺服器端會傳送 `clear` 指令給工作者；工作者收到後，停下背景思考，使用 `newGame()`清空盤面，然後繼續背景思考並回傳訊息與伺服器端確認。

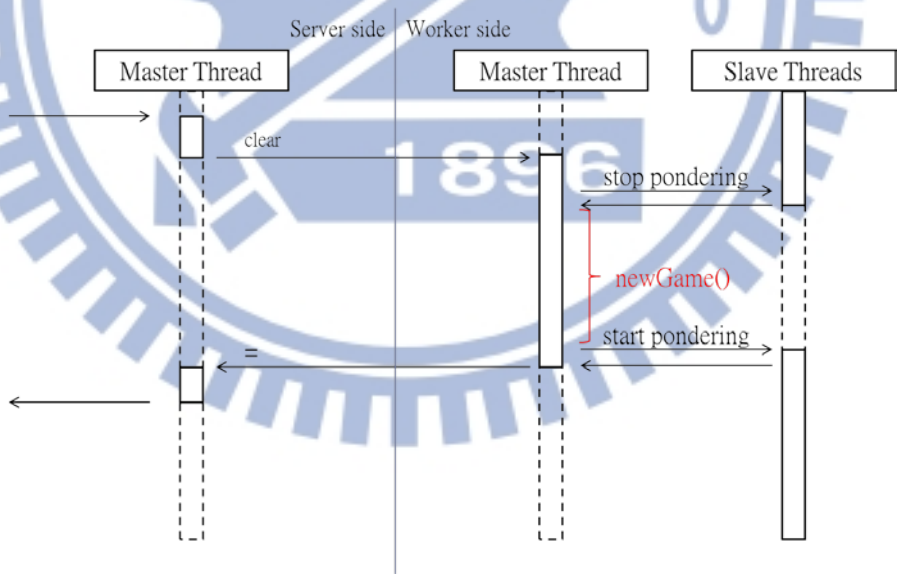


圖 15 伺服器端清除盤面流程圖

2. 改變盤面，如圖 16。當玩家決定動作或伺服器端選出最好步時，伺服器端傳送 `play <move>`指令給工作者；工作者收到後，停下背景

思考，使用 play()改變盤面，然後繼續背景思考並回傳訊息與伺服器端確認。

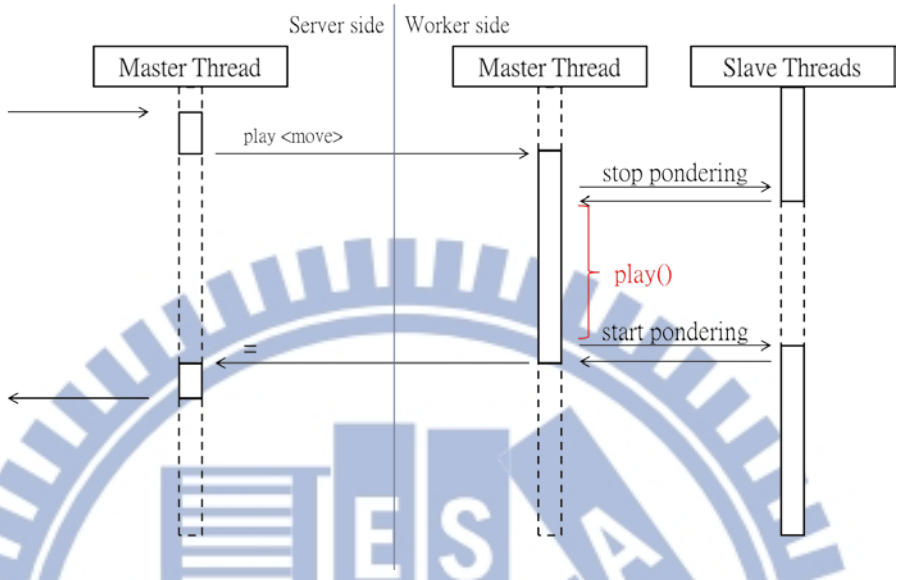


圖 16 伺服器端改變盤面流程圖

3. 回報訊息，如圖 17。每隔T_R秒，工作者主動使用 WorkerMoveInfo 將選點資訊轉換成字串，並使用 R <move_info>傳給伺服器端；伺服器端收到後，將該字串解析後並保留，待選點時使用。

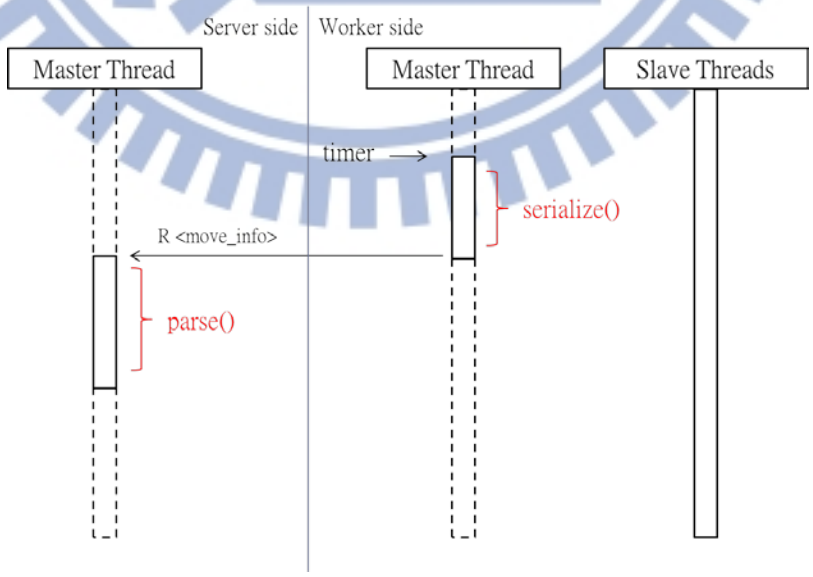


圖 17 工作者回報訊息流程圖

第 四 章 個案研究

在這個章節，我們會使用圍棋為例，說明如何快速地建立人工智慧；另以暗棋為例，說明如何實作部分資訊的遊戲人工智慧；最後比較各種遊戲與軟體框架所要花費的程式碼長度。

4.1 圍棋人工智慧實作範例

圍棋是一個雙人完美資訊遊戲;要實作出一個基本的圍棋人工智慧，開發者需要實作出 Move、GameState 等類別，提供這個軟體框架關於圍棋這個遊戲的相關知識；並實作使用者介面，與玩家互動。故實際需要實作的部分如下：

- GoMove：表達圍棋中一步代表的意義，黑或白、下在哪個位置。
- GoState：表達圍棋盤面狀態。
- GoUctAccessor：繼承 UctAccessor，處理跟 UCT 有關的操作。
- GoPlayoutAgent：繼承 PlayoutAgent，包含模擬的規則，並進行模擬。

若沒有額外的強化，要實際覆寫的函式如表 11：

GoUctAccessor::update(…)
GoPlayoutAgent::run(…)

表 11. 實作圍棋人工智慧所需複寫的函式

4.2 暗棋人工智慧實作範例

暗棋是一個雙人部分資訊遊戲，顏士淨等人提出了非確定性蒙地卡羅樹狀搜尋(Nondeterministic Monte Carlo Tree Search)來處理暗棋的不確定性[21]。若使用我們的軟體框架，可以使用兩層架構表達該篇論文中的非確定性狀態節點(Nondeterministic state node)，其對應如下圖。

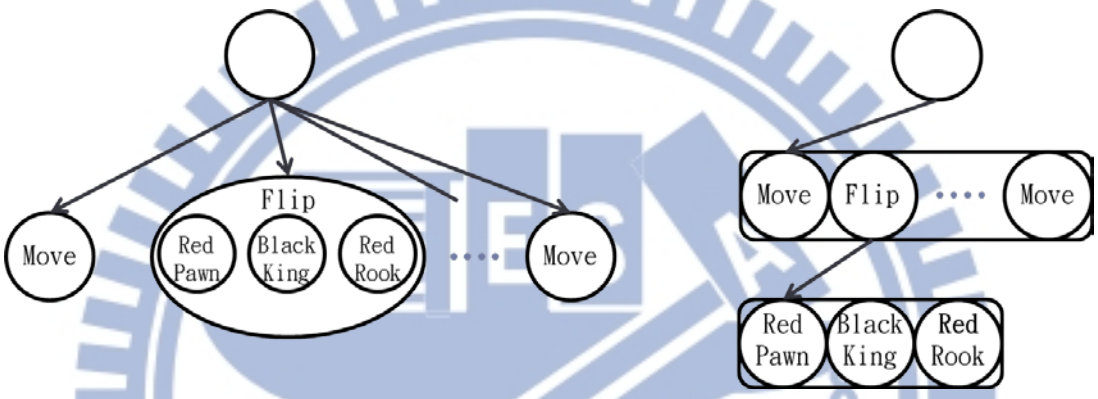


圖 18. 非確定性蒙地卡羅樹狀搜尋實作對應

因此若要套用我們的軟體框架實作暗棋人工智慧，實際需要實作的內容如下：

- DarkChessMove：表達暗棋中一步代表的意義，黑或紅、翻開哪顆棋子或是移動哪顆棋子。
- DarkChessState：表達暗棋盤面狀態。
- DarkChessUctNode：繼承 UctNode。加入記錄翻棋步的資訊，翻到某顆棋子的機率、是翻棋步或是移動步。
- DarkChessUctAccessor：繼承 UctAccessor。在選擇階段時，翻棋步根據翻棋子的機率選擇；並在展開階段設定翻棋步的訊息。

- DarkChessPlayoutAgent：繼承 PlayoutAgent。包含模擬的規則，並進行模擬。

根據上述做法，需要覆寫的函式如表 12：

DarkChessUctAccessor::select(…)
DarkChessUctAccessor::expand(…)
DarkChessUctAccessor::update(…)
DarkChessPlayoutAgent::run(…)

表 12. 實作暗棋人工智慧所需複寫的函式

4.3 比較

除上述遊戲外，我們亦實作了一種單人遊戲–8 puzzle，以及作為基本範例的井字遊戲。統計各項實作所需要花費的行數如表 13。實際完成井字遊戲人工智慧只需要約 300 行的程式碼，相較於軟體框架本身，程式碼長度達 5559 行，有相當的差距。

	Parallel MCTS Framework	Go	Dark Chess	8 Puzzle	Tic-Tac-Toe
Code length	5559 lines	~2500 lines	~1500 lines	~500 lines	~300 lines

表 13. 遊戲人工智慧行數統計

第 五 章 實驗

在這個章節中，我們會對這個軟體框架進行一些效能的測試。我們所做的實驗，會使用圍棋人工智慧的實作—Amigo；Amigo 是將本實驗室開發多年的圍棋程式 HappyGo 移植至平行化環境後的程式。

實驗所使用的機器為國家高速網路與計算中心(National Center for High-Performance Computing)提供的機器。每台機器有四顆 12 核心的 CPU – AMD Opteron Processor 6174，共 48 核心，時脈為 2.2GHz，搭配 128G 的記憶體，作業系統為 Linux。

5.1 共享記憶體架構平行化

在共享記憶體架構下，我們做了兩項實驗；一是測試在無鎖樹平行下，多個執行緒進行 MCTS 是否能確實的提升效能；另一項則是測試平行化能夠確實的提升其強度。

5.1.1 效能測試

實驗測試在相同的時間限制(10 秒)下，不同執行緒數量，能夠進行多少次的 Simulation；以單執行緒的數量為單位，分析提升執行緒數量後，加速率可以達到多少，以及每個核心的平均效率。實驗結果如表 14：

核心數	1	2	4	8	12	16	24	36	48
速度	7000.25	13582.8	26077.9	51141.3	75659.9	98286	143369	208526	252562
加速率	1.00	1.94	3.73	7.31	10.81	14.04	20.48	29.79	36.08
效率	1.00	0.97	0.93	0.91	0.90	0.88	0.85	0.83	0.75

表 14. 共享記憶體架構平行化效能比較

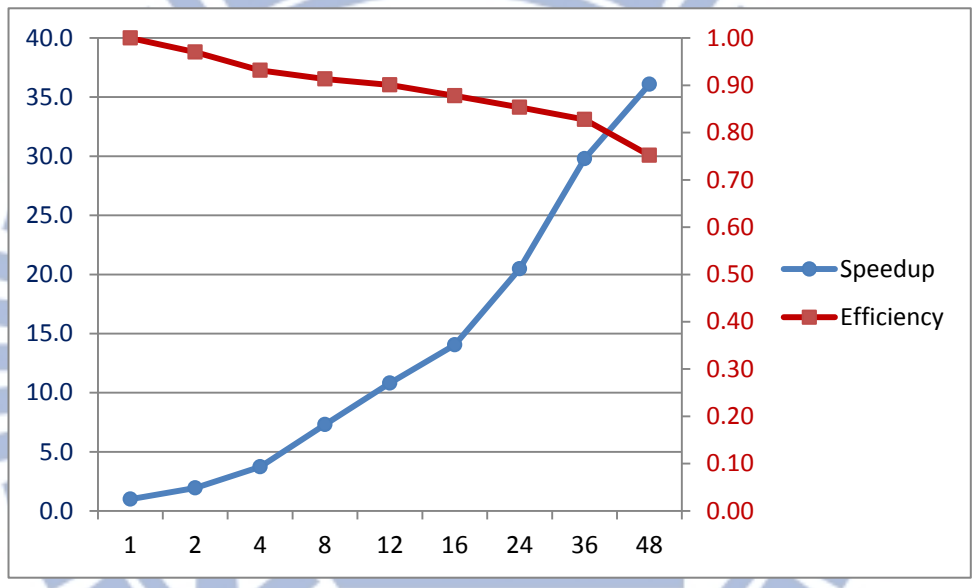


圖 19. 共享記憶體架構平行化效能比較

由圖 19 可以看到，在使用了 48 核心的狀況下，能夠達到 36 倍的加速，相當於每個核心提供了 75%的戰力。若使用較少的核心數，其效能可以達到 90%以上。

5.1.2 強度測試

這個實驗測試在每手一秒的限制下，與單一執行緒版本進行九路圍棋

的比賽。觀察不同執行緒數量帶來的強度變化，勝率變化如表 15。

核心數	1	2	4	8	12	16	24	36	48
勝率	51%	66%	85%	89%	93%	94%	92%	95%	94%

表 15. 共享記憶體架構平行化強度比較

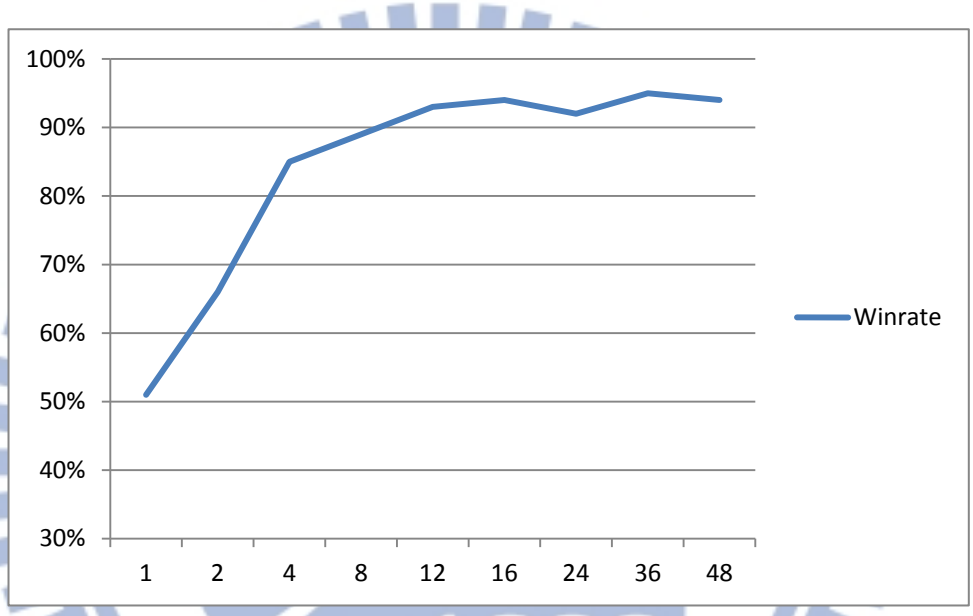


圖 20. 共享記憶體架構平行化強度比較

由實驗數據可以看到，從單核心到 12 核心，勝率有很明顯的提高；由於 12 核心對單核心勝率已超過 90%，超過 12 核心後，便看不出明顯的成長。

5.2 分散式系統平行化

在分散式系統的實驗中，我們測試在不同的機器數量下，是否能夠順增加程式的強度。

由於在分散式系統中會牽涉到網路的延遲，我們將設定每一手的思考時間為五秒，藉此將低網路延遲可能造成的影響。對戰對手選用 Fuego，思考時間也為五秒。其勝率變化如表 16：

機器數	1	2	4	8
勝率	23%	33%	50%	68%

表 16. 分散式系統平行化強度比較

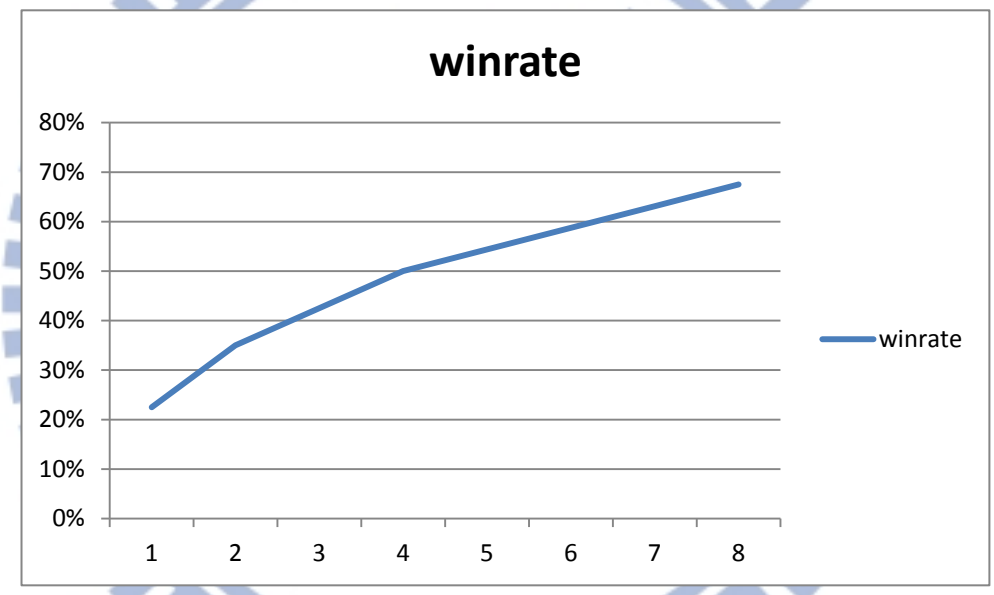


圖 21. 分散式系統平行化強度比較

由勝率變化圖中可以看到，透過軟體框架提供的分散式系統平行化，增加機器的數量可以確實的提高勝率。

第 六 章 結論

本篇論文設計並實作出了一個基於平行化蒙地卡羅樹狀搜尋的軟體框架；利用這個軟體框架，遊戲人工智慧的開發者能夠快速地建立他們的人工智慧，並專注在專家知識的建立。

在共享記憶體架構下，我們實作了無鎖樹平行；在使用 48 個核心時，仍有 75%的效率，並且有效率的提升人工智慧的強度。並在分散式系統上，我們實作了根平行；藉由分散式系統的平行化，能夠大幅度的提升人工智慧的強度。

本軟體框架已經成功地應用至雙人完美資訊遊戲、雙人部分資訊遊戲以及單人遊戲，並取得下列成果。

- 應用至圍棋上，透過分散式平行化，平行至 16 台機器，每台使用 36 核心，共 576 核心同時進行運算，在十七屆電腦奧林匹亞電腦對局競賽中獲得第四名。
- 應用至禁圍棋，使用共享記憶體平行化，使用 8 核心，在十七屆電腦奧林匹亞電腦對局競賽中獲得第一名。
- 應用至暗棋，使用共享記憶體平行化，使用 7 核心，在 2013 TCGA 電腦對局競賽中獲得第四名。

在未來展望的部分，有兩項工作需要繼續研究與加強。為了更進一步的驗證軟體框架的通用性，將其套用至三人以上的遊戲人工智慧如三軍棋等，以及較複雜的最佳化問題如零工式工廠排程問題(Flexible Job Shop

Scheduling Problem)、火力機組發電排程問題(Unit Commit Problem)等。另外一方面，在分散式系統中，需設計更一般化的通訊協定，使開發者能更有彈性讓工作者交換部分資訊，以獲得更佳的平行化成果。

參考文獻

[1] Arneson, B., Hayward, R. B., and Henderson, P., Monte Carlo Tree Search in Hex, *IEEE Transactions on Computational Intelligence and AI in Game*, Vol. 2, No. 4, December 2010.

[2] Baudiš P., Gailly J.-L., PACHI: State of the Art Open Source Go Program, *13th International Conference, ACG 2011*, Tilburg, The Netherlands, November 20-22, 2011.

[3] Boost Librarues, http://www.boost.org/

[4] Browne, C., Monte Carlo Tree Search, http://mcts.ai/.

[5] Bruegmann B., *Monte Carlo Go*. White paper, 1993.

[6] Chaslot G.M.J.B., Winands M.H.M., and H.J. van den Herik. Parallel monte-carlo tree search. *Proceedings of the Conference on Computers and Games* 2008.

[7] Couetoux , A., Hoock, J.-B., Sokolovska N., Teytaud O., and Bonnard N., Continuous Upper Confidence Trees, *Proceedings of the 5th International Conference on Learning and Intelligent OptimizatioN (LION'11)*, Italy, 2011.

[8] Coulom, R., Efficient selectivity and backup operators in monte-carlo tree search. In P. Ciancarini and H. J. van den Herik, editors. *Proceedings of the 5th International Conference on Computers and Games*, Turin, Italy, 2006.

[9] Coulom, R., Monte Carlo tree search in crazy stone. *Game Programming Workshop*, pp. 74–75, Tokyo, Japan, 2007.

[10] Drake P., The Last-Good-Reply Policy for Monte-Carlo Go, *International Computer Games Association Journal, Vol. 32, No. 4,* pp. 221-227, 2009.

[11] Enzenberger M., Muller M., Fuego - An Open-source Framework for Board Games and Go Engine Based on Monte-Carlo Tree Search, *IEEE Transactions Computational Intelligence and AI in Games,* 2009.

[12] Finnsson H. and Bjornsson Y., Simulation-Based Approach to General Game Playing, *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence,* 2008.

[13] Gelly S., Wang Y., Exploration Exploitation in Go: UCT for Monte Carlo Go. In: *Twentieth Annual Conference on Neural Information Processing Systems (NIPS 2006),* 2006.

[14] Gelly, S., and Silver, D., Monte-Carlo tree search and rapid action value estimation in computer Go, *Artificial Intelligence, pp. 1856–1875,* 2011.

[15] Huang S.-C., Coulom R., and Lin S.-S., Monte-Carlo Simulation Balancing in Practice, *Proceedings of the 7th international conference on Computers and games,* pp. 81-92, 2010.

[16] Hunter D. R., MM algorithms for generalized Bradley-Terry models, *The Annals of Statistics,* Vol. 32, No. 1, pp. 384–406, 2004.

[17] Ikehata N. and Ito T., Monte-Carlo Tree Search in Ms. Pac-Man, *IEEE Conference on Computational Intelligence and Games (CIG),* 2011.

[18] Lorentz , R. J., Improving Monte-Carlo Tree Search in Havannah, *Computers and Games Lecture Notes in Computer Science,* Vol. 6515, pp.

105-115, 2011.

[19] Metropolis N., Ulam S., The Monte Carlo method. *Journal of American Statistical Association,* Vol. 44, Issue 247, pp. 335-341, 1949.

[20] Wu, T.-Y., *Multi-Objective Flexible Job Shop Scheduling Problem Based on Monte-Carlo Tree Search*, Master Thesis, National Chiao Tung University, 2013.

[21] Yen, S.-J., Chou, C.-W., Chen, J.-C., Wu, I-C., Kao, K.-Y., The Art of the Chinese Dark Chess Program DIABLE, *Proceedings of the International Computer Symposium ICS, 2012*.