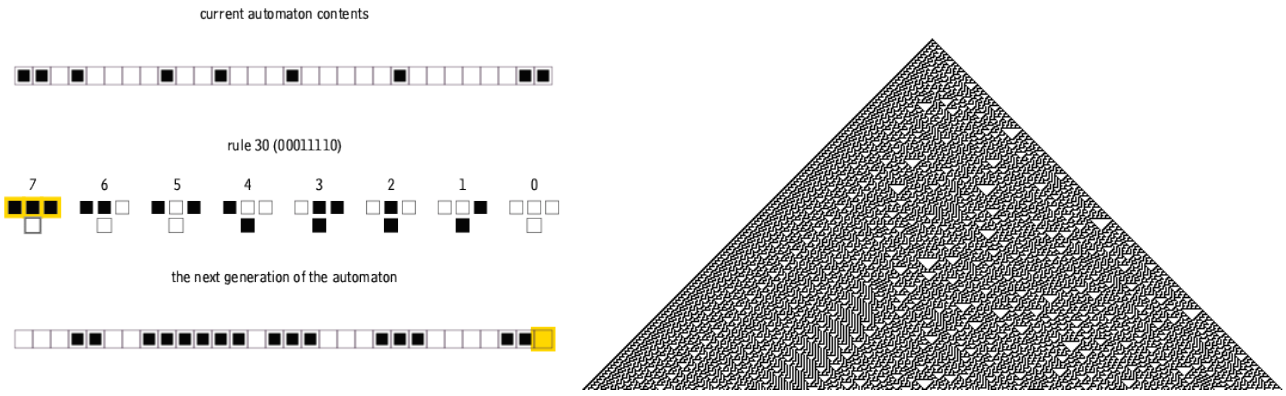
康威生命遊戲 Conway's Game of Life

由來

康威生命遊戲要先從細胞自動機開始說起，細胞自動機是由約翰·馮紐曼在 1950 年代為模擬生物細胞的自我複製而提出的。細胞自動機是排成一排的方格，或者說是細胞，每個細胞都有兩種狀態，**生**或**死**，並會與相鄰的細胞互動，每回合的每個細胞都會去確認相鄰細胞的狀態，並依設計者的規則，判斷自身的下個狀態，即**不變**、**出生**或**死亡**。回合開始前要先設定細胞的狀態，接著開始模擬回合，經過若干回合後讀取所有的細胞的狀態即是計算結果。

由於規則可以隨意被設計與更改，部分被證明過的規則又稱通用電腦，意思是「電腦能解決的問題，都能用細胞自動機解決」。



▲ 左圖為 Rule 30 的規則，而右圖則是把每一回合的結果疊起來形成的圖片。

圖片來源 Google

介紹

細胞自動機理論在提出後並未馬上受到重視，而是到二十年後，英國數學家約翰·何頓·康威設計了「Conway's Game of Life 康威生命遊戲」，並發表於美國當年十月的《Scientific American》雜誌上，才引起了數學家與科學家們的關注，紛紛投入研究並發表新發現。康威生命遊戲相當於二維平面上的細胞自動機，在發明後不久就被證明為通用電腦，可以在遊戲內製作邏輯閘，做成小型電腦進行數學運算，甚至還能製作出生命遊戲中的生命遊戲，Life in Life。

如果在 Google 中查詢「生命遊戲」可以發現背景有小彩蛋。

遊戲內每個細胞會與自身周圍的八個細胞互動，主要由四項規則構成，規則如下：

- 細胞為存活狀態時，如果周圍少於兩個存活細胞，該細胞會**變成死亡**狀態。（模擬生命稀缺）
- 細胞為存活狀態時，如果周圍有兩個或三個存活細胞，該細胞**保持原樣**。（模擬穩定狀態）
- 細胞為存活狀態時，如果周圍大於三個存活細胞，該細胞會**變成死亡**狀態（模擬生命過多）
- 細胞為死亡狀態時，如果周圍有三個存活細胞，該細胞會**變成存活**狀態。（模擬繁殖）

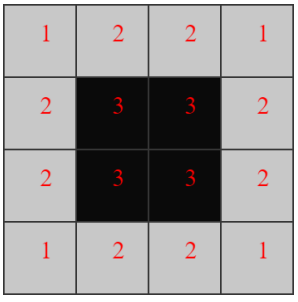
遊戲以回合制，每回合分為兩步驟，先計算整個地圖中細胞的下一個狀態，接著更新所有細胞的狀態，然後再計算下一回合，所以不會有計算順序的問題。
以下我會將細胞周圍的存活細胞視為鄰居以利於解釋和理解。

常見細胞組合

穩定狀態

板凳 (Block)

生命遊戲中的細胞可以和周圍的細胞互相作用，有的時候會有一團細胞不管經過多少回合整體都不會改變，我們就稱它為穩定狀態。最常見的穩定狀態為板凳，它是以 2x2 的細胞所組成，這些細胞周圍都有三個鄰居所以保持不變，而周圍的死亡細胞都沒有三個鄰居，所以不會變成活細胞。

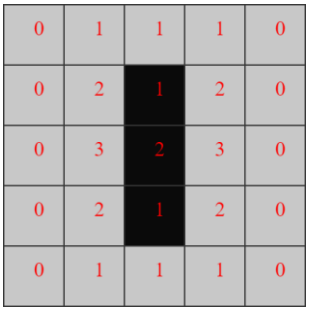


▲ 板凳 (Block)

震盪狀態

信號燈 (Blinker)

震盪狀態是一群細胞經過若干回合後會回到最初的樣子，最常見的例子是 Blinker 信號燈，由三個一排的細胞組成，可以觀察到兩個端點的細胞只有一個鄰居，所以在下回合會死亡，而側邊兩個死細胞有三個鄰居，所以下回合會變活細胞。而整個圖形會旋轉 90 度，故信號燈會以兩回合為周期，呈現直橫的擺動方式。



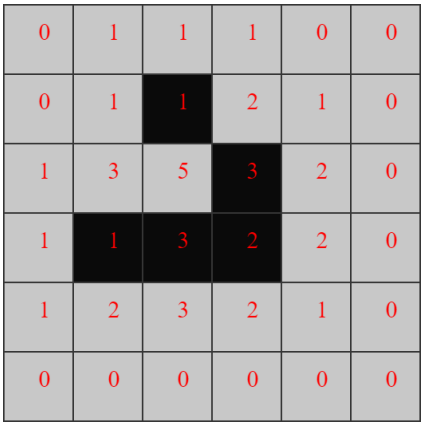
▲ 信號燈 (Blinker)

https://rurl.page.link/blinker

移動狀態

滑翔機 (Glider)

移動狀態是讓生命遊戲被證明為通用電腦的關鍵，和正當狀態一樣具有週期性，每個周期結束後整體圖形會移動若干格，最經典的例子為滑翔機，由五個細胞組成（如右圖），滑翔機以四回合為周期，每周期會向右下移動一格。滑翔機是生命遊戲中斜向移動最快的裝置。

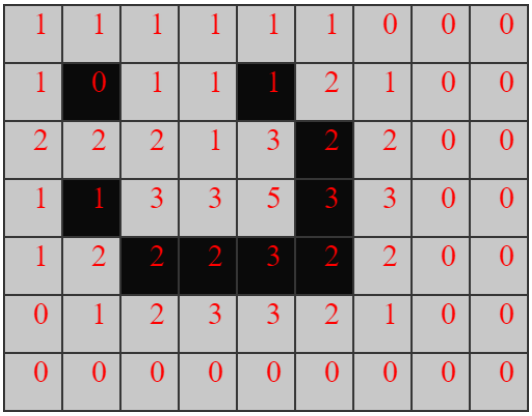


▲滑翔機 (Glider)

https://rurl.page.link/glider

輕型太空船 Light-weight spaceship(LWSS)

跟滑翔機相同以四回合為週期，但是以橫向移動。輕型太空船是生命遊戲中橫向移動最快的裝置。另外還有太空船以及重型太空船。



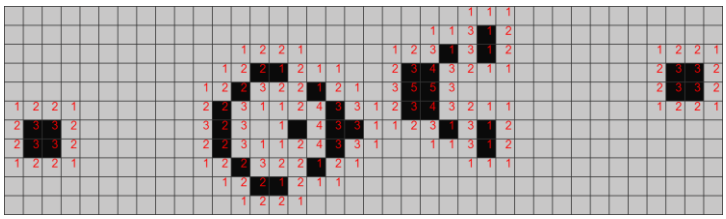
▲輕型太空船

Light-weight spaceship(LWSS)

https://rurl.page.link/spaceship

高斯帕機槍 Gosper's glider gun

康威最初推測，任何起始配置都無法讓細胞數量無限增長，但在 1970 年 11 月麻省理工學院的一個團隊證實了這個推測是錯誤的。此團隊由比 Bill Gosper 領導，因此圖形命名為 Gosper's glider gun 高斯帕機槍。



▲高斯帕機槍 Gosper's glider gun

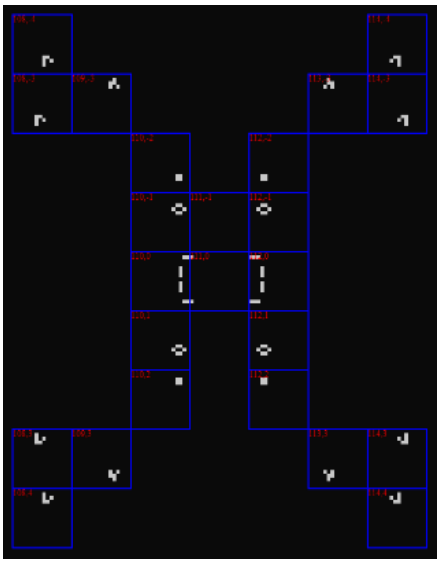
https://rurl.page.link/gosper_glider_gun

單人遊戲開發

在單人遊戲裡，先做出生命遊戲的演算法最為基礎，雖然整體規則簡單，但在實際製作時卻很困難，每次回合都要去計算整張地圖的細胞狀態並更新。最簡單且常見的做法就是挨家挨戶的判斷細胞周圍的鄰居數量，但這種方式的效率會非常差，而且我發現 Google 上大多數人都採用這種方式去實作，讓我非常的不解，使用這種方法不但要限定地圖大小，還會犧牲額外的電腦效能，遊玩體驗很差。

關於這點，我想到與眾不同的方法，與其在每次計算時都去尋找細胞周圍的鄰居數量，倒不如直接在細胞切換生死時，告訴周圍八個細胞（不論死活）鄰居的數量是否要增加或減少，這樣每個細胞就會擁有自己身邊的鄰居數量，接著回合開始時每個細胞就只需要依自身紀錄的鄰居數量，來決定自己是否要更改狀態。

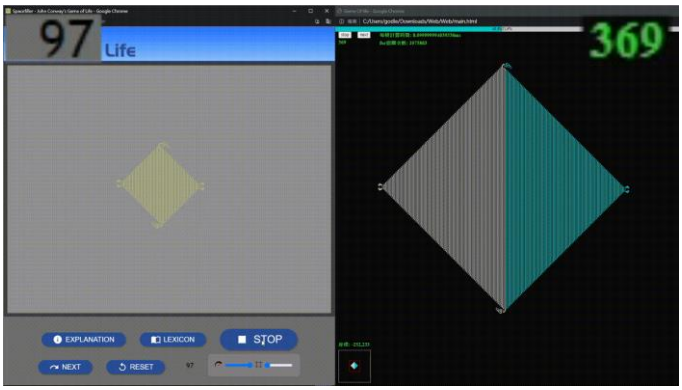
且為了讓地圖可以無限延伸，我選擇效仿 Minecraft 使用的 Chunk 遊戲機制，也就是將地圖切分，視 16x16 的細胞一組，計算時只需要計算有活細胞的 Chunk，省掉了計算整個地圖的時間，也可以做出無限大的世界，將整體效能最大化。



▲ 藍色框為單個 Chunk，白色為細胞

▶ 右圖兩個網站的效能測試，左側網站與右側我製作的網站。雙方在起初運算速度並沒有明顯落差，但在細胞的繁衍下，左側明顯較慢。

注：上方數字為總渲染幀數。利用 space filler 做為比較基準，它是在 1993 年 9 月被 Hartmut Holzwart 發現的，細胞能夠向四面八方繁殖擴散。



效能測試影片：https://rurl.page.link/gameoflife_pt

為了方便模擬與測試，我在右側新增了一些範例可以放置，像是最基本的板凳、滑翔機、高斯帕機槍，甚至是較困難的太空船發射器等。

▶ 右圖為單人遊戲畫面，可點選右邊範例，並放置在遊戲地圖中。

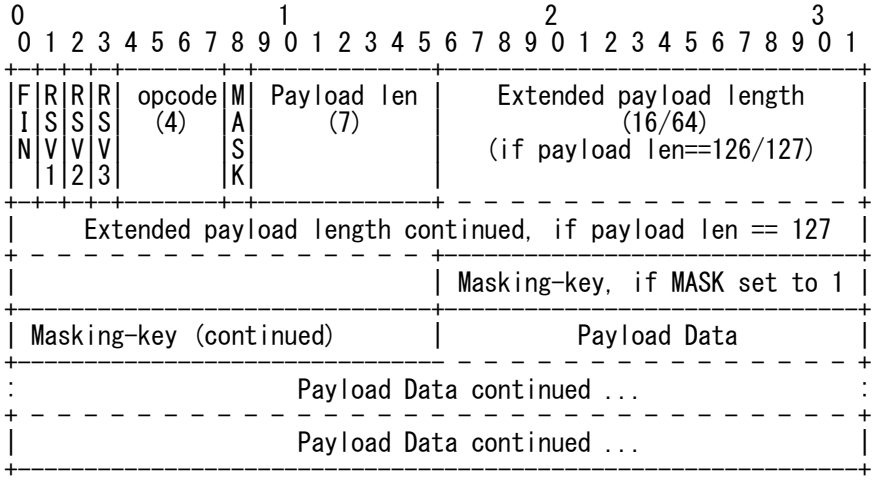
單人遊戲網址：

https://wavjaby.github.io/GameOfLife/

多人遊戲開發

完成演算法後，我開始開發連線的對戰，細胞狀態也從原始的**生**與**死**變成了**白**、**藍**與**死**，比較各隊細胞顏色站比超過一定時間與比值獲勝。伺服器使用 Java 編寫，挑戰不使用任何套件，不過這也是我第一次編寫伺服器，所以還有很多問題，像是執行順序，封包的處理與壓縮等問題，可能等之後學到了更多後可以再回來修復，做出更好的成品。資料的傳輸是使用 WebSocket，一個常用於前後端傳輸的協議，因為 JavaScript(簡稱 JS)有內建但 Java 沒有，所以我必須參考 rfc6455 WebSocketProtocol 去把協議做出來才能網頁溝通，協議就像我們人在溝通，需要規定使用相同的語言才能正常的理解雙方需要什麼，我主要是看下面這張表，他寫了封包內容的規定，就是我要怎麼傳送跟接收封包。

rfc6455#section-5.2



第 0 個 bit(**FIN**)如果是 1 代表此封包是結尾，否則接收端將會讀取封包直到 **FIN** 等於 0，接著把所有封包的 **Payload Data** 合併。

opcode 用來辨識此封包的用途，rfc 定義 0x2 是 Binary frame(用來傳送位元)，0x8 是關閉連接，其餘詳見 rfc6455#section-5.2。

Payload len 用於告訴對方此封包的大小，使用 7 個 bit 不算負數，最大可以存到 127，但是因為不是所有封包都那麼小，所以 Websock 協議定義如果 **Payload len** 設為 126 會開啟 16bit 的 **Extended payload length**，表示封包可裝下 65,535byte 的資料，約 65.53KB，如果還是不夠裝的話，可以將 **Payload len** 設為 127，將會開啟 64 個 bit 的 **Extended payload length**，最大可以裝下約 18.44EB 的資料，但通常不會傳送那麼大的資料。

如果第 8 個 bit(**MASK**)設為 1，在 **Payload len** 或 **Extended payload length** 後面會加上 4 個 Byte 的 **Masking-key** 資料，rfc6455#section-5.3 規定 Client to Server 必須要將 **MASK** 設為 1 並攜帶 **Masking-key**，供伺服器解讀。解讀的時候會將 **Payload Data** 的每個 bit 和 **Masking-key** 做 XOR 運算，就可以解讀出正確的資料。解讀程式見 ClientHandler.java#L113

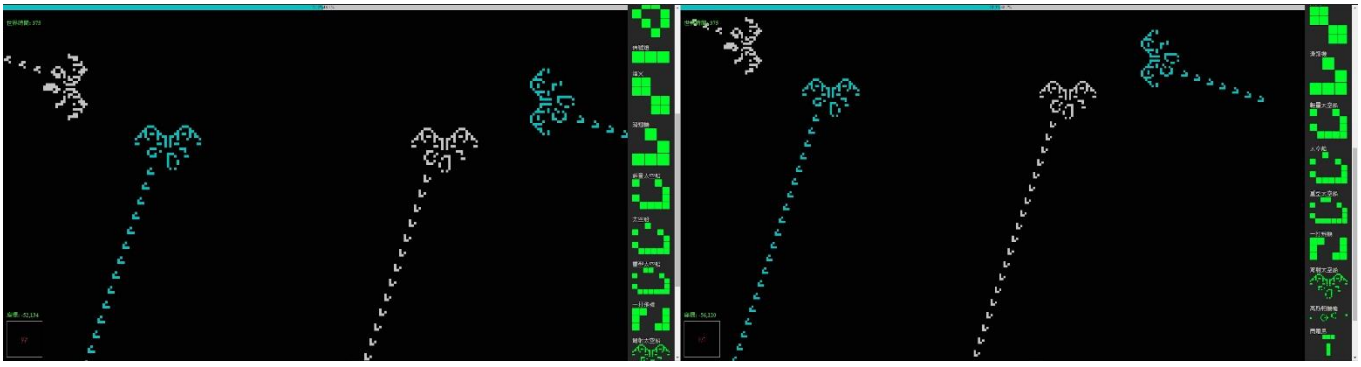
寫完協議，能夠與網頁溝通後，我開始設計多人遊戲，首先就是先把 JS 的程式轉換成 Java，因為單人遊戲是用 JS 開發，有可視化的介面在開發這種遊戲上會比較方便。

接著是多人的連接處理，我的資料傳輸設計為開頭會有一個英文字母，代表這個封包的用途，讓程式能分類處理，實例：

- connectSuccess = c　　//連接成功資料
- loginSuccess = 1　　//登入成功資料
- connectFailed = f　　//連接失敗資料
- login = l　　//連接資料
- data = d　　//一般資料
- error = e　　//錯誤訊息

設計客戶端與伺服器端的連接流程，玩家需要先登入到伺服器，客戶端會跟伺服器請求視野範圍內的 Chunk，接著伺服器會回傳 Chunk 的所有內容並記住玩家的視野。回合計算完畢，伺服器會依據每個玩家的視野，把更新的細胞、當前遊戲時間與兩隊的數量等等的數據發送到客戶端。當玩家放置細胞時客戶端會傳送要放置的細胞到伺服器，伺服器確認後會將細胞放置到下一個回合，當回合計算完畢時就會順便把放置的細胞回傳。

▼以下是多人遊戲展示



影片連結 https://rurl.page.link/gameoflife_mp

GitHub: https://github.com/WavJaby/GameOfLife