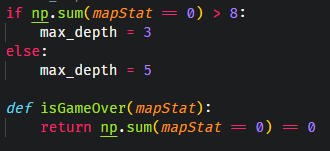
Project 2

109550087 單宇晟

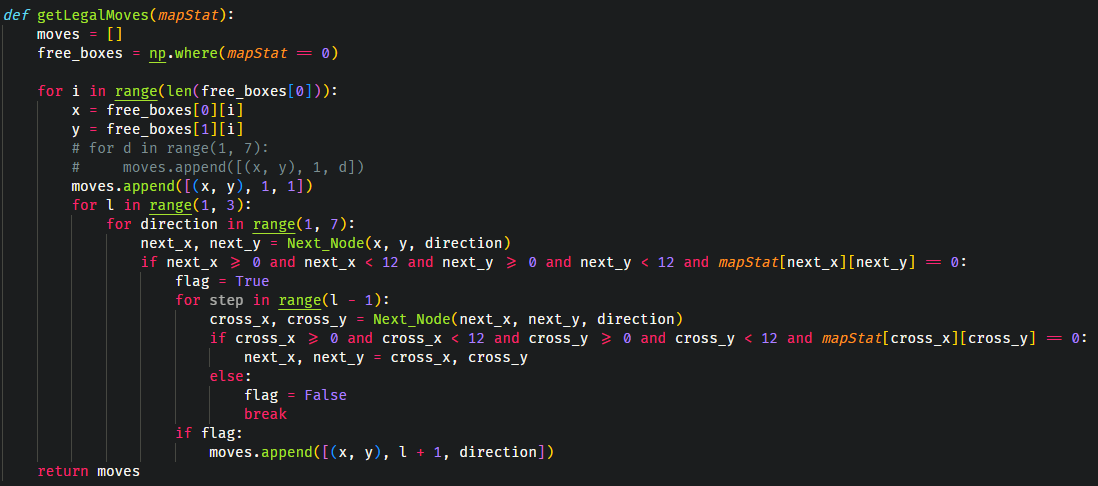
在這次的project中，我選擇用minimax來當作我主要的演算法，主要有以下幾點原因：

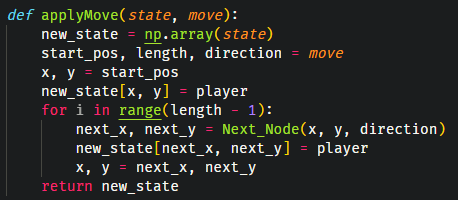
1. minimax就是根據two-player的遊戲來設計，加上本次遊戲的outcome又十分明確，讓minimax可以正確地做出optimal choice。
2. minimax相對於其他演算法來較為簡單，比較可以根據自己的想法做出調整(像是evaluation function)
3. 雖然本次遊戲的可能性很多，但只要加上alpha-beta pruning，minimax就能更有效率的搜尋整個game tree，做出更好的決策

**程式結構：**



首先，我的max\_depth 會根據剩下的格子數改變，因為越接近結尾的決策就越重要，因此我會在剩下8格的時候將max\_depth提升到5；後面則是簡單判斷遊戲結束與否的function。





接著是我的getLegalMove() 和 applyMove()，顧名思義，getLegalMove() 先找到所有合法的路徑，再由applyMove() 去更新map state。



再來就是我的minimax本身了，不過我並沒有特別更改某個步驟，主要就是從所有可能的路徑中，盡可能地選出最好的路徑。



最後，是我認為除了演算法本身的選擇之外，這次project影響performance最大的部分 — evaluation function。由於minimax會需要兩個evaluation function，一個負責找min-value、另一個則負責max-value，而我的寫法中，兩者只差了一個負號，所以這裡就只以max-value舉例。最基本的，當然就是判斷勝負，如果只剩1個格子是空的，就代表我贏；如果沒有剩，代表我走到最後一步，也就是我輸了，所以根據這兩個情況分別給reward的值(1000、-1000)。接著是格子剩5個以下的狀況，如果我走完之後的格子剛好剩五格，那不管格子的分布、連接狀況如何，有很高的機率會是我贏(這部分是我自己窮舉的)，雖然有時候還是可能會敗北，但對我來說剩五格幾乎等同於獲勝，所以一樣return 1000；而如果剩不到五格，我自己是假設這個情況下，每個人每次都只走一格，那剩下奇數個格子的話，就也會是我贏(對手-我-對手…)，反之則會是我輸，所以分別return 1000、-1000。

**總結**

這次project我花的時間主要都是在想evaluation function上，在一開始主要架構的部分寫好後，只要我有新想法想要嘗試，我就讓我寫的程式自己跟自己打(舊版本vs新版本)，確保我在evaluation function中新加上去的code有助於程式做出正確的判斷，如果錯了就再想新的評估方式，如此往復，才變成我現在的evaluation function。