國 立 中 央 大 學

資 訊 管 理 學 系

碩 士 論 文

以 Solidity 語言實作之多對多配對雛型系統

研 究 生：顏 裕

指導教授：蔡明宏 博士

蘇雅惠 博士

中 華 民 國 一〇六 年 六 月

國立中央大學圖書館
碩博士論文電子檔授權書

(104 年 5 月最新修正版)

本授權書授權本人撰寫之碩/博士學位論文全文電子檔(不包含紙本、詳備註 1 說明)，在「國立中央大學圖書館博碩士論文系統」。(以下請擇一勾選)

(ｖ)同意 (立即開放)

(　)同意 (請於西元 _____年_____月_____日開放)

(　)不同意，原因是：_____

在國家圖書館「臺灣博碩士論文知識加值系統」

(ｖ)同意 (立即開放)

(　)同意 (請於西元 _____年_____月_____日開放)

(　)不同意，原因是：_____

以非專屬、無償授權國立中央大學、台灣聯合大學系統圖書館與國家圖書館，基於推動「資源共享、互惠合作」之理念，於回饋社會與學術研究之目的，得不限地域、時間與次數，以紙本、微縮、光碟及其它各種方法將上列論文收錄、重製、與利用，並得將數位化之上列論文與論文電子檔以上載網路方式，提供讀者基於個人非營利性質之線上檢索、閱覽、下載或列印。

研究生簽名: 顏裕　　學號：104423043

論文名稱: 以 Solidity 語言實作之多對多配對雛型系統

指導教授姓名：蔡明宏　博士　　蘇雅惠　博士

系所　：資訊管理所　□博士班　■碩士班

填單日期：2017.06.28

備註：

1. 本授權書之授權範圍僅限**電子檔**，紙本論文部分依著作權法第 15 條第 3 款之規定，採推定原則即預設同意圖書館得公開上架閱覽，如您有申請專利或投稿等考量，不同意紙本上架陳列，須另行加填申請書，詳細說明與紙本申請書下載請至本館數位博碩論文網頁。
2. 本授權書請填寫並**親筆**簽名後，裝訂於各紙本論文封面後之次頁（全文電子檔內之授權書簽名，可用電腦打字代替）。
3. 讀者基於個人非營利性質之線上檢索、閱覽、下載或列印上列論文，應遵守著作權法規定。

國立中央大學碩士班研究生

論文指導教授推薦書

資訊管理學系/研究所　顏　裕　研究生所提之論文

以 Solidity 語言實作之多對多配對雛型系統

係由本人指導撰述，同意提付審查。

指導教授 蔡明宏 蘇雅惠 (簽章)

106年 6月 2日

104.07.24

國立中央大學碩士班研究生

論文口試委員審定書

資訊管理學系/研究所　顏　裕　研究生所提之論文

以 Solidity 語言實作之多對多配對雛型系統

經本委員會審議，認定符合碩士資格標準。

學位考試委員會召集人　陳和志

委　　員　蔡明宏

蘇雅惠

中華民國 106 年 6 月 2 日

101.06.15

以 Solidity 語言實作之多對多配對雛型系統

國立中央大學 資訊管理學系 顏　裕

摘要

區塊鏈是一項與交易相關且正在發展中的技術，並等待其的應用能夠多元的蓬勃發展，配合良好的商業模式來開發新的商機，帶來嶄新的生活改變。交易在生活中有很多配對的例子與解決辦法。本研究著重於多對多的配對，對於每個使用者皆有需求的喜好順序，根據 Gale – Shapley 的極大化配對原則，連續性的搜尋最好的配對組合，讓多對多的交易能夠良好互動，得到最大的利益。

本研究針對多對多配對如何能夠運作在區塊鏈的技術上，實現 Gale – Shapley 的延遲接受演算法，以 Solidity 語言在區塊鏈上部署智能合約，蒐集系統的結果，配對的使用者仍可有同意或否決結果的選擇，並重新作分配，本研究後段會對於系統的結果進行分析，是否確實為最佳的配對組合，也繼續探討未來發展的可能性。

關鍵詞：多對多配對、區塊鏈、交易、Solidity、智能合約、Gale–Shapley、延遲接受演算法

Implementation of prototypical system to solve matching problem by using Solidity language

Abstract

Blockchain is an ongoing technology which is related to transactions, decentralized ledger, and encryption. And it is expected to have more applications developing in diversified way gloriously. With creating good business models will bring huge opportunities and change people's life. There are many solutions for matching problems of transaction in our daily life that this study focus on. For each user's order of preference, we want to find out the best match by using Gale-Shapley's matching principle. To get the greatest benefit, we use continuous searching for best pairs to make transactions in a better interaction.

In the study, the goal is to implement Gale-Shapley algorithm in the blockchain technology to solve the matching problem. The smart contract which is written in Solidity language will be deployed in the blockchain and prototype system will collect the results. Users own the option of agreeing or rejecting the results, and then wait for result of rematching if they choose to reject. The study will analyze the results of matching system and check whether they are best pairs or not. It also explores the further developing possibility in the future.

Keywords : Matching problem, Blockchain, Transaction, Solidity, Smart contract, Gale-Shapley algorithm

致謝

本論文的完成，最要感謝的是蘇雅惠老師，安排了我給蔡明宏老師指導的機會，讓這份論文及研究的過程中是由兩位老師共同指導來產生的，在初期的題目討論時給予我很多參考，更提供我許多新科技知識的方向。而當確定研究領域及方向後，非常感謝蘇雅惠老師陪同我去參加研討會，並協力聯絡相關領域的資源與人脈。而一路走來，兩位老師以嚴謹與專業的態度協助本研究的討論，老師們非常有耐心的講述相關的知識，並指導我在研究上的不足，給予許多寶貴的建議。

另外要特別感謝研究室的戰友們，雖然每次開會都很擔心自己的進度以及研究的發展，但我們還是在彼此的加油打氣下，渡過了一個又一個的關卡，順利完成了口試，更要感謝的是謝榮軒，協助我一起研究相關領域的知識及文獻，也在我建立測試環境的時候有很大的幫助。

再來要感謝的是我的家人，謝謝他們當初支持我進修研究所的決定，也在研究過程中成為我的精神支柱。

最後我僅以本論文獻給我的母親。

目錄

圖目錄

表目錄

1. 緒論

本章節主要介紹目前研究的背景，並透過其說明研究動機，最後指出研究的目的與重要性。

1-1 研究背景

Satoshi Nakamoto 於 2008 提出了比特幣的協定與其的相關軟體，而比特幣是一種電子式的貨幣，並且透過協定有它特殊的交易技術，雖然作者發表論文以來它的真實身分外界無從得知，但無疑的是，這項發明對當時到現在的科技發展投下了一顆不小的震撼彈，而這不僅是它的交易技術，比特幣作為一個記帳系統，它核心思想是去中心化，也就是比特幣是不由某個中央機構或單位作發行、維護交易等行為，上述提到的工作皆由網路中的參與者節點協力去完成，機制上透過了數位的加密演算法來保障交易上的安全性，另外交易的紀錄也被整個網路中個個節點儲存作維護，並且在每筆交易在進行時，該電子貨幣的有效性都必須經過檢驗的合格來取得確認，來避免發生雙花攻擊以及其他可能的漏洞。

區塊鏈是比特幣所提出的重要概念之一，也在 Satoshi Nakamoto 於 2008 年所提出的白皮書中所提及，區塊鏈是一串由是用數位密碼學所產生出來與資料相關的區塊，每筆所新增的區塊會連結到上一個區塊，而這累積所形成交易歷史紀錄就好比是一條條的區塊鏈，並且是透過分散式資料庫的概念，儲存於網路中的各節點中，Satoshi Nakamoto 也在 2009 創立了比特幣的社會網路，開發出第一個區塊，即創世區塊。區塊鏈運用對等網路和開源軟體來將密碼學、共識機制、時間戳記來作結合，以保障網路的各節點中，能保持區塊鏈的完整性和正確性，資訊也能夠追朔，最重要的是難以篡改，創造了擁有隱私、安全、效能的交易模式與體系。

現今的區塊鏈的技術已有許多的環境平台與協定機制，除了最早為 Satoshi

Nakamoto 於 2008 提出了比特幣及其白皮書外，另有以太坊(https://www.ethereum.org/)和由國立台灣大學發展的 Gcoin(http://g-coin.org/)，都為了這項技術以及其背後的終極理念付出了努力。

從區塊鏈所衍生探討的一個思想：去中心化，或許是人類未來科技發展的一個趨勢與項目，以金流作一個例子，現今我們無論在做任何的交易，當與金錢有關的時候，必須透過一個中間的機構 (通常是銀行的角色) 來做交易的維護，它保障的是這個交易的安全性與確認性，但如果以區塊鏈來發展金流的情況，由於它本身就已透過網路節點們的參與來認證這筆交易，因此再也不需要中介單位的存在，這就是去中心化的概念，也是影響科技未來發展的一個可能。區塊鏈的概念是跟隨著比特幣的誕生而出現，產生的應用也多是與金融所相關，亦即非常流行的金融科技(FinTech)，大多是銀行透過這項科技，讓企業的金融服務更有效率和安全性，而形成的一種經濟產業，但其實區塊鏈可以的應用不只如此，非金融式的應用更能彰顯區塊鏈的核心價值，從版權的認證、學歷病例的認證...等等，都是可以透過區塊鏈所提供的功能來達到，而前述所提到的區塊鏈基礎層已經有許多的平台所提供，並且也有與其溝通的智能合約和腳本語言，這讓開發的參與者可以花更多的心力在思考、創造區塊鏈上的應用，無論是金融或是非金融，都將是改變人類社會的發明。

1-2 研究動機

生活中有許多的需求需要被解決，大多數的交易是以多對多配對的形式等待被處理，如：群眾募資、考試分發、實習生應徵、換腎、購物...等，而 Lloyd S. Shapley 和 Alvin E. Roth 所提出的延遲接受演算法，提供了多對多配對的實現方法與步驟，得到最好的配對組合。

但在智能科技的的發展的現今，於區塊鏈上則是欠缺的，區塊鏈擁有匿名性與交易的確認性，使得交易能夠完整的被記錄，並且也確保其安全性。

1-3 研究目的

根據上述之原因，本研究希望可以於區塊鏈技術上，建立一個多對多的配對實作，並且試圖將複雜的生活中的配對問題轉化簡單，本研究將以四對四的婚姻配對問題來做範例，每位使用者以喜好度來表示其希望配對的列表，當所有人皆以排序好自己的喜好度後，系統會透過延遲配對演算法的反覆步驟，來得到最佳的配對組合，使用者可以確認配對的結果是否符合自己的期望，符合且確認後，區塊鏈便扮演儲存的腳色，以上研究結合區塊鏈的特性：匿名性與確認性，藉此改善原有交易的品質。

主要分為下列幾點要向：

- 了解區塊鏈與智能合約的原理和運作。
- 了解 Gale–Shapley 演算法的步驟。
- 學習開發 Solidty 語言，以編寫智能合約。
- 於區塊鏈技術上，建立一個多對多的配對實作。
- 驗證結果是否正確，以及比較系統的差異。

1-4 研究重要性

多對多配對是一個常見的交易模式，無論是在實體市場或是電子市場，區塊鏈的發展則是處理交易的信任機制，以及分散式的儲存方式，這是現今金融科技的發展的

趨勢，但其發展出來的應用是否僅限於金融相關？這答案肯定是否定的，對於多對多的配對交易，更廣大的需求市場是在非金融層面的應用，而這些應用的範例已經在我們的日常生活中被大量的使用，並且也有相當多的社會實驗所支持，包括了群眾募資，一方透過媒體宣傳自己的計畫與內容，另一方做為投資者提供資金去促使完成該計畫，這正是典型的多對多配對；聯考分發亦相同，學校和學生就好比是買方與賣方，或許多多少少會有金錢的交易參與其中，不過這些主要為非金錢類的交易模式是多對多配對的主要訴求。另外值得一提的是換腎的配對，實際上卻存在著與市場並行的黑市，這也指出很多一般市場的缺陷，而以區塊鏈的技術來完成的配對與交易，是有著訊息完全透通的特性，並且擁有信任的機制，這項改變是能夠讓市場上的交易有不同的變化，改變原有的交易模式。

本研究希望能透過區塊鏈上實作多對多配對，讓交易的資訊及機制因為區塊鏈的特性：資訊的透通性、強大的信任機制，來試圖改變原有的交易模式，多對多的配對需求是一直都存在，而在區塊鏈上實作這個演算法是一個必要的發展與挑戰。

1-5 論文架構

本研究主要架構分別為緒論、文獻探討、系統架構與分析、系統驗證與討論、研究成果。緒論：介紹研究的背景，並說明研究的動機，從而導出本研究的目的與重要性，描述本研究的重要價值。文獻探討分為三個類別：區塊鏈、智能合約與多對多配對的延遲接受演算法，區塊鏈描述原理與平台間的比較；智能合約則介紹功能與在區塊鏈技術上的應用；多對多配對是講述延遲接受演算法的步驟，並且在幾個範例與數學邏輯來做解釋。系統架構與開發為對於本研究的多對多配對雛形系統做架構的解

說，並透過使用案例圖、流程圖與循序圖來闡述本研究系統的事件和流程，並在開發的章節內說明智能合約和網頁端程式的功能函式。系統驗證與討論說明系統的操作簡易流程並驗證系統所產出的結果是否與一般的延遲接受演算法算出的答案一致，且透過討論智能合約編寫上的困難及其對應解決方式，來給予未來程式面可以的發展選擇。研究成果與未來方向則談到本研究所做的系統是否有達到預期的成效，並討論研究遇到的限制和本研究能帶來可能的發展。

2. 文獻探討

本章節為相關文獻與資料的探討，主要分為區塊鏈、智能合約，以及多對多配對。區塊鏈講述其起源及原理，並介紹相關的發展平台及可發展之產業應用。智能合約除了介紹起源與功能外，並介紹 Solidity 語言的開發。最後多對多配對的部分說明延遲接受演算法的步驟與詳細細節。

2-1 區塊鏈

2-1-1 區塊鏈之技術原理：以比特幣為例

Satoshi Nakamoto (2008) 提出了比特幣的概念以及其演算法 (NAKAMOTO, S , 2008, BITCOIN: A PEER-TO-PEER ELECTRONIC CASH SYSTEM)，對電子交易給予了不同的應用方式與協定，他認為網路上的商業幾乎完全仰賴於中央的金融機構，來作為第三方處理的電子支付，雖然這樣的做法到現今對於大多數的交易效率一直都很好，但仍受限於信任模式的弱點，對於交易的可逆性，由於調解成本增加意味著交易成本的增加，金融機構也無法完全避免調解的爭議，使得最小的實際交易規模受到了限制，導致小型的交易無法產生，隨著信任機制的需求逐漸擴大，交易雙方必須更小心地面對交易，例如蒐集更多的資訊來預防詐騙的可能性，然而這些蒐集資訊的費用和付款的不確定性，能夠透過使用實體的貨幣來有效率的預防詐騙的問題，但卻沒有機制是用於在電子市場貨幣交易的預防。基於電子密碼加密學所衍生的電子支付系統，是允許交易雙方直接地做電子貨幣地交換，並且是不需要身為第三方信任的中央單位

或是機構，在這篇文章中提出了保護買賣雙方的機制，以點對點的分散式資料庫概念，來防止雙花攻擊的發生，並透過時間戳記的伺服器，以交易生成的時間做計算證明，來完成一次又一次的安全交易。

交易會以上一個交易與公鑰，透過雜湊法的方式產出，並用私鑰去做驗證的動作，如圖 1 所示。

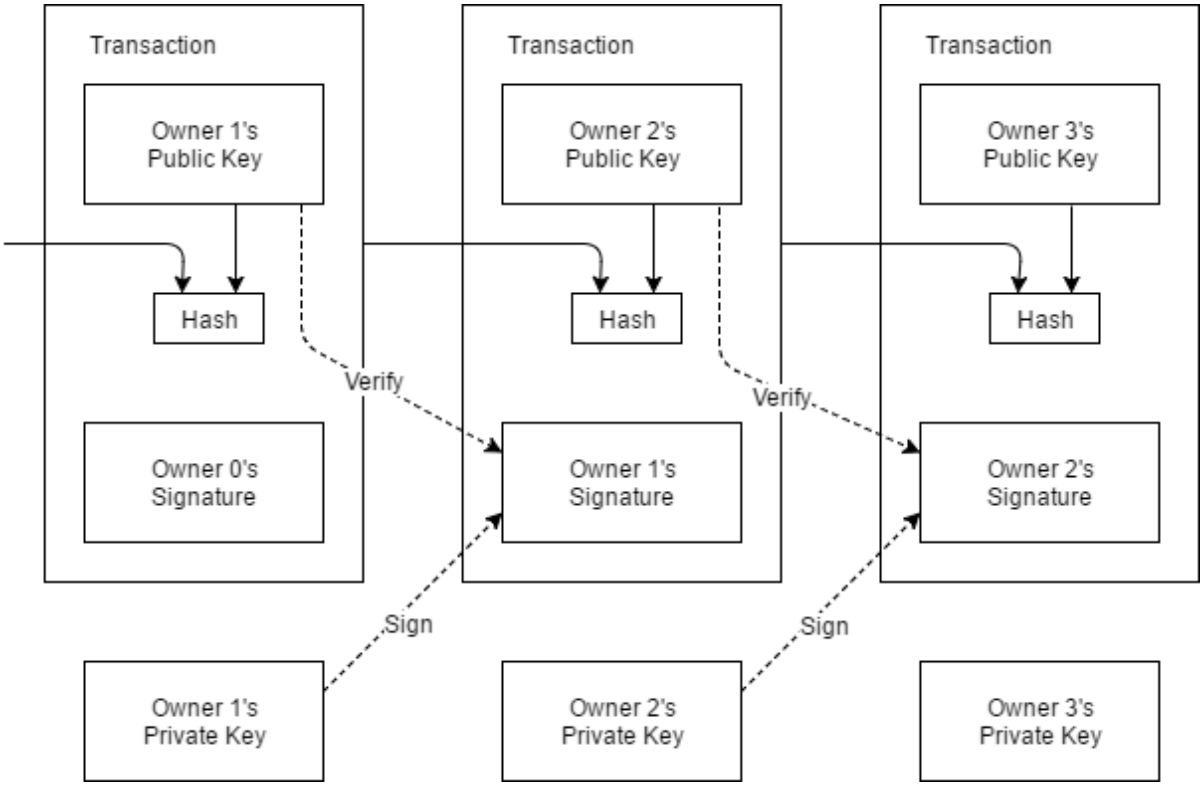

圖 1　電子貨幣的數位簽章

資料來源：(Nakamoto, 2008)

後續章節則提到，透過時間伺服器，來添加時間戳記的參數，而每次雜湊一樣必須包含上一個區塊，如圖 2 所示。

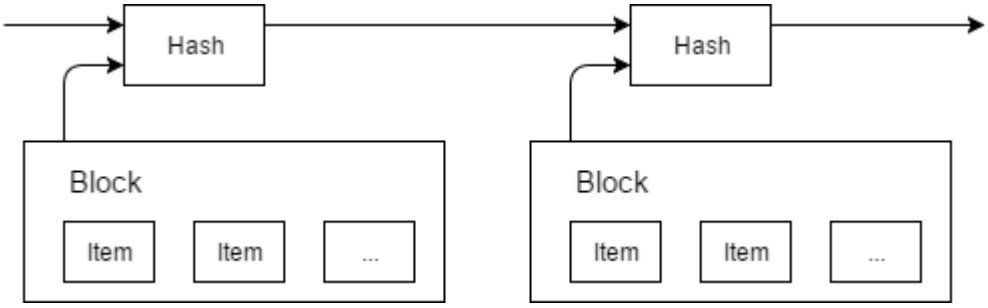

圖 2　區塊添加時間戳記並雜湊

資料來源：(Nakamoto, 2008)

區塊鏈技術特色在於工作量證明，必須做一定難度的工作來得出一個結果，驗證方法卻是相對很簡單，若結果不符則須重新運算再得出另一個結果，直到節點中有得出符合條件的結果，如圖 3 所示。

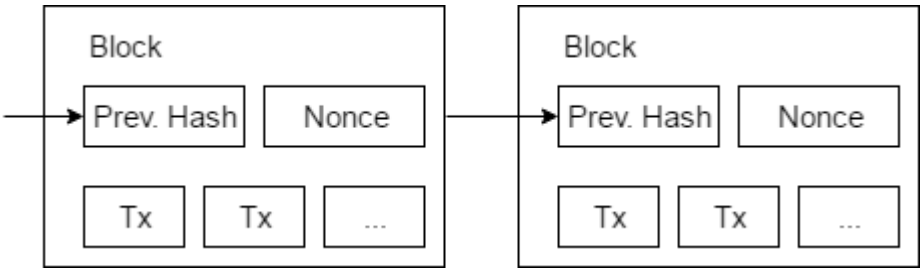

圖 3　工作量證明

資料來源：(Nakamoto, 2008)

隱私的議題在區塊鏈上特別被提到，指出傳統銀行模型為高隱私層級，限制存取多數的資訊以及交易內容。區塊鏈由於只要保持公鑰的匿名，便可在交易的過程中做到與傳統模型一樣的隱私層級，但交易的紀錄卻又是公開透明的，如圖 4 所示。

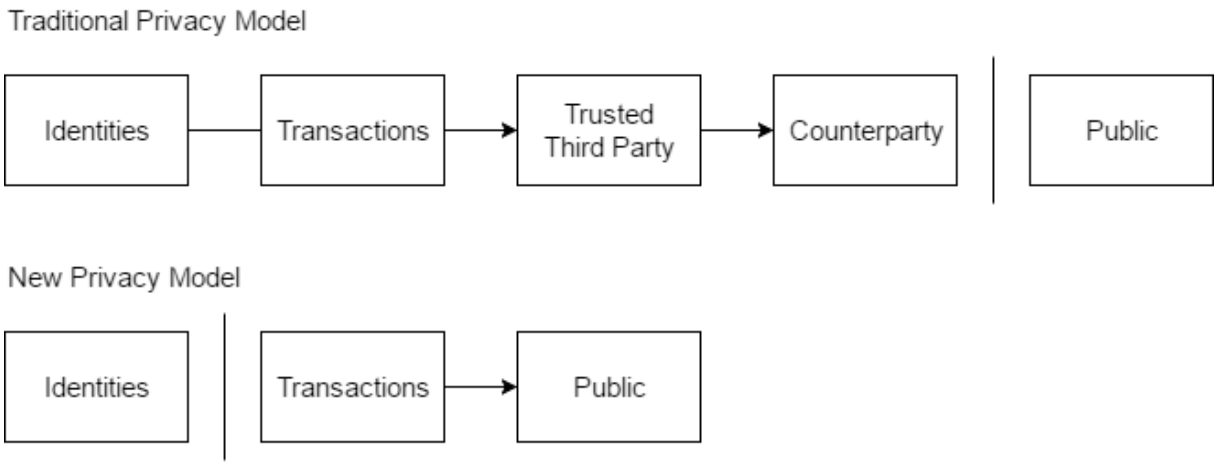

圖 4　比特幣交易隱私

資料來源：(Nakamoto, 2008)

對於區塊鏈技術專有名詞，整理如表 1。

表 1　區塊鏈技術專有名詞

專有名詞	解釋
區塊鏈	分散式儲存、點對點傳輸、共識機制、加密演算法等技術所共同衍生的新型應用模式。
分散式	相對於集中式，分散式是區塊鏈的一大特徵，不依賴於中心的伺服器，或是利用分散的計算資源來運轉的模式。
金融科技	透過科技技術來讓金融的服務更完備與效率，亦稱 FinTech。
電子貨幣	貨幣數位化，透過資料交易，並發揮交易媒介、記帳單位與價值儲存的功能，不屬於任何國家或地區的貨幣。
共識機制	為了達成在分散式系統下，不同節點間，對於資料或區塊的一致認同，需要的共識演算法。因為過程中可能包含網路延遲、節點故障或是惡意攻擊導致資料不一致性。

專有名詞	解釋
智能合約	透過程式碼去模擬合約的邏輯，利用機器對資料判讀的一致性與自動化達到快速的合約驗證。
挖礦	電子貨幣系統中，透過運算能力爭取記帳權，從中獲得獎勵的行為。是比特幣中用來達成共識機制的手段之一。 註：並非所有區塊鏈都有挖礦行為。
分散式帳本	一個可以儲存於多個節點、不同地點或單位的共享資料庫，資產可以是貨幣、實體，或是電子資產。

資料來源：本研究整理

2-1-2 區塊鏈之應用：金融與非金融應用

區塊鏈是分散式儲存、點對點傳輸、共識機制、加密計算等技術在互聯網的創新應用模式，也被認為是一項突破式的創新，很有可能對人類造成技術性的革新與產業的變革，許多國家都對於區塊鏈的發展給予高度的關注，並積極的探索和推動區塊鏈相關的應用，目前區塊鏈的應用已經延伸到物聯網、工業 4.0、供應鏈管理等多個領域，區塊鏈的應用開發以金融科技為代表逐漸擴大，在這些多重因素的催化下，區塊鏈已經開啟了一個快速發展的時期，但我們也必須以客觀的角度去看，區塊鏈是從比特幣所衍生發展出來的一項技術，是否真的成熟可用，是目前需要投入新的技術研發和更多的應用實作來證明它的價值，而區塊鏈曾發生過一系列安全的事件，透露出區塊鏈的技術仍有安全風險的挑戰需要被解決。

區塊鏈的發展，已從原本的區塊鏈 1.0 演進到現在的區塊鏈 2.0，2014 年前後，業界開始認識並注意到區塊鏈技術的重要價值，並將其用於電子貨幣之外的領域，而

這些應用稱為分散式應用(DAPP)。區塊鏈 1.0 比較著重在基礎層的發展，並且應用也多是金融與電子貨幣相關；而區塊鏈 2.0 的重心則是智能合約的發展，應用也開始多元化，身份的驗證、版權與著作權、學歷證明，甚至到醫療的數字病歷，都是區塊鏈的非金融面的應用，區塊鏈 2.0 試圖創建一個共用的技術平台，向開發者提供服務，提高了交易速度且大大的降低消耗的資源，也支持多數的共識機制運算，企圖讓分散式應用 (DAPP) 的開發變得更容易，如圖 5 所示。

圖 5　區塊鏈發展路徑
資料來源：本研究整理

在支付領域，金融單位特別是機構之間的對帳、清算、結算的成本較高，亦涉及了很多的手工流程，不僅是造成用戶端與金融單位後台業務所產生的支付業務費用高昂，也讓小額支付的業務難以發展。在資產管理的領域，股權、債權等各種資產是由不同的中介單位所託管，提高了此類的資產交易成本，同時亦有憑證遭偽造的隱憂。在證券領域，由於證券的交易生命週期流程耗時較長，增加了後台處理的成本。用戶

身分的驗證識別，不同的機構單位之間難以實現高效率的資訊交換，讓重複認證的成本高，除此之外也帶來用戶身分被中介機構洩漏的風險。

為了解決上述的問題，區塊鏈的技術具有不可篡改和可追朔的特性，可以用來建構監管部門所需要的功能，以利實施精準即時、與更多維度的監控。基於區塊鏈能實現點對點的資料價值轉移，透過資產數據化和重構金融單位的基礎設施，可達到大幅度提升金融資產交易清算、結算的流程效率，並降低交易成本，這可以很大程度地解決支付所面臨的問題。

以一些應用來當範例解說，支付領域：區塊鏈有助於降低金融機構之間的對帳成本及爭議解決，能夠顯著地提高支付業務的處理速度和效率，因為區塊鏈的這項技術，讓支付能夠打破原本因為成本因素考量的困境。

資產數位化：如股權、債券、票據等都可以被整合進區塊鏈中，成為鏈上的數位資產，並讓這些資產無須通過各種中介機構便能直接發起交易，資產的發行能夠靈活地採用保密或是公開的方式來進行。

智能債券：金融資產地交易是要由各方之間所協訂達成的合約，區塊鏈能透過智能合約的程式碼來達成這些業務上的邏輯，讓這些智能合約自動執行，並保證交易的雙方公開而對無關的第三者是保密的狀態，透過區塊鏈的技術讓智能債券能夠有相應的機制運行，亦能符合特定法律的規範和監管框架。

客戶識別：以金融機構為例，為顧客提供服務時必須履行客戶識別的責任，以傳統的方式去識別是非常耗時的流程，並缺少自動驗證消費者身分的技術，因此無法提高發展的效率，尤其在不同的機構之間更難實現訊息和交易紀錄的一致性，而區塊鏈的技術能夠實現數位化身分的資訊安全、可靠管理，保證客戶的隱私下亦能提升客戶識別的效率，以此來降低成本。

物流：透過數位簽章與公私鑰加密的機制，讓資訊和交易雙方的隱私得以安全，

例如在快遞交接時雙方以私鑰簽名，而簽收時只需要檢查交易即可，保證快遞員不會有偽造簽名的行為，透過區塊鏈的技術，智能合約能夠減化物流的程序並藉此大幅提升物流的速率。

音樂市場：傳統模式下，音樂製作人很難獲得合理的版稅，透過區塊鏈的技術，讓音樂的生產、傳播與收費都是透明化的資訊做呈現，能夠有效確保音樂製作人的銷售收益，也因為是通過區塊鏈的平台做發行，解決了侵權的問題，只須將心思花在管理自己的作品上。

區塊鏈以本質上來說，是利用分散式技術與共識機制的算法，而打造的一個信任機制，而儲存於鏈上的資訊則是高可靠且不易被篡改的，透明化的資訊幫助了如捐贈項目、募資明細、資金流向等，均可放於區塊鏈上，在滿足項目的參與者在隱私保護與法律的要求下，有條件地進行公開公示。

本研究之 1-4 論文重要性所提及，區塊鏈的重點不應只是放在金融相關的應用，由於市場上有太多的交易需求，這些需求也不全是與金融所相關，使得它更應該要往非金融的應用去發展，如圖 6 所示為區塊鏈應用的範例與領域。

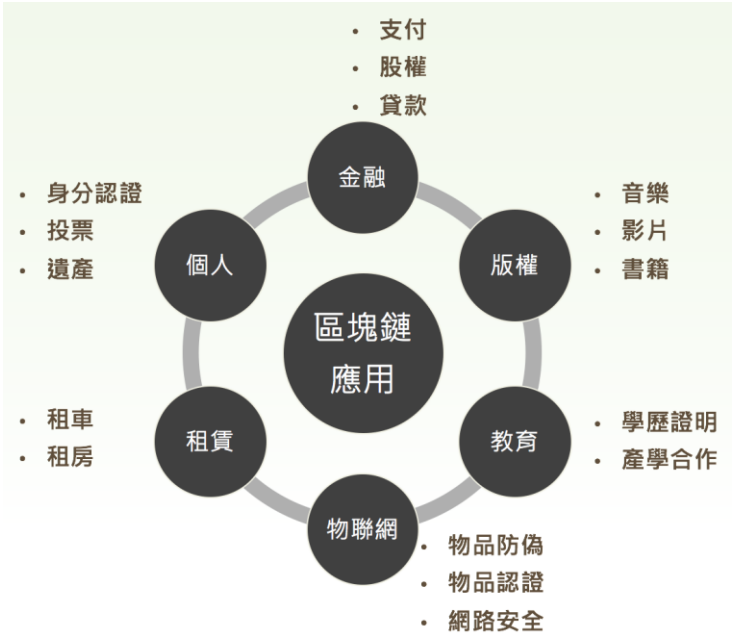

圖 6　區塊鏈相關應用

資料來源：本研究整理

2-1-3 區塊鏈技術之比較：比特幣、以太坊、Gcoin

國立台灣大學發展的 Gcoin 平台，以 Bitcoin 基礎來改良，有多中心階層式架構、多種資產功能、交易頻率改善、避免算力壟斷的特色，如表 2 所示。

表 2　Gcoin 特色一覽

項目	特色
多中心階層式架構	● 實現中央控管。 ● 多階層架構：聯盟成員、資產發行商...等。
多種資產功能	● 能夠乘載任意數量的資產，並擁有互相兌換的功能。
交易頻率改善	● 平均每 15 秒鐘生成一個區塊，大幅提升交易頻率。
避免算力壟斷	● 避免單一的節點掌握大部分的運算能力。 ● 動態工作量證明機制及調整驗證的困難度。 ● 降低電力的消耗。

資料來源：本研究整理

Gcoin 的階層式架構，主要分為聯盟成員、發行商、完整節點的角色，並且擁有不同的權限。

Gcoin 提出了將工作量證明調整，以動態難度的調整制度，此設計可避免單一成員擁有過多的說話權，進而影響網路之公正性，如下表 3 為 Gcoin 與 Bitcoin 的比較。

表 3　Gcoin 與 Bitcoin 的比較

	比特幣	Gcoin
共識算法	使用 SHA256 工作量證明	使用 SHA256 的 Consortium 工作量證明
平均區塊確認時間	10 分鐘	15 秒
區塊鏈方案	公有區塊鏈	聯盟區塊鏈
加密資產	單一資產	多種資產
鑄幣機制	礦工所得的獎勵	持有憑證的發行商發行

資料來源：本研究整理

參考 RSK 公司所整理有關以太坊和比特幣的差異 (https://docs.google.com/spreadsheets/d/1DQ77onGnHfJOoRSqTLmIkhuVK5CAbs-Fgqb6UoGMfVM/edit)，並提到實作區塊鏈的智能合約，環境建立與設定，在以太坊的環境下是容易實現的，如表 4 所示。

表 4　以太坊和比特幣的差異

	比特幣	以太坊
作者	Nakamoto S.	以太坊
區塊鏈狀態	公開	公開
腳本語言	比特幣腳本語言	Solidity
智能合約	有(但有限制)	有
圖靈完備性	無	有
平均交易時間	10 分鐘	15 秒

資料來源：本研究整理

學者亦提到建置環境的需求與步驟，透過 testrpc 在環境中建立虛擬的區塊鏈，提供測試之使用，Solidity 為開發智能合約之語言，類似 JavaScript 語法，完成合約的撰寫後，經由 Web3js 作佈署，讓使用者用其作互動(陳昶吾，2017)。

經過了評估以太坊與 Gcoin 的比較之後，由於 Gcoin 是以多中心化的角度去發展，並且沒有提供腳本語言讓開發者方便的撰寫智能合約，因此選擇了以太坊所提供去中心化平台，利用其所使用的 Solidity 來編寫智能合約，如圖 7 所示。

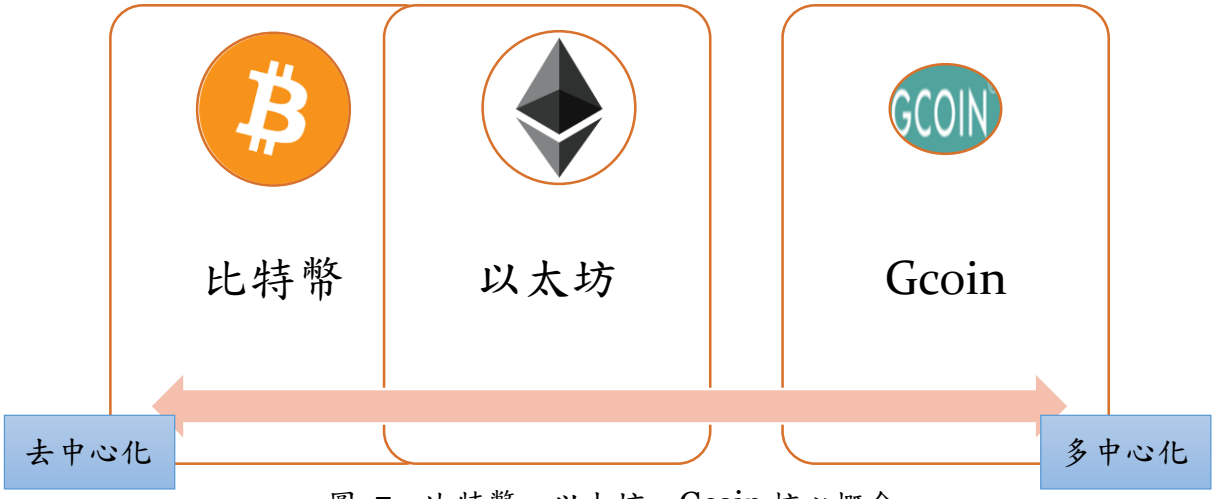

圖 7　比特幣、以太坊、Gcoin 核心概念

資料來源：本研究整理

2-1-4 區塊鏈技術之開源平台：以太坊

以太坊是一個開源且擁有智能合約功能的一個平台，並提供公共的區塊鏈讓發展的應用能夠透過其運作，使用專用的加密貨幣-以太幣(Ether)，利用去中心化的虛擬機器，來處理對等合約。也由於以太坊是一個分散式的平台來處理智能合約，應用程式完全地按照編譯的程式去執行，保證交易的可信任性、去中心化的運作。這些應用會在客製的區塊鏈上運行，而區塊鏈之間的價值允許互相轉移，這讓開發應用能夠自己

創建市場、儲存交易資訊，也提供開發中和未來可能開發的應用一個沒有中間單位的，且信任的交易市場。在以往的每個應用都必須架設自己的伺服器，自行運算伺服器上編成的程式碼，因此也讓應用程式之間的資訊傳遞與共享十分困難，如果單一的應用程式節點遭到入侵或是當機，則會造成該應用程式的用戶和該應用程式所提供的資訊受到影響；而在以太坊的區塊鏈上，任何人都可以設置應用程式的節點，以分散式的方式儲存，在符合使用協定下也能夠容易地取得其他節點上應用程式的必要資訊，這允許用戶的資訊是保有隱私。

如表 5 所示，為以太坊的技術架構，最下端網際網路與硬體端做配置，往上分為三個部分，SWARM 用以儲存資訊、WHISPER 用以資訊的傳遞與溝通、EVM 則提供了平台與共識機制的協定，第二層為分散式應用 (Decentralized Applications)，也是本研究著重的應用開發部分，而最上層則是 Mist Browser，提供一個瀏覽與使用分散式應用的一個瀏覽器工具。

表 5　以太坊技術架構

Mist Browser		
DAPP(分散式應用)		
SWARM(儲存)	WHISPER(溝通)	EVM(共識機制)
硬體端		
網際網路		

資料來源：本研究整理

2-2 智能合約

2-2-1 智能合約之起源與功能

智能合約最初是由 Nick Szabo 於 1994 所提出的理念，但因為缺少可執行的環境，智能合約並沒有實際的被應用於產業之中，隨著比特幣的概念發行，發現比特幣的底層技術區塊鏈可以為智能合約提供一個可被信任的執行環境，而以太坊首先看到了兩者可契合的可能性，並致力發展可以讓智能合約執行的平台。

智能合約是一個可以被自動執行的程式，它可以儲存和發送資訊，並且可在收到資訊後執行運算，而智能合約必須是經過預先編寫和部署，來處理接受的資訊，以完成交易。

2-2-2 以太坊實作智能合約：Solidity 語言

當比特幣的概念被發行，實驗不斷的在開始時，主要是兩大項目，一為分散式的資料庫，並且擁有嚴謹密碼學的支持，二為強大的交易系統，提供交易能夠在全世界在沒有中介單位的情況下被執行。而在其後的發展中，終於提出了第三個特徵，也就是一個圖靈完備性的腳本語言。到目前爲止，身份註冊、發行電子貨幣、智能財產及合約，實現這些應用都需要創建一個完整的協定，因此需要一個強大的基礎層與腳本語言。以太坊是一個已經模組化、圖靈完備性的腳本語言系統，並且與區塊鏈結合，以簡單、普遍性的理念來做開發，其目標是希望能夠為這些分散式的應用提供一個平台。

上述所提到的腳本語言即為 Solidity，它是一個類似於 JavaScript 的語言，並且提供 ABI (Application Binary Interface) 來讓網頁端能夠呼叫智能合約上的功能函式。除此之外，Solidity 提供線上即時編譯的 IDE 讓開發者能夠快速的撰寫智能合約內的功能，如圖 8。

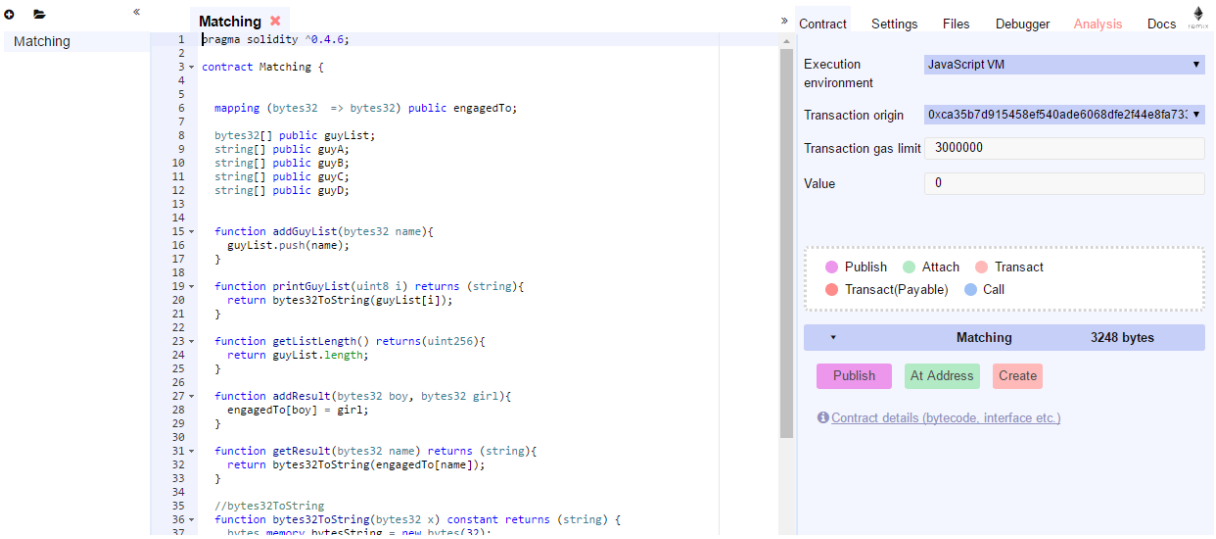

圖 8　線上編譯 Solidity 語言

資料來源：(https://ethereum.github.io/browser-solidity/#version=soljson-v0.4.7+commit.822622cf.js)

2-3 多對多配對：延遲接受演算法

2-3-1 穩定婚姻問題

穩定婚姻問題是一個配對的難題，有數個男生與數個女生，而每個人均有自己心目中的排名，該如何做出最佳的配對？最好的配對方案當然是，每個人的另一半正好都是自己的第一選擇。這雖然很完美，但絕大多數情況下都不可能實現。比方說，男

1 號最喜歡的是女 1 號，而女 1 號的最愛不是男 1 號，這兩個人的最佳選擇就不可能被同時滿足。如果好幾個男孩兒最喜歡的都是同一個女孩兒，這幾個男孩兒的首選也不會同時得到滿足。其實，找的對象太完美不見得是好事兒，和諧才是婚姻的關鍵。如果男 1 號和女 1 號各有各的對象，但男 1 號覺得，比起自己現在的，女 1 號更好一些；女 1 號也發現，在自己心目中，男 1 號的排名比現男友更靠前。這樣一來，這兩人就可能拋棄各自現在的對象，如果出現了這種情況，我們就說婚姻搭配是不穩定的。為男女配對時，雖然不能讓每個人都得到最滿意的，但搭配必須得穩定。換句話說，對于每一個人，在他心目中比他當前伴侶更好的異性，都不會認為他也是一個更好的選擇。因此必須面對的問題是：穩定的婚姻搭配總是存在嗎？應該怎樣尋找？

為了方便分析，使用字母 A、B、C 對男性進行編號，用數字 1、2、3 對女性進行編號。我們把所有男性從上到下列在左側，括號里的數字表示每個人心目中對所有女性的排名；再把所有女性列在右側，用括號里的字母表示她們對男性的偏好。圖 9 所示的就是 2 男 2 女的一種情形，每個男的都更喜歡女 1 號，但女 1 號更喜歡男 B，女 2 號更喜歡男 A。若按 A-1、B-2 進行搭配，則男 B 和女 1 都更喜歡對方一些，這樣的婚姻搭配就是不穩定的。但若換一種搭配方案如圖 10，這樣的搭配就是穩定的了。

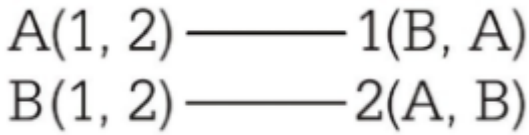

圖 9　一個不穩定的婚姻搭配圖

資料來源：(http://www.rocidea.com/roc-6058.aspx)

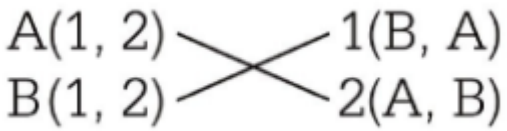

圖 10　一個穩定的婚姻搭配圖

資料來源：(http://www.rocidea.com/roc-6058.aspx)

如果還有想要私奔的男女對，就繼續按照他們的願望交換情侶，直到最終消除所有的不穩定組合。容易看出，透過這種修補策略所得到的最終結果一定滿足穩定性，但這種策略的問題在于，它不一定存在最終結果。事實上，按照上述方法反覆調整搭配方案，最終可能會陷入一個死循環，因此該策略甚至不能保證得出一個確定的方案來，如圖 11 所示。

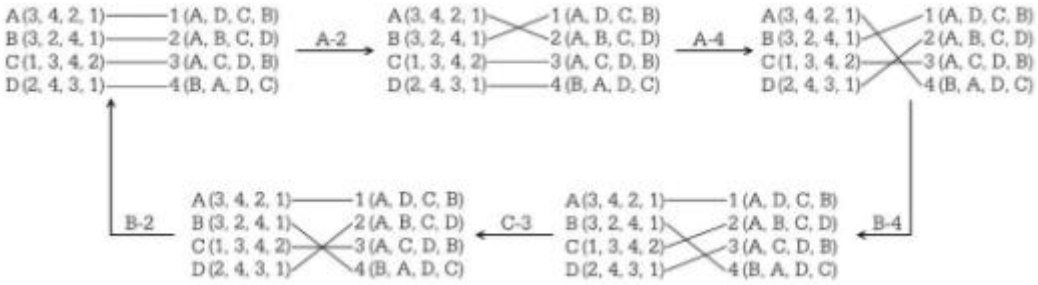

圖 11　修補策略可能導致死循環

資料來源：(http://www.rocidea.com/roc-6058.aspx)

1962 年，美國數學家 David Gale 和 Lloyd Shapley 發明了一種尋找穩定婚姻的策略。不管男女各有多少人，也不管他們的偏好如何，應用這種策略後總能得到一個穩定的搭配。換句話說，他們證明了穩定的婚姻搭配總是存在的，而下一節會說明延遲接受演算法細節。

2-3-1 延遲接受演算法

參考 Alvin E. Roth 的著作-Who Gets What - and Why 創造金錢買不到的機會，指出許多的市場配對需要有良好的機制去處理，如：群眾募資、考試分發、實習生應徵、購物，並且提出延遲接受演算法的機制，認為配對要發揮最大的的功效，必須盡可能符合每個使用者的喜好順序。

為了描述這個演算法，將會以應徵者與雇主在採取行動，下面說明如何操作：

步驟 0：應徵者與雇主向交易所提供志願表，並按喜好順序作排序。

步驟 1：每個雇主向他的首選候選人提出一輪工作合約，合約數量最多是工作職缺數。每名應徵者檢視收到的合約，並暫時接受一個最好的合約，也就是喜好度中順位最高者，同時拒絕其他合約。

步驟 N：每個雇主將前幾個步驟被拒絕的工作合約轉而向下一輪他所希望錄取的應徵者提出(視雇主的喜好度而定)，而當應徵者收到合約或是收到新合約時，會暫時選擇其最喜歡(喜好度順位最高)的那一個，無論之前是否已經暫時接受過其他合約，只考慮喜好度的比較。

當沒有合約被拒絕的時候，演算法就結束，因為沒有公司會想提出更多的合約，此時應徵者和雇主皆已完成配對。

2-3-2 延遲接受演算法之解決穩定婚姻問題

假設有 N 男 N 女，每個人對異性對象提出喜好程度 1~N 的排序，並使得不出現以下

情況：在 N 男 N 女中任意兩對夫婦 (M,W)、 (m,w)，若有 M 男對 w 女喜好度大於現任妻子 W 女，並且 w 女對 M 男喜好度也大於現任丈夫 m 男的情形發生 (代表不穩定)。

```
function stableMatching {
    Initialize all m ∈ M and w ∈ W to free
    while ∃ free man m who still has a woman w to propose to {
        w = first woman on m's list to whom m has not yet proposed
        if w is free
            (m, w) become engaged
        else some pair (m', w) already exists
            if w prefers m to m'
                m' becomes free
                (m, w) become engaged
            else
                (m', w) remain engaged
    }
}
```

圖 12　延遲接受演算法範例：穩定婚姻問題

資料來源：https://en.wikipedia.org/wiki/Stable_marriage_problem

每個人都有選擇自己喜歡對象的權利。假設在一個 n 男 n 女的聯誼會中，如果每個人都對異性排出一個喜好程度表(我們假設每個人只喜歡異性)，以表 6 與表 7 為例，其中 A,B,C,D 代表男生，w,x,y,z 代表女生，就 A 而言，他喜歡對方的程度是 w>x>y>z，就 w 而言，他喜歡對方的程度是 D>C>A>B，每個人都表明自己的志願之後，我們是否可以為他們找到一組穩定的配對方式呢？也就是說，如果 A,w 被配在一對，C,y 被配在一對，就不可以發生 A 喜歡 y 勝過 w，而且 y 喜歡 A 勝過 C 的情形。這個問題，稱作穩定配對 (stable matching) 問題。

表 6　男性喜好程度表

A	w	x	y	z
B	x	z	w	y
C	x	w	y	z
D	y	x	z	w

資料來源：(劉俊宏, 穩定配對問題)

表 7　女性喜好程度表

	w	x	y	z
	D	B	D	C
	C	A	C	B
	A	D	B	A
	B	C	A	D

資料來源：(劉俊宏, 穩定配對問題)

接著用數學的方式來描述**定義**：

(1) 一個配對(matching) M 是一個集合，其中的元素都是這種形式的數對(order pair) (A,a)其中 A 為男，a 為女。 i.e. $M = \{(m,w) | m: male, w: female\}$，且每一男至多只配一女，每一女至多只配一男。

(2) 一個完美配對是一個配對，而且每一男恰好配一女，每一女也恰好配一男。

(3) 若一個配對中有一男 m 和一女 w，使得 m 喜歡 w 勝過他喜歡現在的對象，而且 w 也喜歡 m 勝過他喜歡現在的對象，則稱這是一個不穩定的配對。

(4) 若一個完美配對不是不穩定的配對，則稱這是一個穩定配對。

以之前的例子來看，{(A,z), (B,x), (C,w), (D,y)}就是一穩定配對，但是{(A,z), (B,x), (C,w), (D,w)}不是一個配對，當然就不是穩定配對；而{(A,z), (B,x), (C,w)}雖然是配對，但卻不是完美配對；至於{(A,x), (B,z), (C,w), (D,y)}雖然是個完美配對，但卻

不是個穩定配對，因為 B 喜歡 x 勝過他現在的對象 z，而且 x 喜歡 B 勝過他現在的對象 A。而給定任何一個志願表，都會存在穩定配對，而延遲接受演算法保證可以得到一個穩定配對。首先，每個男人都去向其最喜歡的人求婚，如果說每一個女生都恰好被一個人生求婚，那就停止，如果某一個女生被兩個以上的男生求婚，則取該女生比較喜歡的人。接下來，被拒絕的男生們去向其第二喜歡的女生求婚，若女方在此可以選擇這次比較好的對象而拒絕前一次的對象，以此類推。

以上面的例子為例：

{(A,w), (B,x), (C,x), (D,y)}

=>{(A,w), (B,x), (C,w), (D,y)}

=>{(A,x), (B,x), (C,w), (D,y)}

=>{(A,y), (B,x), (C,w), (D,y)}

=>{(A,z), (B,x), (C,w), (D,y)}

值得注意的是，隨著一次次求婚，男方的對象他會越來越不喜歡，女方的對象會越來越喜歡。但儘管如此，這還是會得到一個對男方最有利的穩定配對 M，也就是不會有另一個穩定配對 M'，使得 M'中每個男人的對象都比 M 中的好或相同。

現在我們來證明這個演算法必然會得到一組穩定配對：我們需要證明 2 件事，一是證明這演算法會結束，二是證明此演算法會得到穩定配對。首先，先證明這個演算法一定會結束。因為經過一次又一次的求婚，沒有老婆的男人會越來越少，而且每個男人最多求婚 n 次。如果有一男被所有的女人拒絕，表示拒絕他的人都已經有對象了，也就是說，已經有 n 個男人已被配對，但是因為男女人數一樣多，矛盾。所以當演算法結束時，必然會得到一組配對。接下來證明得到的這組配對必然是穩定配對。假設配對中的(m,w)與(M,W)造成不穩定，則表示 m 喜歡 W 勝過 w，而且 W 喜歡 m 勝過 M。但是，既然演算法得到這樣的結果，表示 m 曾向 W 求婚，但被 W 拒絕了，

所以 m 才會去向其他比較不喜歡的人求婚，所以說，W 應該要喜歡她的對象勝過 m，但這得到了矛盾。所以，此演算法必然生成一個穩定配對。

上述執行演算法的時候，是讓男方主動出擊去向女方求婚，如果改成讓女方主動，而得到下裂的結果，和先前結果的一致。

{(D,w), (B,x), (D,y), (C,z)}

=>{(C,w), (B,x), (D,y), (C,z)}

=>{(C,w), (B,x), (D,y), (B,z)}

=>{(C,w), (B,x), (D,y), (A,z)}

以另一例子配對，如表 8 所示，由男方主動：

{(A,w), (B,x), (C,w), (D,y)}

=>{(A,w), (B,x), (C,y), (D,y)}

=>{(A,w), (B,x), (C,z), (D,y)}

由女方主動：

{(A,w), (A,x), (B,y), (D,z)}

=>{(A,w), (C,x), (B,y), (D,z)}

得到了兩個不同的穩定配對，可見得穩定配對不一定會是唯一的。

表 8　配對案例

	w	x	y	Z
A	w,A	x,A	y,B	z,D
B	x,B	y,C	z,A	w,C
C	w,C	y,B	z,D	x,B
D	y,D	x,D	w,C	z,A

資料來源：(劉俊宏, 穩定配對問題)

另一個例子，現在有 2n 男 2n 女，每列的數字代表男，每行的數字代表女，如表 9，那麼，$\{(i,i)|i=1,...2n\}$是一組穩定配對，$\{(1,2),(2,1)\} \cup \{(i,i)|i=3,...2n\}$亦是一組穩定配

對，$\{(1,2),(2,1),(3,4),(4,3)\} \cup \{(i,i)|i=5,...2n\}$還是一組穩定配對，因此可以得到$2^n$種穩定配對。如果女方的每個人的喜好都一樣，則只會有一組穩定配對。得知穩定配對有可能是唯一的。

表 9　2n 男 2n 女配對範例

	1	2	3	4	…	2*n-1	2*n
1	1,2	2,1	…	…	…	…	…
2	2,1	1,2	…	…	…	…	…
3	3,4	4,3	…	…	…	…	…
4	4,3	3,4	…	…	…	…	…
…	…	…	…	…	…	…	…
2*n-1	2n-1,2n	2n,2n-1	…	…	…	…	…
2*n	2n,2n-1	2n-1,2n	…	…	…	…	…

資料來源：(劉俊宏, 穩定配對問題)

2-3-3 延遲接受演算法之意義與應用

每個算法都有它的實際意義，能給我們帶來很多啟發。Gale-Shapley 所提出的延遲接受演算法最大的意義就在於，如果作為一個為這些男女牽線的媒人，你不需要親自計算穩定婚姻匹配，甚至根本不需要了解每個人的偏好，只需要按照這個算法組織一個男女配對活動就可以了。而需要做的只是把算法流程當作遊戲規則告訴全部人，游戲結束後會自動得到一個大家都滿意的婚姻匹配。整個算法可以簡單地描述為：每個人都去做自己想做的事情。對於男性來說，從最喜歡的女孩兒開始追起是順理成章的事；對於女性來說，不斷選擇最好的男孩兒也正好符合她的利益。因此，所有人會自動遵守游戲規則，不用擔心有人虛報自己的偏好。

歷史上，這樣的配對有過實際應用，而更有意思的是，這個算法的應用居然比算

法本身的提出還早 10 年。早在 1952 年，美國就開始用這種辦法給醫學院的學生安排工作，這被稱之為全國住院醫師配對項目。配對的基本流程就是，各醫院從尚未拒絕這一職位的醫學院學生中選出最佳人選並發送聘用通知，當學生收到來自各醫院的聘用通知後，系統會根據他所填寫的意願表自動將其分配到意願最高的職位，並拒絕掉其他的職位。如此反覆執行，直到每個學生都分配到了工作。那時人們並不知道這樣的流程可以保證工作分配的穩定性，只是憑直覺認為這是很合理的。直到 10 年之后，Gale 和 Shapley 才系統地研究了這個流程，提出了穩定婚姻問題，並證明了這個算法的正確性。

這個算法還有很多有趣的性質。比如這種男追女女拒男的方案對男性更有利還是對女性更有利呢？答案是，這種方案對男性更有利。事實上，穩定婚姻搭配往往不止一種，然而上述算法的結果可以保證，每一位男性得到的伴侶都是所有可能的穩定婚姻搭配方案中最理想的，同時每一位女性得到的伴侶都是所有可能的穩定婚姻搭配方案中最差的。

這個算法會有一些潛在的問題，對於每位女性來說，得到的結果都是所有可能的穩定搭配中最差的一種。此時，倘若有某位女性知道所有其他人的偏好列表，經過精心計算，她有可能發現，故意拒絕掉本不該拒絕的人（暫時保留一個較差的人在身邊），或許有機會等來更好的結果。因而，在實際生活中應用這種算法，不得不考慮一些可能的欺詐與博弈。

這個演算法還有一些局限。例如，它無法處理 2n 個人不分男女的穩定搭配問題。一個簡單的應用場景便是宿舍分配問題：假設每個宿舍住兩個人，已知 2n 個學生中每一個學生對其餘 2n-1 個學生的偏好評價，如何尋找一個穩定的宿舍分配？此時，Gale-Shapley 算法就無法發揮。而事實上，宿舍分配問題中很可能根本就不存在穩定的搭配。例如，有 A、B、C、D 四個人，其中 A 把 B 排在第一，B 把 C 排在第

一，C 把 A 排在第一，而且他們三人都把 D 排在最後。容易看出，此時一定不存在穩定的宿舍分配方案。倘若 A、D 同宿舍，B、C 同宿舍，那 C 會認為 A 是更好的室友（因為 C 把 A 排在了第一），同時 A 會認為 C 是更好的室友（因為他把 D 排在了最后）。同理，B、D 同宿舍或者 C、D 同宿舍也都是不行的，因而穩定的宿舍分配是不存在的。此時，重新定義宿舍分配的優劣性便是一個更為基本的問題。穩定婚姻問題還有很多其他的變種，有些問題甚至是 NP 完全問題，至今仍然沒有一種有效的算法。

2-3-4 延遲接受演算法之實作

在大多數的程式語言中皆有此演算法被實作的範例，本研究參考 https://rosettacode.org/wiki/Stable_marriage_problem，選擇以 JavaScript 語言為範例，來進行編寫智能合約的多對多實作雛型系統。

3. 以區塊鏈技術為基礎之多對多配對雛形系統架構與開發

3-1 多對多配對雛形系統架構

本研究之系統可由網頁端介面操作，也可以使用終端機來運行，當作後端控制，網頁透過 HTML 和 JavaScript 來運行，而終端機則是以 Node.js 的指令介面。以 Solidity 語言撰寫智能合約並透過 Web3.js 來部署智能合約與其互動，本研究使用 testrpc，在環境中建立虛擬的區塊鏈中執行並測試，使用者透過網頁瀏覽器輸入資料，經由 JavaScript 的執行，呼叫智能合約的功能函式，包括了儲存輸入值和配對結果；而 Node.js 的指令介面同樣也可以呼叫智能合約中的功能函式，觀察儲存的情況和數值的變化，如圖 13 所示。

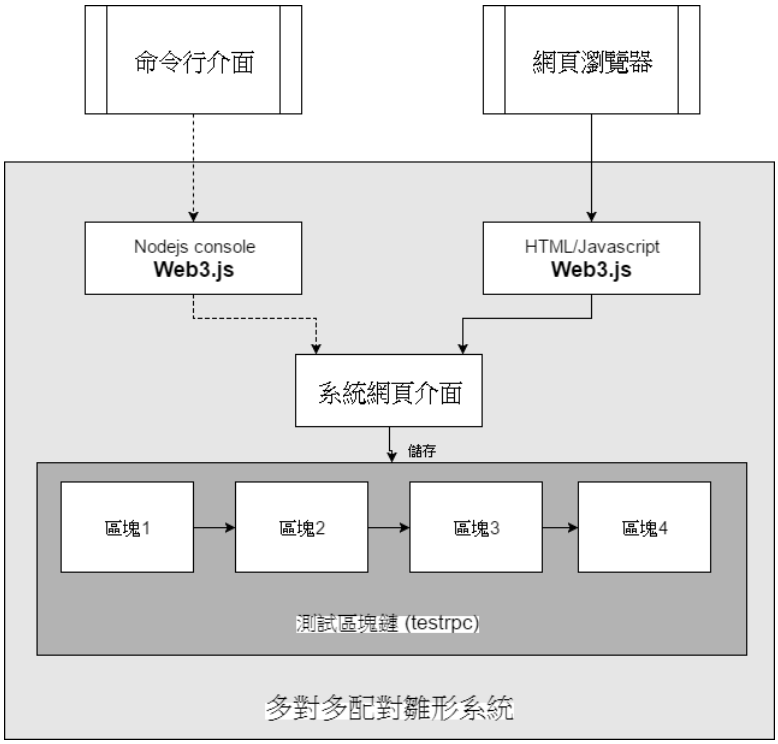

圖 13　系統架構圖

資料來源：本研究整理

使用案例圖，如圖 8 所示，使用者可以做下列四個動作，第一項，於系統的介面上觀看候選名單及之前配對的結果，提供使用者做排序選擇的參考。第二項，使用者編排自己的喜好順序，送出後系統會於介面上呈現，讓使用者確認是否無誤。第三項，當系統依照延遲接受演算法完成多對多的排序之後，使用者可從系統介面得知自己配對的結果。第四項，使用者對於配對的結果是否滿意，做出確定或是取消的決定，若想取消，可以透過第二項功能重新決定並輸入自己的喜好順序，等待系統再一次的配對結果。

3-2 多對多配對雛形系統分析

3-2-1 使用案例圖

如圖 14 所示，為本研究系統之使用案例圖，使用者對於系統的操作可以分為主要四個動作行為，觀看候選結果名單：使用者可以透過系統介面得知可選的配對對象，提供其選擇的考量。編排喜好順序：使用者可以透過系統介面決定配對的喜好順序，並送出最後決定的偏好順序。觀看配對結果：當系統完成多對多配對後，使用者可以查看自己配對的結果。同意配對結果：使用者對於配對的結果可以選擇決定或是放棄，若決定則系統會儲存最後配對結果，反之放棄則使用者可以重新前三項的行為，觀看名單、重新編排順序並等待新的配對結果。

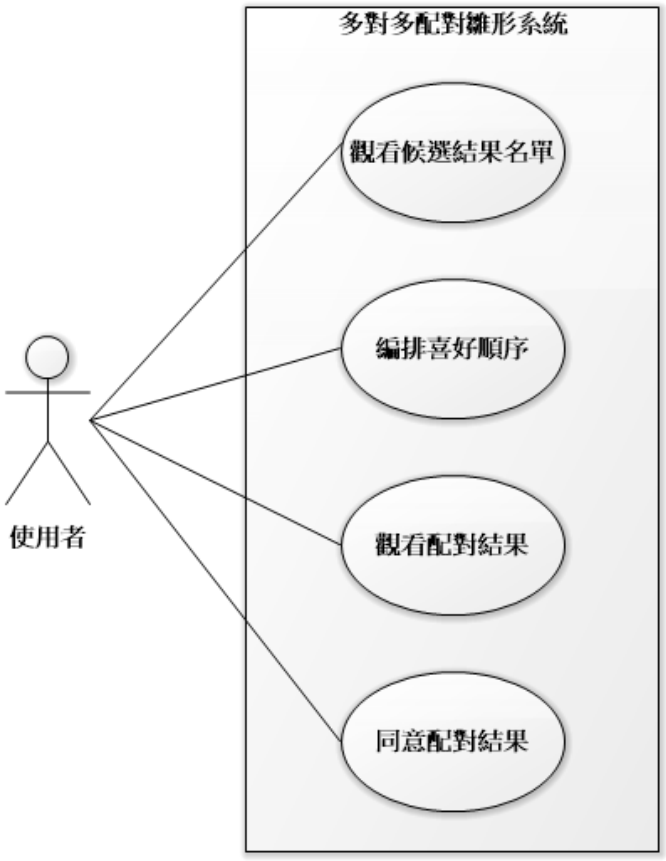

圖 14　使用案例圖

資料來源：本研究整理

3-2-2 活動圖

如圖 15 所示為系統之活動圖，分為使用者端與多對多配對系統端，首先使用者決定並輸入欲配對的喜好順序，送出後使用者端等待配對的結果，而系統端則會依照延遲接受演算法做多對多的配對，完成配對後會回傳顯示結果於系統中，接著使用者可以做確認配對結果的動作，若滿意則流程結束；反之不滿意則回到輸入或調整喜好的順序，重新等待系統再一次的配對結果。

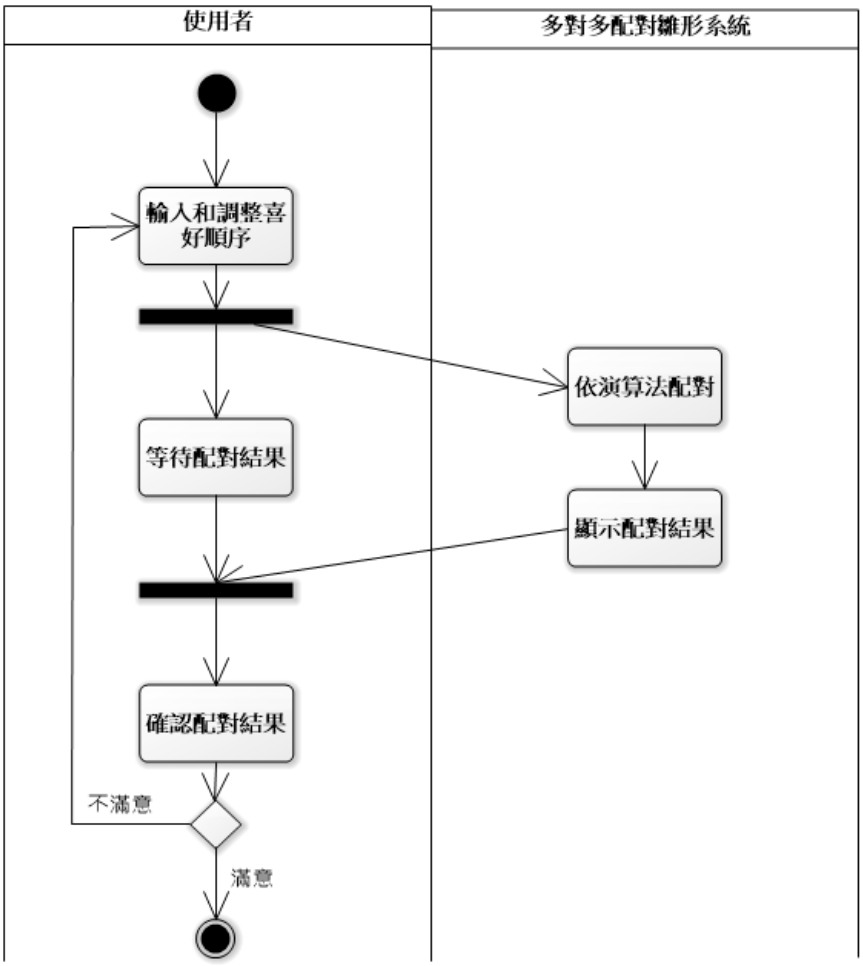

圖 15　活動圖

資料來源：本研究整理

3-2-3 循序圖

如圖 16 為輸入喜好順序之循序圖，分為使用者、系統、測試區塊鏈，共三個區段，使用者呼叫函式向系統送出喜好順序的輸入，接著系統將該使用者的喜好順序透過智能合約的函式儲存於測試的區塊鏈上。

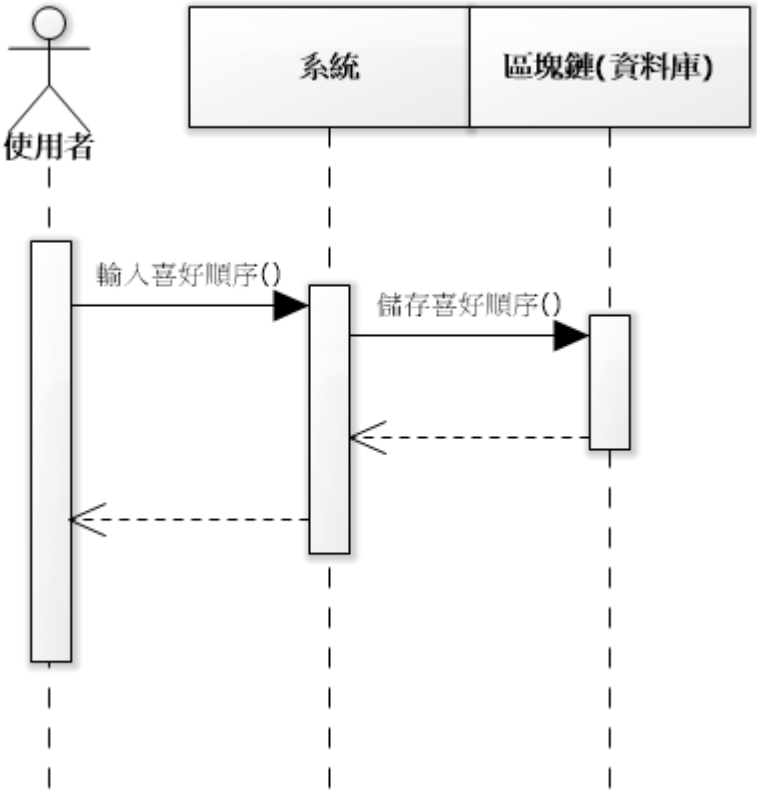

圖 16　循序圖(輸入喜好順序)

資料來源：本研究整理

如圖 17 所示為多對多配對的循序圖，系統對於已存在的使用者及其喜好的順序，執行多對多的配對 (延遲接受演算法)，使用者可透過系統查詢配對之結果，當該使用者接受配對的結果，則透過函式呼叫系統，系統便會將該使用者的配對結果，透過另一個函式將資訊儲存於測試的區塊鏈上。

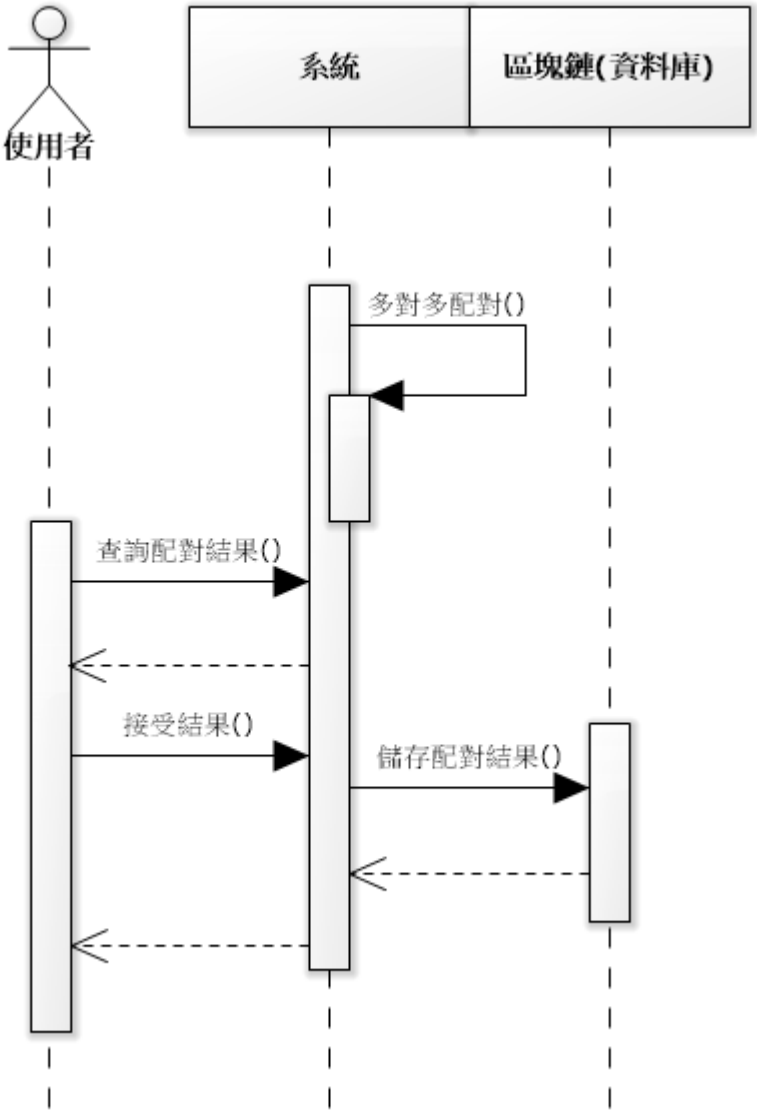

圖 17　循序圖(多對多配對)

資料來源：本研究整理

3-2 環境建立

在作業系統的選擇上，由於安裝測試區塊鏈在 Windows 系統並不健全，亦常發生不可預期的錯誤，因此建議環境應選擇 Linux 的系統，而本研究所使用的作業系統為 CentOS7-1611，安裝過程參考 https://d3vv6lp55qjaqc.cloudfront.net/items/1j3I3T3G1f2C3P2w1D43/ETH_dev.html，以下為本研究整理之建立區塊鏈環境的步驟。

3-2-1 基本環境套件部署

安裝基本套件

```
$ sudo yum install -y curl git-all irb python-setuptools ruby tmux
```

上述安裝 git-all.noarch、ruby.x86_64、ruby-irb.noarch、tmux.x86_64，並安裝 56 個相關套件，如圖 18、圖 19 所示。

```
Dependency Installed:
  apr.x86_64 0:1.4.8-3.el7                 apr-util.x86_64 0:1.5.2-6.el7              cvs.x86_64 0:1.11.23-35.el7
  emacs-git.noarch 0:1.8.3.1-6.el7_2.1     emacs-nox.x86_64 1:24.3-19.el7_3           git.x86_64 0:1.8.3.1-6.el7_2.1
  git-gui.noarch 0:1.8.3.1-6.el7_2.1       git-p4.noarch 0:1.8.3.1-6.el7_2.1          git-svn.x86_64 0:1.8.3.1-6.el7_2.1
  libyaml.x86_64 0:0.1.4-11.el7_0          perl-Authen-SASL.noarch 0:2.15-10.el7      perl-Compress-Raw-Bzip2.x86_64 0:2.061-3.el7
  perl-DBI.x86_64 0:1.627-4.el7            perl-Data-Dumper.x86_64 0:2.145-3.el7      perl-Digest.noarch 0:1.17-245.el7
  perl-Digest-SHA.x86_64 1:5.85-3.el7      perl-Error.noarch 1:0.17020-2.el7          perl-GSSAPI.x86_64 0:0.28-9.el7
  perl-IO-Compress.noarch 0:2.061-2.el7    perl-IO-Socket-IP.noarch 0:0.21-4.el7      perl-IO-Socket-SSL.noarch 0:1.94-5.el7
  perl-Net-SMTP-SSL.noarch 0:1.01-13.el7   perl-Net-SSLeay.x86_64 0:1.55-4.el7        perl-PlRPC.noarch 0:0.2020-14.el7
  ruby-libs.x86_64 0:2.0.0.648-29.el7      rubygem-bigdecimal.x86_64 0:1.2.0-29.el7   rubygem-io-console.x86_64 0:0.4.2-29.el7
  rubygem-rdoc.noarch 0:4.0.0-29.el7       rubygems.noarch 0:2.0.14.1-29.el7          subversion.x86_64 0:1.7.14-10.el7
  tcl.x86_64 1:8.5.13-8.el7                tk.x86_64 1:8.5.13-6.el7
```

圖 18　安裝基本套件

資料來源：本研究整理

```
cvsps.x86_64 0:2.2-0.14.b1.el7                emacs-common.x86_64 1:24.3-19.el7_3
git-cvs.noarch 0:1.8.3.1-6.el7_2.1            git-email.noarch 0:1.8.3.1-6.el7_2.1
gitk.noarch 0:1.8.3.1-6.el7_2.1               liblockfile.x86_64 0:1.08-17.el7
perl-Compress-Raw-Zlib.x86_64 1:2.061-4.el7   perl-DBD-SQLite.x86_64 0:1.39-3.el7
perl-Digest-HMAC.noarch 0:1.03-5.el7          perl-Digest-MD5.x86_64 0:2.52-3.el7
perl-Git.noarch 0:1.8.3.1-6.el7_2.1           perl-Git-SVN.noarch 0:1.8.3.1-6.el7_2.1
perl-Net-Daemon.noarch 0:0.48-5.el7           perl-Net-LibIDN.x86_64 0:0.12-15.el7
perl-TermReadKey.x86_64 0:2.30-20.el7         perl-YAML.noarch 0:0.84-5.el7
rubygem-json.x86_64 0:1.7.7-29.el7            rubygem-psych.x86_64 0:2.0.0-29.el7
subversion-libs.x86_64 0:1.7.14-10.el7        subversion-perl.x86_64 0:1.7.14-10.el7
```

圖 19　安裝基本套件(續)

資料來源：本研究整理

安裝 zsh，指令如下，如圖 20 所示為安裝 zsh 成功時的畫面，且安裝完成後終端機介面改為 zsh(z shell)的畫面如圖 21 所示。

```
$ sudo yum install -y zsh
```

```
$ sh -c "$(curl -fsSL https://raw.githubusercontent.com/robbyrussell/oh-my-zsh/master/tools/install.sh)"
```

```
[dan@localhost ~]$ sudo yum install -y zsh
Loaded plugins: fastestmirror, langpacks
Loading mirror speeds from cached hostfile
 * base: ftp.yzu.edu.tw
 * extras: ftp.yzu.edu.tw
 * updates: ftp.yzu.edu.tw
Resolving Dependencies
--> Running transaction check
---> Package zsh.x86_64 0:5.0.2-25.el7_3.1 will be installed
--> Finished Dependency Resolution

Dependencies Resolved

================================================================================
 Package        Arch          Version                    Repository      Size
================================================================================
Installing:
 zsh            x86_64        5.0.2-25.el7_3.1           updates        2.4 M

Transaction Summary
================================================================================
Install  1 Package

Total download size: 2.4 M
Installed size: 5.6 M
Downloading packages:
zsh-5.0.2-25.el7_3.1.x86_64.rpm                            | 2.4 MB  00:00:00
Running transaction check
Running transaction test
Transaction test succeeded
Running transaction
  Installing : zsh-5.0.2-25.el7_3.1.x86_64                                 1/1
  Verifying  : zsh-5.0.2-25.el7_3.1.x86_64                                 1/1

Installed:
  zsh.x86_64 0:5.0.2-25.el7_3.1

Complete!
```

圖 20　安裝 zsh

資料來源：本研究整理

```
Cloning Oh My Zsh...
Cloning into '/home/dan/.oh-my-zsh'...
remote: Counting objects: 831, done.
remote: Compressing objects: 100% (700/700), done.
remote: Total 831 (delta 14), reused 775 (delta 10), pack-reused 0
Receiving objects: 100% (831/831), 567.68 KiB | 348.00 KiB/s, done.
Resolving deltas: 100% (14/14), done.
Looking for an existing zsh config...
Using the Oh My Zsh template file and adding it to ~/.zshrc
Time to change your default shell to zsh!
變更 dan 的 shell。
密碼：
shell 已變更。
         __                                     __
  ____  / /_     ____ ___  __  __   ____  _____/ /_
 / __ \/ __ \   / __ `__ \/ / / /  /_  / / ___/ __ \
/ /_/ / / / /  / / / / / / /_/ /    / /_(__  ) / / /
\____/_/ /_/  /_/ /_/ /_/\__, /    /___/____/_/ /_/
                        /____/                       ....is now installed!


Please look over the ~/.zshrc file to select plugins, themes, and options.

p.s. Follow us at https://twitter.com/ohmyzsh.

p.p.s. Get stickers and t-shirts at http://shop.planetargon.com.

➜  ~
```

圖 21　安裝 zsh 後終端機改變

資料來源：本研究整理

安裝 Linuxbrew，指令如下，安裝成功畫面如圖 22 所示。

```
$ ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Linuxbrew/install/master/install)"
```

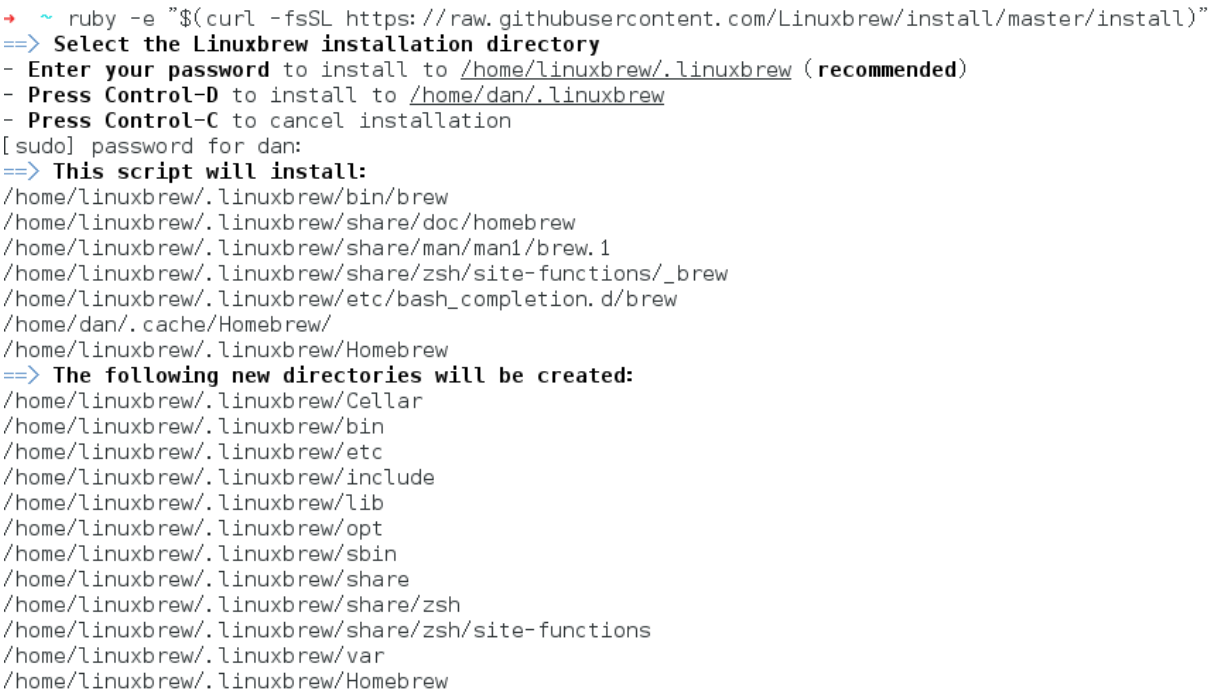

圖 22　安裝 Linuxbrew

資料來源：本研究整理

安裝完 Linuxbrew 後須添加路徑，如下指令，完成畫面如圖 23 所示。

```
$ PATH="$HOME/.linuxbrew/bin:$PATH"
export PATH="/home/user/.linuxbrew/bin:$PATH"
export PATH="/home/user/.linuxbrew/sbin:$PATH"
export MANPATH="/home/user/.linuxbrew/share/man:$MANPATH"
export INFOPATH="/home/user/.linuxbrew/share/info:$INFOPATH"
```

```
➜  ~ PATH="/home/linuxbrew/.linuxbrew/bin:$PATH"
➜  ~ brew
Example usage:
  brew search [TEXT|/REGEX/]
  brew (info|home|options) [FORMULA...]
  brew install FORMULA...
  brew update
  brew upgrade [FORMULA...]
  brew uninstall FORMULA...
  brew list [FORMULA...]

Troubleshooting:
  brew config
  brew doctor
  brew install -vd FORMULA

Developers:
  brew create [URL [--no-fetch]]
  brew edit [FORMULA...]
  http://docs.brew.sh/Formula-Cookbook.html

Further help:
  man brew
  brew help [COMMAND]
  brew home
```

圖 23　添加 Linuxbrew 路徑

資料來源：本研究整理

安裝開發環境套件，如下指令，完成畫面如圖 24~27 所示。

```
$ sudo yum groupinstall -y 'Development Tools'
```

```
Installed:
  autoconf.noarch 0:2.69-11.el7     automake.noarch 0:1.13.4-3.el7  bison.x86_64 0:2.7-4.el7
  diffstat.x86_64 0:1.57-4.el7      doxygen.x86_64 1:1.8.5-3.el7    flex.x86_64 0:2.5.37-3.el7
  intltool.noarch 0:0.50.2-6.el7    libtool.x86_64 0:2.4.2-22.el7_3 patch.x86_64 0:2.7.1-8.el7
  rpm-build.x86_64 0:4.11.3-21.el7 rpm-sign.x86_64 0:4.11.3-21.el7 swig.x86_64 0:2.0.10-5.el7
```

圖 24　安裝開發套件

資料來源：本研究整理

```
byacc.x86_64 0:1.9.20130304-3.el7 cscope.x86_64 0:15.8-9.el7         ctags.x86_64 0:5.8-13.el7
gcc-c++.x86_64 0:4.8.5-11.el7     gcc-gfortran.x86_64 0:4.8.5-11.el7 indent.x86_64 0:2.2.11-13.el7
patchutils.x86_64 0:0.3.3-4.el7   rcs.x86_64 0:5.9.0-5.el7           redhat-rpm-config.noarch 0:9.1.0-72.el7.centos
systemtap.x86_64 0:3.0-7.el7
```

圖 25 安裝開發套件(續)

資料來源：本研究整理

```
Dependency Installed:
  dwz.x86_64 0:0.11-3.el7                    gettext-common-devel.noarch 0:0.18.2.1-4.el7
  libquadmath-devel.x86_64 0:4.8.5-11.el7    libstdc++-devel.x86_64 0:4.8.5-11.el7
  perl-Thread-Queue.noarch 0:3.02-2.el7      perl-XML-Parser.x86_64 0:2.41-10.el7
```

圖 26　安裝開發套件(續)

資料來源：本研究整理

```
gettext-devel.x86_64 0:0.18.2.1-4.el7   libgfortran.x86_64 0:4.8.5-11.el7   libquadmath.x86_64 0:4.8.5-11.el7
m4.x86_64 0:1.4.16-10.el7               mokutil.x86_64 0:0.9-2.el7          perl-Test-Harness.noarch 0:3.28-3.el7
perl-srpm-macros.noarch 0:1-8.el7       systemtap-client.x86_64 0:3.0-7.el7 systemtap-devel.x86_64 0:3.0-7.el7
```

圖 27　安裝開發套件(續)

資料來源：本研究整理

安裝 Node.js，它允許在後端（跳脫瀏覽器環境）運行如 JavaScript 等程式碼，如下指令，完成畫面如圖 28 所示。

```
$ brew update && brew install -v node
```

```
→  ~ brew update && brew install -v node
Already up-to-date.
==> Installing dependencies for node: patchelf, zlib, binutils, linux-headers, glibc, xz, gmp, mpfr, libmpc, isl, gcc, icu4c
==> Installing node dependency: patchelf
```

圖 28　安裝 Node.js

資料來源：本研究整理

3-2-2 安裝 testrpc

首先要安裝的是 testrpc，它提供一個方便快速的測試環境，每次執行時會創建十組帳號，並且有充足的資源提供測試用的運行，如下指令，完成畫面如圖 29 所示。

```
$ npm install -g ethereumjs-testrpc
```

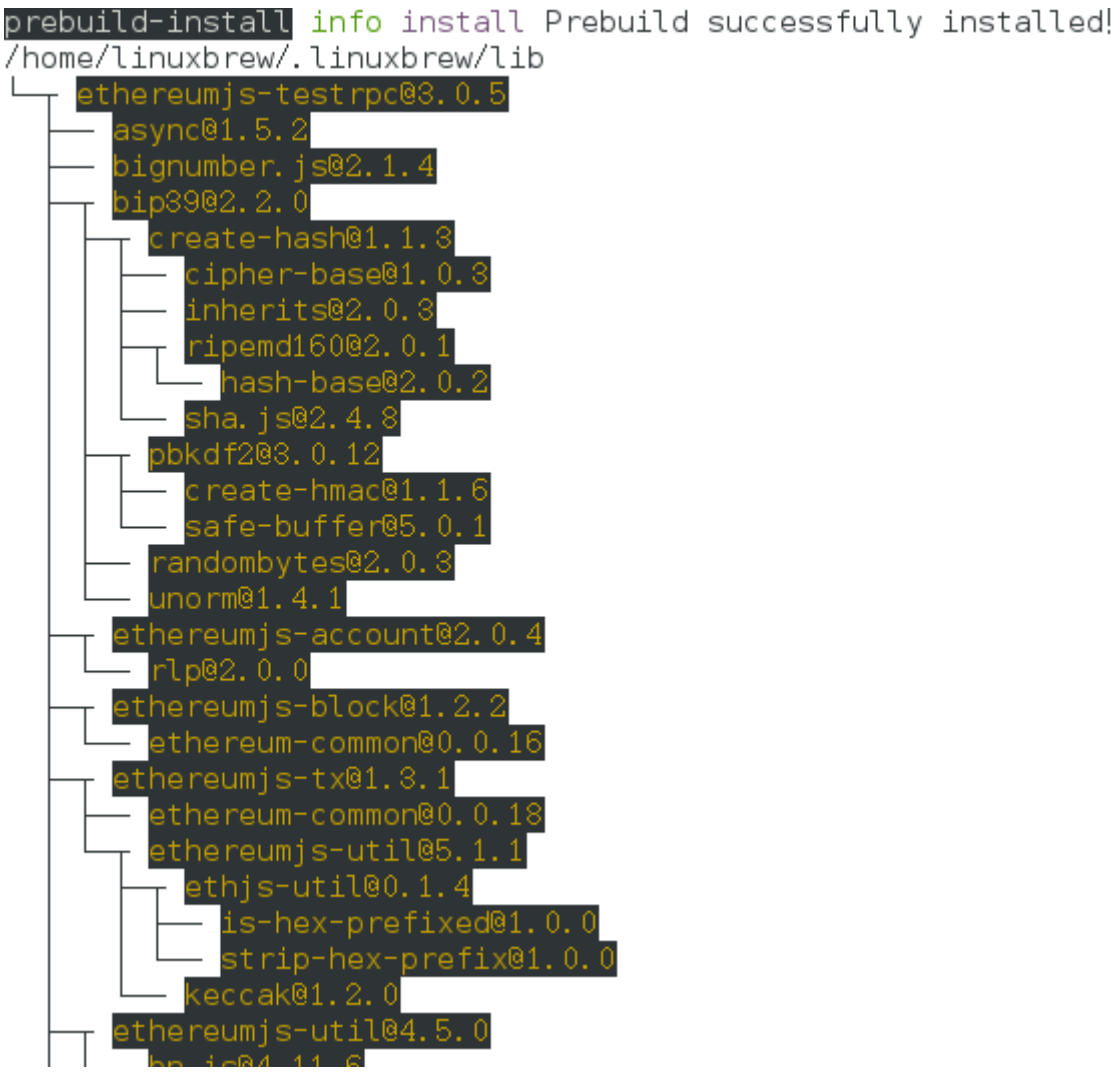

圖 29　安裝 testrpc

資料來源：本研究整理

安裝完輸入 testrpc，測試是否安裝完成，若出現如圖 30 所示，為安裝成功。

```
$ testrpc
```

```
→  ~ testrpc
EthereumJS TestRPC v3.0.3

Available Accounts
==================
(0) 0x15cc42b3b8f63ff1c2de640f714b665e5ddf9491
(1) 0x7d5860941331ccca551de7fe54c777ac689ef166
(2) 0x47da61900310a430c12bf6d666c849f679c266f2
(3) 0xb5600ccd5cde77d577f5dc47f20bd0d48908e60f
(4) 0xf36d8c8866a535e2afe7c75470ca1d26ab65d0b2
(5) 0x9b611d7f440104857a0a65a07c7352ecfa025aae
(6) 0x4ac28da7942250159a6078f57fcdb1149e5e8b0e
(7) 0xa12b0787b564b5597a4a363c0dba2808f1c31c04
(8) 0x14336a71764113cfe4fe5ba66eb161ad12d9a28f
(9) 0x7956a3f439bea7b5e6d06f6e635ecf034d6ede2d

Private Keys
==================
(0) 8442c5a0951a1c6867734df0296928e15b1d75c134784417de3da601f66e5f5b
(1) e26ba31bd634d686585fdccbce31db6806ec532d75d474f0358b41f590b9d146
(2) 51d6c54a2ca4748ca325e1f5e58a6d566d584023210fd17e0dc29851cc3b3409
(3) 36f9f4214c595ee2544443f54e1de8b9ac2eef7e0b686f49b02c5573258149e6
(4) 39b3f9169228bf831bb873ce068d2046d421331f5792357a76616a711f9126fd
(5) 6b407d27cc4c20cdeff09e36e44183ad5b62db2398a2d747f998298848d9bd6b
(6) a832ca0f45c5a07950f3cda998b0399b85325dbe197f53ad64f33ad3e1489544
(7) 932010a7808f36d59911f27e2b1009b37c8401551961475c5dba22a6f5d0081c
(8) fb99dc19b39d39f1f2798a081e8327621cbe6fff10483dc703157dea79a62f47
(9) 0f3287519a5db50a389352180b83f930d31a35556665ef54cc3c331244865696

HD Wallet
==================
Mnemonic:      hood keen legal harbor dolphin slot believe achieve wool merry film belt
Base HD Path:  m/44'/60'/0'/0/{account_index}

Listening on localhost:8545
```

圖 30　testrpc 安裝成功畫面

資料來源：本研究整理

3-2-3 安裝 truffle

truffle 是一個強大的開發框架，用於建立開發 Solidity，並提供其部屬和測試，自動化從合約的編譯到合約的部屬，須搭配 testrpc 來完成，如下指令，完成畫面如圖 31 所示。

```
$ npm install -g truffle
```

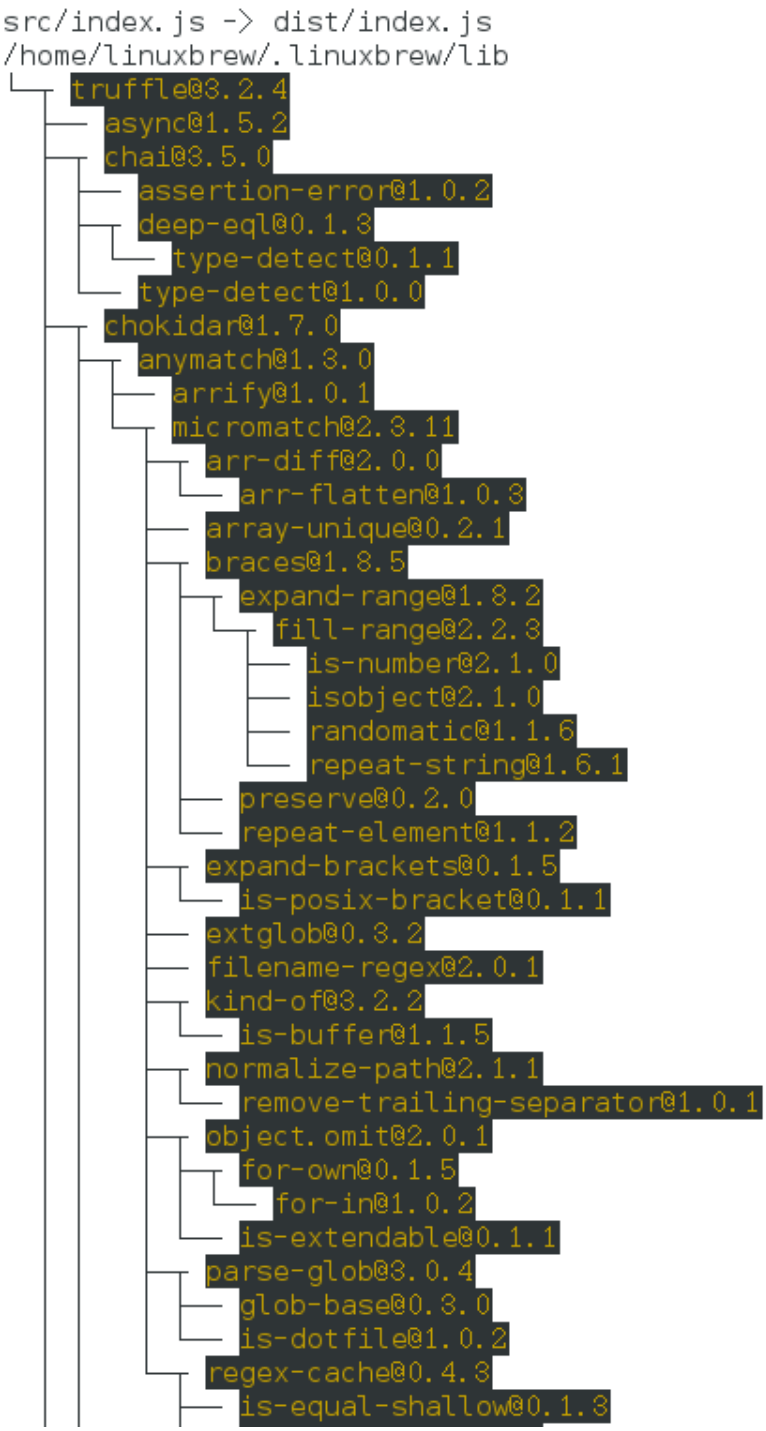

圖 31 安裝 truffle

資料來源：本研究整理

3-2-4 部署智能合約

照以下指令順序輸入，完成測試區塊鏈的建置，並創建 matching_demo 資料夾提供 JavaScript 與智能合約的擺放，完成畫面如圖 32 所示。

$ sudo apt-get install build-essential python

$ mkdir matching_demo

$ cd matching_demo

~/matching_demo$ npm install ethereumjs-testrpc web3

圖 32　於 matching_demo 下安裝 testrpc

資料來源：本研究整理

於創建的 matching_demo 下啟動 testrpc 之測試區塊鏈，如下指令，完成畫面如圖 33 所示。

~/matching_demo$ node_modules/.bin/testrpc

```
→  matching_demo node_modules/.bin/testrpc
EthereumJS TestRPC v3.0.5

Available Accounts
==================
(0) 0x7a6b8714f384e74e54699aec2bf870ebf295048c
(1) 0x94fc9a733abe5ed51128168f766eebf9758b8e7d
(2) 0x37414ec8c86680f9df7e4400e90979b8ea5a1e2d
(3) 0x2bb92b4d77d0119f72abaac58804cb1e7af3134f
(4) 0x51c7f0cb850cdd9c2b497723b5bd4217d346be24
(5) 0x6227eed053a7067130cc288c488370a7c9a2e4c5
(6) 0x1a8d01c5687874957e89c9a6949fafbeb1ecc9fa
(7) 0xb0002d7de659dcbfe344a255e2175f4e2c35df87
(8) 0x823d54df81729820a8ef05110893c786e2177869
(9) 0x4927a3205768bfe07be52e3163818d7078df3a46

Private Keys
==================
(0) ad96a8bcb2058d88479a4fd557fdfbe46f993b576f53509e2ac2e82070a18d7a
(1) 10effa8ed3afc4d39174b8618e876adc0a1579da56e730d4215336478f643c56
(2) e36a2ef6df8c85d8a86b0830ffb82f9c0cbf74f46568763ea2f3f950e3a3ea7d
(3) f03cb3940770551edc117652f9fe540c99335d073911519ca44a81e5738f39b4
(4) d0833b217ff80f7ff34ade9881fa835335b60438a968fad2c2939e1d456184ed
(5) 92c367a92977afaa8921e8eb8aae1257e36995fa93b18825257e25731965059b
(6) 03e0b94d27302f488d2e404926d2a5056250e7343e5fd41a5c642e903756906c
(7) f2ad77e5c28182f6634d1a2e61ab0f4b6a2e5d413beec33723074a9fd20aa2ce
(8) 8be7836dd18b22b3cef601fe38dd5a52df601091bb17c83180faf78491a3ec1d
(9) 9e97ec7e3d2ee5a8e26524bed3b5e231df2755fde2245a595b8db5d8bf372a31

HD Wallet
==================
Mnemonic:       emotion grow search danger subject erode empower ocean busy beach family wet
Base HD Path:   m/44'/60'/0'/0/{account_index}

Listening on localhost:8545
```

圖 33　於 matching_demo 下啟動 testrpc

資料來源：本研究整理

上述完成 testrpc 的建置後，接著另開一個終端機，並輸入以下指令：

~/matching_demo$ node

> Web3 = require('web3')

> web3 = new Web3(new Web3.providers.HttpProvider("http://localhost:8545"));

> code = fs.readFileSync('Voting.sol').toString()

> contract = web3.eth.compile.solidity(code)

> MatchingContract = web3.eth.contract(contract.info.abiDefinition)

> deployedContract = MatchingContract.new({data: contract.code, from: web3.eth.accounts[0], gas: 4700000})

> contractInstance = MatchingContract.at(deployedContract.address)

如下圖 34~36 所示為結果介面。

圖 34 部屬智能合約命令行介面結果

資料來源：本研究整理

```
  sign: { [Function: send] request: [Function: bound ], call: 'eth_sign' },
  compile: { solidity: [Object], lll: [Object], serpent: [Object] },
  submitWork: { [Function: send] request: [Function: bound ], call: 'eth_submitWork' },
  getWork: { [Function: send] request: [Function: bound ], call: 'eth_getWork' },
  coinbase: [Getter],
  getCoinbase: { [Function: get] request: [Function: bound ] },
  mining: [Getter],
  getMining: { [Function: get] request: [Function: bound ] },
  hashrate: [Getter],
  getHashrate: { [Function: get] request: [Function: bound ] },
  syncing: [Getter],
  getSyncing: { [Function: get] request: [Function: bound ] },
  gasPrice: [Getter],
  getGasPrice: { [Function: get] request: [Function: bound ] },
  accounts: [Getter],
  getAccounts: { [Function: get] request: [Function: bound ] },
  blockNumber: [Getter],
  getBlockNumber: { [Function: get] request: [Function: bound ] },
  protocolVersion: [Getter],
  getProtocolVersion: { [Function: get] request: [Function: bound ] },
  iban:
   { [Function: Iban]
     fromAddress: [Function],
     fromBban: [Function],
     createIndirect: [Function],
     isValid: [Function] },
  sendIBANTransaction: [Function: bound transfer] },
transactionHash: null,
address: '0x9bec7ec8f211ec50072aaac612429f7171489c68',
abi:
 [ { constant: true,
     inputs: [Object],
     name: 'guyA',
     outputs: [Object],
     payable: false,
     type: 'function' },
```

圖 35　部屬智能合約命令行介面結果(續)

資料來源：本研究整理

```
  { constant: true,
    inputs: [Object],
    name: 'guyB',
    outputs: [Object],
    payable: false,
    type: 'function' },
  { constant: false,
    inputs: [Object],
    name: 'printGuyList',
    outputs: [Object],
    payable: false,
    type: 'function' },
  { constant: true,
    inputs: [Object],
    name: 'guyList',
    outputs: [Object],
    payable: false,
    type: 'function' },
  { constant: false,
    inputs: [Object],
    name: 'addGuyList',
    outputs: [],
    payable: false,
    type: 'function' },
  { constant: true,
    inputs: [Object],
    name: 'bytes32ToString',
    outputs: [Object],
    payable: false,
    type: 'function' },
  { constant: true,
    inputs: [Object],
    name: 'guyC',
    outputs: [Object],
    payable: false,
    type: 'function' },
```

圖 36　部屬智能合約命令行介面結果-3

資料來源：本研究整理

3-2-5 與 JavaScript 介接

在 3-1-4 完成智能合約的部署後，透過下面的設定來與 JavaScript 做介接，終端機輸入 contractInstance.address，以取得合約的位址，如圖 37 所示，取得並覆寫在 JavaScript 的程式碼中，方能與智能合約順利溝通。

```
> contractInstance.address
'0x9bec7ec8f211ec50072aaac612429f7171489c68'
```

圖 37　取得智能合約位址

資料來源：本研究整理

對於和智能合約內的參數及功能函式做溝通，必須由 JSON 格式進行包裝，宣告在 JavaScript 中，其中包含了函式的名稱、輸入值、輸出值，這會在 3-4 詳細說明。

3-3 系統使用流程

多對多配對雛形系統之網頁介面，如圖 38 所示，使用者可透過介面得知可選配對的對象、輸入自己的喜好順序並送出、查詢自己的配對結果，最後可以確認是否接受系統所配對的結果，若不接受則可重調整輸入新的喜好順序，簡易流程如圖 39 所示。

圖 38　多對多配對雛形系統網頁介面

資料來源：本研究整理

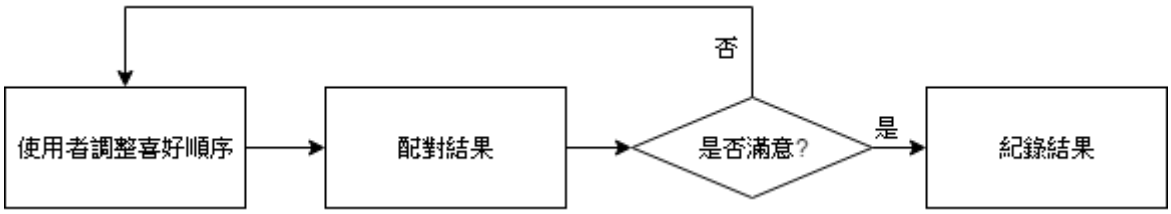

圖 39　系統簡易操作流程

資料來源：本研究整理

3-4 多對多配對雛形系統實作：Solidity 與 JavaScript

3-4-1 多對多配對雛形系統實作：Solidity 程式說明

利用 Solidity 編寫智能合約，必須給予編譯的版本，否則無法執行，如圖 40。

```
pragma solidity ^0.4.6;

contract Matching {
```

圖 40 智能合約 Matching

資料來源：本研究整理

宣告變數，mapping 擁有 map 的特性，而陣列可宣告成 bytes32 與 string 的類型，如圖 41。

```
mapping (bytes32    => bytes32) public engagedTo;

bytes32[] public guyList;
string[] public guyA;
string[] public guyB;
string[] public guyC;
string[] public guyD;
```

圖 41　智能合約中變數宣告

資料來源：本研究整理

如表 10，智能合約內的函式主要為 addGuyList()：透過 JavaScript 傳回的值添加並儲存喜好順序；printGuyList()：取得陣列的第某筆資料；getListLength()：可以取得喜好順序的陣列長度，主要提供開發者後台監控，透過終端機介面操作；addResult()：可將 JavaScript 索回傳的配對結果回傳，透過智能合約儲存於 mapping 的變數中；getResult()：同樣主要是提供開發者在終端機介面操作，確定結果的配對是否無誤，程式碼如圖 42 所示。

表 10　智能合約功能函式

函式名稱	功能簡述	輸入值	回傳值
addGuyList	添加喜好順序	**bytes32**	
printGuyList	取得某筆喜好順序	**uint8**	**string**
getListLength	取得喜好順序陣列長度		**uint256**
addResult	添加配對結果	**bytes32, bytes32**	
getResult	取得配對結果	**bytes32**	**string**

資料來源：本研究整理

```
function addGuyList(bytes32 name){
    guyList.push(name);
}

function printGuyList(uint8 i) returns (string){
    return bytes32ToString(guyList[i]);
}

function getListLength() returns(uint256){
    return guyList.length;
}

function addResult(bytes32 boy, bytes32 girl){
    engagedTo[boy] = girl;
}

function getResult(bytes32 name) returns (string){
    return bytes32ToString(engagedTo[name]);
}
```

圖 42　智能合約功能函式

資料來源：本研究整理

智能合約中添加一個 bytes32ToString 的函式，提供 bytes32 轉型成 string 的型態做

回傳，程式碼如圖 43。

```
//bytes32ToString
  function bytes32ToString(bytes32 x) constant returns (string) {
    bytes memory bytesString = new bytes(32);
    uint charCount = 0;
    for (uint j = 0; j < 32; j++) {
        byte char = byte(bytes32(uint(x) * 2 ** (8 * j)));
        if (char != 0) {
            bytesString[charCount] = char;
            charCount++;
        }
    }
    bytes memory bytesStringTrimmed = new bytes(charCount);
    for (j = 0; j < charCount; j++) {
        bytesStringTrimmed[j] = bytesString[j];
    }
    return string(bytesStringTrimmed);
  }
```

圖 43　bytes32 轉 String 之功能函式

資料來源：本研究整理

3-4-2 多對多配對雛形系統實作：JavaScript 與 html

程式說明

此段為 html 的格式，簡單設計多對多配對雛形系統的介面，並與 JavaScript 做互動，如圖 44。

```
<!DOCTYPE html>
<html>
<head>
  <link href='https://fonts.googleapis.com/css?family=Open+Sans:400,700' rel='stylesheet' type='text/css'>
  <link href='https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css' rel='stylesheet' type='text/css'>
  <meta charset="UTF-8">
</head>
<body class="container">
  <h1>多對多配對雛型系統</h1>
  <div class="table-matching">
    <table class="table table-matching">
      <thead>
        <tr>
          <th>男生</th>
          <th>喜好順序</th>
          <th>配對結果</th>
        </tr>
      </thead>
      <tbody>
        <tr>
          <td>A</td>
          <td id="guy-1"></td>
```

```
      <td id="guyPrefer-1"></td>
    </tr>
    <tr>
      <td>B</td>
      <td id="guy-2"></td>
      <td id="guyPrefer-2"></td>
    </tr>
    <tr>
      <td>C</td>
      <td id="guy-3"></td>
      <td id="guyPrefer-3"></td>
    </tr>
            <tr>
      <td>D</td>
      <td id="guy-4"></td>
      <td id="guyPrefer-4"></td>
    </tr>
  </tbody>
     <thead>
    <tr>
      <th>女生</th>
      <th>喜好順序</th>
      <th>配對結果</th>
    </tr>
  </thead>
     <tbody>
    <tr>
      <td>W</td>
      <td id="girl-1"></td>
      <td id="girlPrefer-1"></td>
    </tr>
    <tr>
      <td>X</td>
      <td id="girl-2"></td>
```

```
          <td id="girlPrefer-2"></td>
        </tr>
        <tr>
          <td>Y</td>
          <td id="girl-3"></td>
          <td id="girlPrefer-3"></td>
        </tr>
        <tr>
          <td>Z</td>
          <td id="girl-4"></td>
          <td id="girlPrefer-4"></td>
        </tr>
      </tbody>
    </table>
        </div>
        <select name="who" id="who">
                <option value="guy-1">A</option>
                <option value="guy-2">B</option>
                <option value="guy-3">C</option>
                <option value="guy-4">D</option>
                <option value="girl-1">W</option>
                <option value="girl-2">X</option>
                <option value="girl-3">Y</option>
                <option value="girl-4">Z</option>
        </select>
    <input type="text" id="prefer1" /><input type="text" id="prefer2" />
        <input type="text" id="prefer3" /><input type="text" id="prefer4" />
        <a href="#" onclick="addPrefer()" class="btn btn-primary">送出喜好順序
</a></br>
        <a href="#" onclick="defaultSetting()" class="btn btn-primary">預設值
</a>
        <a href="#" onclick="doMarriage()" class="btn btn-primary">開始配
對!</a>
```

```
        <a href="#" onclick="confirmResult()" class="btn btn-primary">確認結果
</a></br>
        <div id="result"></div>
</body>
<script
src="https://cdn.rawgit.com/ethereum/web3.js/develop/dist/web3.js"></script>
<script src="https://code.jquery.com/jquery-3.1.1.slim.min.js"></script>
<script src="./index.js"></script>
</html>
```

圖 44　多對多配對雛形系統-html 程式碼

資料來源：本研究整理

此為 JavaScript 的前段，主要部分是與智能合約的介接與設定，JSON 格式宣告將與智能合約互動的函式內容，而要注意的是，要將終端機輸入 contractInstance.address 後的回傳值覆蓋於 VotingContract.at(‘’)之中，才能順利完成 JavaScript 與智能合約的介接，如圖 45。

```
web3 = new Web3(new Web3.providers.HttpProvider("http://localhost:8545"));
abi =
JSON.parse('[{"constant":false,"inputs":[{"name":"","type":"bytes32"}],"name":"a
ddGuyList","outputs":[],"payable":false,"type":"function"},{"constant":false,"input
s":[{"name":"","type":"uint8"}],"name":"printGuyList","outputs":[{"name":"","typ
e":"string"}],"payable":false,"type":"function"},{"constant":false,"inputs":[],"name
":"getListLength","outputs":[{"name":"","type":"uint256"}],"payable":false,"type":
"function"},{"constant":false,"inputs":[{"name":"boy","type":"bytes32"},{"name":"
girl","type":"bytes32"}],"name":"addResult","outputs":[],"payable":false,"type":"fu
nction"}]')
VotingContract = web3.eth.contract(abi);
web3.eth.defaultAccount = web3.eth.accounts[0];
```

```
// In your nodejs console, execute contractInstance.address to get the address at which the contract is deployed and change the line below to use your deployed address
contractInstance = VotingContract.at('0x9bec7ec8f211ec50072aaac612429f7171489c68');
```

圖 45　JavaScript 前段介接部分

資料來源：本研究整理

如圖 46 所示，JavaScript 與智能合約的介接所需的 JSON 格式，以 addGuyList()函式為例，包含了函式名稱、型態、輸入輸出值的名稱與型態；而如圖 47 則為 addResult()的函式，因為其輸入值有兩個 bytes32，因此在輸入值部分的格式必須輸入兩個變數的名稱與型態。


```
[
  {
    "constant":false,
    "inputs":[
      {
        "name":"",
        "type":"bytes32"
      }
    ],
    "name":"addGuyList",
    "outputs":[
    ],
    "payable":false,
    "type":"function"
  },
  {…},
  {…},
  {…}
]
```


圖 46　介接用 JSON

資料來源：本研究整理

```
[
  {…},
  {…},
  {…},
  {
    "constant":false,
    "inputs":[
      {
        "name":"boy",
        "type":"bytes32"
      },
      {
        "name":"girl",
        "type":"bytes32"
      }
    ],
    "name":"addResult",
    "outputs":[
    ],
    "payable":false,
    "type":"function"
  }
]
```


圖 47　介接用 JSON(續)

資料來源：本研究整理

如圖 48，addPrefer()在 JavaScript 中執行將喜好順序添加至介面，以及透過智能合約將其存至陣列中，這邊要特別提到的是，智能合約所取出的 string 值必須透過 call()的內建函式轉譯成一般看的懂的文字，而非一串程式碼。

```
function addPrefer(){
        var order = '';
        who = $("#who").val();
        var length = contractInstance.getListLength.call().toString();
        for (var i = 1 ; i <= 4 ; i++){
                prefer = $("#prefer" + i).val();
                contractInstance.addGuyList();
                order += prefer + "   ";
        }
        $("#" + who).html(order);
```

```
}
```

圖 48　JavaScript 添加喜好度之功能函式

資料來源：本研究整理

圖 49 為延遲接受演算法的部分，把每個使用者宣告成一個 person 的物件，以準備做多對多的配對。

```
function Person(name) {

    var candidateIndex = 0;

    this.name = name;
    this.fiance = null;
    this.candidates = [];

    this.rank = function(p) {
        for (i = 0; i < this.candidates.length; i++)
            if (this.candidates[i] === p) return i;
        return this.candidates.length + 1;
    }

    this.prefers = function(p) {
        return this.rank(p) < this.rank(this.fiance);
    }

    this.nextCandidate = function() {
        if (candidateIndex >= this.candidates.length) return null;
        return this.candidates[candidateIndex++];
    }

    this.engageTo = function(p) {
        if (p.fiance) p.fiance.fiance = null;
```

```
        p.fiance = this;
        if (this.fiance) this.fiance.fiance = null;
        this.fiance = p;
    }

    this.swapWith = function(p) {
        console.log("%s & %s swap partners", this.name, p.name);
        var thisFiance = this.fiance;
        var pFiance = p.fiance;
        this.engageTo(pFiance);
        p.engageTo(thisFiance);
    }
}
```

圖 49　JavaScript 延遲接受演算法(宣告物件)

資料來源：本研究整理

圖 50 為多對多配對，延遲接受演算法的核心，將圖 45 所宣告的物件做配對，並回傳結果，而圖 51 則為將網頁上的資訊整理，並開始做物件宣告及配對演算法。

```
function engageEveryone(guys) {
    var done;
    do {
        done = true;
        for (var i = 0; i < guys.length; i++) {
            var guy = guys[i];
            if (!guy.fiance) {
                done = false;
                var gal = guy.nextCandidate();
                if (!gal.fiance || gal.prefers(guy))
                    guy.engageTo(gal);
            }
        }
```

```
    } while (!done);
}
```

圖 50　JavaScript 延遲接受演算法

資料來源：本研究整理

```
 function doMarriage() {

var A   = new Person("A");
var B   = new Person("B");
var C   = new Person("C");
var D   = new Person("D");
var W   = new Person("W");
var X   = new Person("X");
var Y   = new Person("Y");
var Z   = new Person("Z");

var guys = [A, B, C, D];
var gals = [W, X, Y, Z];

pushGuysCandidates(A, $("#guy-1").html().trim().split("   "), W, X, Y, Z);
pushGuysCandidates(B, $("#guy-2").html().trim().split("   "), W, X, Y, Z);
pushGuysCandidates(C, $("#guy-3").html().trim().split("   "), W, X, Y, Z);
pushGuysCandidates(D, $("#guy-4").html().trim().split("   "), W, X, Y, Z);
pushGalsCandidates(W, $("#girl-1").html().trim().split("   "), A, B, C, D);
pushGalsCandidates(X, $("#girl-2").html().trim().split("   "), A, B, C, D);
pushGalsCandidates(Y, $("#girl-3").html().trim().split("   "), A, B, C, D);
pushGalsCandidates(Z, $("#girl-4").html().trim().split("   "), A, B, C, D);

engageEveryone(guys);

for (var i = 0; i < guys.length; i++) {
                    switch (guys[i].name){
```

```
                    case 'A' :
                            $("#guyPrefer-1").html(guys[i].fiance.name);
                            break;
                    case 'B' :
                            $("#guyPrefer-2").html(guys[i].fiance.name);
                            break;
                    case 'C' :
                            $("#guyPrefer-3").html(guys[i].fiance.name);
                            break;
                    case 'D' :
                            $("#guyPrefer-4").html(guys[i].fiance.name);
                            break;
                    default :
                            break;
            }
            switch (guys[i].fiance.name){
                    case 'W' :
                            $("#girlPrefer-1").html(guys[i].name);
                            break;
                    case 'X' :
                            $("#girlPrefer-2").html(guys[i].name);
                            break;
                    case 'Y' :
                            $("#girlPrefer-3").html(guys[i].name);
                            break;
                    case 'Z' :
                            $("#girlPrefer-4").html(guys[i].name);
                            break;
                    default :
                            break;
            }
    }
```

```
}
```

圖 51　JavaScript 延遲接受演算法(續)

資料來源：本研究整理

將網頁端使用者所輸入的資訊傳遞進 person 的物件中，如圖 52 所示。

```
function pushGuysCandidates(who, order, W, X, Y, Z){
        for(var i = 0 ; i < order.length ; i++){

                switch (order[i]){
                        case 'W' :
                                who.candidates.push(W);
                                break;
                        case 'X' :
                                who.candidates.push(X);
                                break;
                        case 'Y' :
                                who.candidates.push(Y);
                                break;
                        case 'Z' :
                                who.candidates.push(Z);
                                break;
                        default :
                                break;
                }

        }
}

function pushGalsCandidates(who, order, A, B, C, D){
        for(var i = 0 ; i < order.length ; i++){
                switch (order[i]){
                        case 'A' :
```

```
                        who.candidates.push(A);
                        break;
                case 'B' :
                        who.candidates.push(B);
                        break;
                case 'C' :
                        who.candidates.push(C);
                        break;
                case 'D' :
                        who.candidates.push(D);
                        break;
                default :
                        break;
            }
        }
}
```

圖 52　JavaScript 延遲接受演算法(輸入喜好順序)

資料來源：本研究整理

最後一個 JavaScript 函式為 confirmResult()，若使用者接受本次配對的結果，則會透過 contractInstance.addResult()的函式與智能合約互動，將配對的結果回傳並儲存至智能合約的變數中，如圖 53 所示。

```
function confirmResult(){

        var result = "";
        result += "A 和 " + $("#guyPrefer-1").html() + " 完成配對</br>";
        result += "B 和 " + $("#guyPrefer-2").html() + " 完成配對</br>";
        result += "C 和 " + $("#guyPrefer-3").html() + " 完成配對</br>";
        result += "D 和 " + $("#guyPrefer-4").html() + " 完成配對";
        $("#result").html(result);
```

```
        contractInstance.addResult("A", $("#guyPrefer-1").html());
        contractInstance.addResult("B", $("#guyPrefer-2").html());
        contractInstance.addResult("C", $("#guyPrefer-3").html());
        contractInstance.addResult("D", $("#guyPrefer-4").html());
}
```

圖 53　JavaScript 儲存配對結果之功能函式

資料來源：本研究整理

4. 以區塊鏈技術為基礎之多對多配對雛形系統驗證與討論

4-1 系統之配對結果驗證

本研究之系統以穩定婚姻問題來做情境，並以四對四來做多對多配對的範例，以ABCD 做為男生方，WXYZ 做為女生方，各自交出自己的喜好順序表，如表 11 和表 12 所示，最後系統的配對結果如表 13，其配對的結果和一般套用延遲接受演算法的結果一致，以驗證本研究之系統的配對是可靠的。

表 11　男生喜好順序表

	1	2	3	4
A	W	X	Y	Z
B	Y	X	W	Z
C	W	Y	X	Z
D	X	Z	W	Y

資料來源：本研究整理

表 12　女生喜好順序表

	1	2	3	4
W	B	D	C	A
X	C	A	D	B
Y	B	C	D	A
Z	A	B	D	C

資料來源：本研究整理

表 13　配對結果

配對組	男生	女生
配對 1	**A**	**X**
配對 2	B	Y
配對 3	C	W
配對 4	D	Z

資料來源：本研究整理

4-2 系統開發遭遇之問題檢討

透過以太坊所提供的 Solidity 語言來撰寫智能合約，由於區塊鏈的運算是要分散式儲存在各節點上，因此它特別要求智能合約上運算量及儲存量要做到輕量化，限制了智能合約上的運算量，容易導致成本太高而出現錯誤，以致系統無法在智能合約上執行多對多配對的延遲接受演算法，取而代之的作法為將演算法的運算調整至網頁端的 JavaScript 去做執行，最後再將配對的結果透過呼叫智能合約上的功能函式來儲存。

4-3 多對多配對雛形系統之延伸應用

由於 JavaScript 的運算能力有限，如果多對多的配對遇到大量使用者的情況，會讓程式內運作的陣列變得複雜，本研究認為未來對此需求上，可套用 Mathematica 系統做外接，該程式語言擁有數學的高技術計算系統，廣泛的被運用在科學、工程、數學、計算等領域，透過其開發，程式相依性、軟體可靠度都大幅地提升，可與開發環境做整合，它分為兩個部分，內核和前端，內核會運算後回傳完成結果，除了預設地前端，開發者可以透過其協定所提供的通訊，來與其他應用程式做通用的介面增加了 Mathematica 的適用性，如此用來彌補大量配對運算在 JavaScript 或是智能合約上的不足。

5. 研究成果與未來研究方向

5-1 研究成果

本研究所發展的多對多配對雛形系統，能夠透編寫於系統的延遲接受演算法，對於使用者所提出的喜好順序做多對多的配對，如何能夠運作在區塊鏈的技術上，實現 Gale–Shapley 的演算法，編寫於智能合約中，讓使用者互動並回饋使用經驗加以改善，完成在區塊鏈上的實作，並比較與舊有系統的差異。本研究希望藉由此 Solidity 實作，幫助多對多的配對案例能夠在區塊鏈上更有發展性，透過延遲演算法的配對機制，讓更多的交易能夠基於這方法找到最好的配對組合，獲得最大的利益。

5-2 研究限制

本研究在撰寫多對多配對的智能合約，主要遇到的困難是由於以太坊的區塊鏈儲存希望做到輕量化，而在智能合約的編譯和執行有所限制，例如陣列的類型宣告及儲存的次數與大小，常導致因為交易成本過高而出現程式上的錯誤，為了繼續完成本研究的多對多配對實作，才考慮將延遲接受演算法，也就是複雜的計算部分調整至網頁端的 JavaScript 做運算，喜好順序與配對結果的儲存仍透過與智能合約的溝通，儲存於測試的區塊鏈上，達到交易資訊使用區塊鏈來做分散式的儲存，亦保證其安全性及完整性。

5-3 未來研究方向

本研究針對多對多配對的演算法實作在區塊鏈的智能合約上，未來可以有更多的配對例子可以沿用此方法去紀錄喜好順序及配對結果，並且讓使用者決定是否滿意此次的配對，才送出最終的結果。但在對於各種應用仍有待更多的調整，包括了使用者是否有最後接受配對結果的階段、配對資訊的公開程度、配對的時間設定與流程，都必須依照各種應用與產業的需求不同，而有不同程度的調整。

參考文獻

[1] 陳昶吾，“以太坊初探，” 2017. [線上]. Available: https://www.youtube.com/watch?v=uFBu2P1mwFU.

[2] 杜宏毅，“Blockchain 的前世今生與未來，” 2016. [線上]. Available: http://www.twse.com.tw/ch/products/broker_service/download/d105061508.pdf.

[3] 工业和信息化部信息化和软件服务业司，“中国区块链技术和应用发展白皮书，” 2016. [線上]. Available: https://img2.btc123.com/file/0/chinabolckchaindevwhitepage2016.pdf.

[4] 劉俊宏，“穩定配對問題，” [線上]. Available: https://www.google.com.tw/url?sa=t&rct=j&q=&esrc=s&source=web&cd=2&ved=0ahUKEwj6zNv--5bUAhVBkZQKHSmTCmsQFggtMAE&url=http%3A%2F%2Fwww.math.ntu.edu.tw%2F~msa%2Fact%2Fmathcamp%2F95page%2Flecture%2FB.doc&usg=AFQjCNFL1bo-3h4WwVozEI7KliXlszsHzg&sig2=HaLQgX1tjVG.

[5] 顧森，“穩定婚姻問題和 Gale-Shapley 算法，” [線上]. Available: http://www.rocidea.com/roc-6058.aspx.

[6] S. Nakamoto, “Bitcoin: A Peer-to-Peer Electronic Cash System,” 2008. [線上]. Available: https://bitcoin.org/bitcoin.pdf.

[7] M. Murthy, “Full Stack Hello World Voting Ethereum Dapp Tutorial ,” 2017. [線上]. Available: https://medium.com/@mvmurthy/full-stack-hello-world-voting-ethereum-dapp-tutorial-part-1-40d2d0d807c2#.gars2468e.

[8] Gcoin, “Gcoin white paper Chinese,” 2016. [線上]. Available: https://github.com/hanky312/gcoin-community/wiki/Gcoin-white-paper-Chinese#_Gcoin_.

[9] Ethereum, “Ethereum Builder's Guide,” 2015. [線上]. Available: https://ethereumbuilders.gitbooks.io/guide/content/en/index.html.

[10] Stable marriage problem,” [線上]. Available: https://rosettacode.org/wiki/Stable_marriage_problem.

[11] Wikipedia, “Stable marriage problem,” [線上]. Available: https://en.wikipedia.org/wiki/Stable_marriage_problem.

[12] Wikipedia, “Mathematica,” [線上]. Available: https://zh.wikipedia.org/wiki/Mathematica.

[13] RSK, “Key Metrics of Blockchain Platforms,” RSK, [線上]. Available: https://docs.google.com/spreadsheets/d/1DQ77onGnHfJOoRSqTLmIkhuVK5CAbs-Fgqb6UoGMfVM/edit#gid=0.