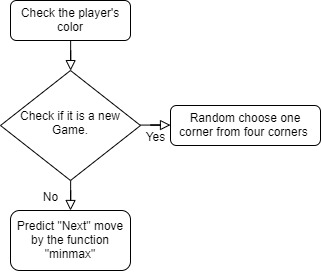
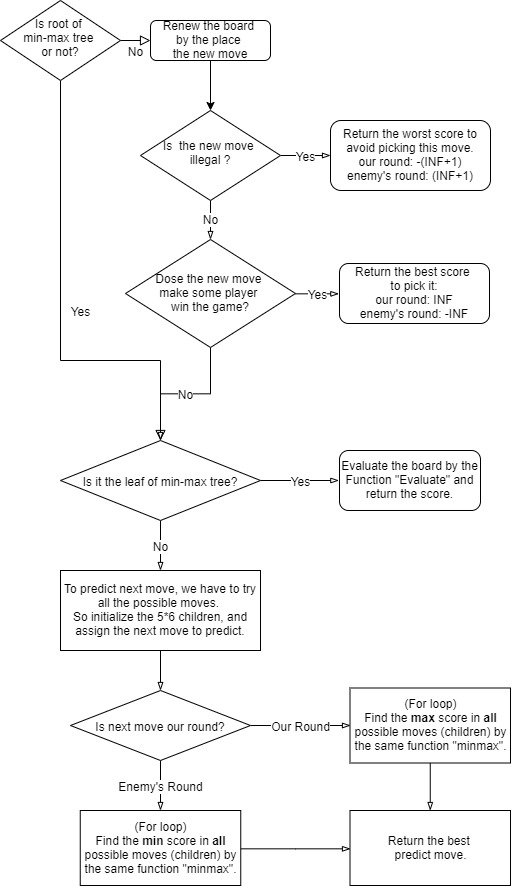
Data Structure Final Project: Chain Reaction

1. Project Description這次採用對弈常用的Min-Max Tree的Algo. 處理：
   1. Program Flow Char
      1. Main:



* + 1. “minmax” function:



* 1. Detailed Description
     1. Min-max tree:

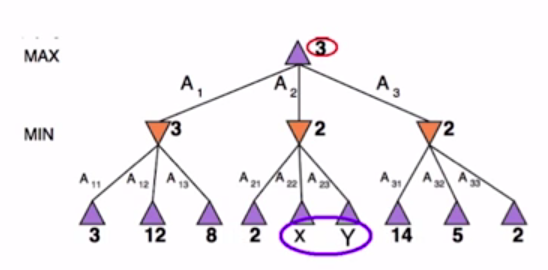
在對弈中，我方下棋要從棋局中找出最佳解，相對來說，對於對手則要找出對我方最不利的點，如果我們把每種可能的下法作為一個Node，那麼預測的棋局會變成一棵樹，且因為輪流下，我方的回合會是Max層從預測的下一層中選最大值，敵方的回合則是min層從預測的下一層中選最小值，層層相扣，大小排列，會形成Min-max tree。

Figure 1 min-Max Tree (Wikipedia)

而在這次的Chain Reaction中，我預測了後四步，如上圖的深度。

* + 1. Alpha-beta pruning:

然而，因為每一步有30種可能，預測後四步就有種可能要計算，實際上會花將近一秒的時間做運算，為了加速運算，採用Alpha-beta pruning的概念，白話來說，在下棋時，當發現有一步比較爛，已經看到是有利對方的一步，就不需要再繼續想這一步後續的可能。



以上圖為例，左子樹探索完畢，知道MAX 層至少大於3，所以中間子樹不需要探索X, Y的節點，因為MIN從中會從子樹中選擇最小的，所以中子數的MIN 勢必比2小，那X, Y不管怎麼探索，都不會比3大，因此可以直接放棄。

而在演算中，alpha 代表目前探索的最大值，beta代表目前探索的最小值，如果alpha > beta，代表這次選到的值必定比alpha小，就不需要繼續探索(pruning)。

Performance:

由上述可知，如果一開始就找到相當大的alpha值，就會發生大量的Pruning，以節省時間，相反的，可能一個都Pruning不到，就Performance上

Worst Case 仍是，Average Case則是

然而，即便如此，加速幅度仍不夠大，計算時間雖然快一點，卻沒辦法讓我在1秒之內再往下探索到第五步之後。

如果需要改進，可能在第二層就需要先做Pruning，提早撇除很差的可能再繼續探索。

* + 1. Evaluation:

1. Screen Shots
   1. Partial Implemented Code
   2. GitHub Control History
   3. Compare with TA’s AI Code with Student Id