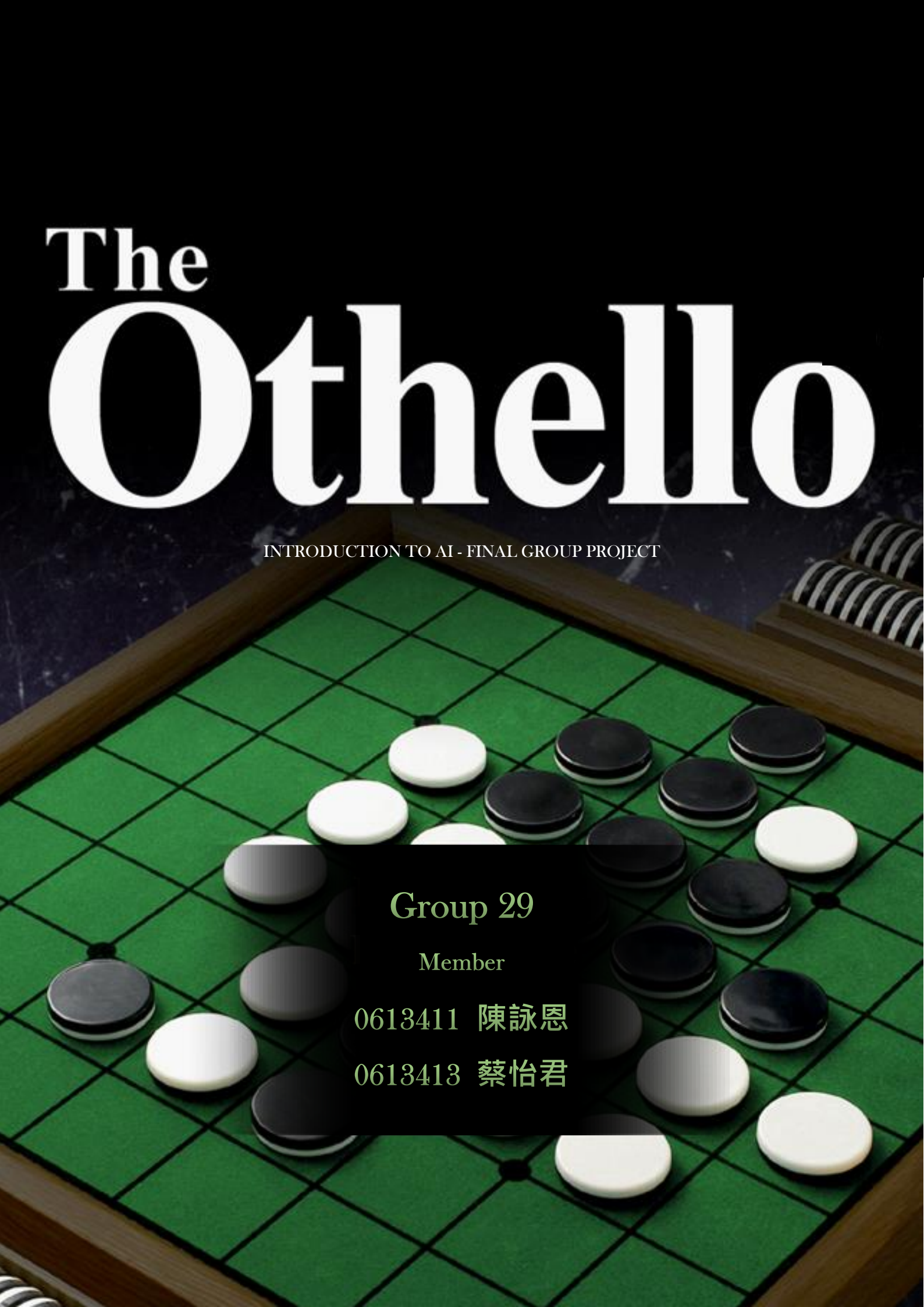
The Othello

INTRODUCTION TO AI - FINAL GROUP PROJECT



Group 29

Member

0613411 陳詠恩

0613413 蔡怡君

研究方法

利用 Monte Carlo tree search ，加上 Policy 及 UCT（Upper Confidence Bounds to Trees）公式，設計一個 game-playing agent，並比較參數及 policy 以獲得最佳勝率。

一、 程式設計

1. Class & function

Class name	variable	function
Position	int x; int y;	Getters and Setters

Board	int boardValues[8][8];
	int totalMoves;
		Getters and Setters
		void Print board()
		void performMove()
		int checkStatus()//檢查遊戲是否結束並算分
		vector<Position> getPossiblePositions()
		//尋找現在可以下的位置

State	Board board;
	int playerNo;// 1 or 2
	Position move;// 動了哪一步才到目前的狀態
		Getters and Setters
		int getStateOpponent()
		vector<State> getAllPossibleState()
		State randomPlay()//for simulation step

Node	State state; Node* parent; vector<Node> childArray; int visitCount; double winScore; bool root;//is root node	Getters and Setters Node* getRandomChildNode() double getChildScore(int i) Node* getChildWithMaxScore()
Tree	Node root;	Getters and Setters
UCT		static double uctValue(int parentVisit, double nodeWinScore, int nodeVisit) static Node* findBestNodeWithUCT(Node* node,int level)

MonteCarlo TreeSearch	double WIN_SCORE;
	//贏了一次加幾分
	int opponent;//對手
	double t;//time count
	int level;
	//目前棋盤上的剩餘空格
		Node* selectPromisingNode(Node* rootNode,int level)
		void expandNode(Node* node)
		int simulateRandomPlayout(Node* node)
		void backPropogation(Node* nodeToExplore, bool playerwin)
		Position findNextMove(Board board, int playerNo,int levels)

2. UCT

- wi = number of wins after the i-th move
- ni = number of simulations after the i-th move
- c = exploration parameter
- t = total number of simulations for the parent node

$$\frac{w_i}{n_i} + c\sqrt{\frac{\ln t}{n_i}}$$

3. Policy

(1) Action probabilities weight

```
double probability[8][8] = {0   ,1.20,1   ,1.1 ,1.1 ,1   ,1.20,0   ,
                            1.20,0.80,0.95,0.95,0.95,0.95,0.80,1.15,
                            1   ,0.95,1   ,1   ,1   ,1   ,0.95,1   ,
                            1.1 ,0.95,1   ,1   ,1   ,1   ,0.95,1   ,
                            1.1 ,0.95,1   ,1   ,1   ,1   ,0.95,1   ,
                            1   ,0.95,1   ,1   ,1   ,1   ,0.95,1   ,
                            1.20,0.80,0.95,0.95,0.95,0.95,0.80,1.15,
                            0   ,1.20,1   ,1.1 ,1.1 ,1   ,1.20,0   };
```

(2) Heuristic function(conditional probabilities)

二、 實驗方法

(由最陽春的 MCTS Agent 與加上 Policy 及更改 UCT 係數後的 Agent 作對手)

1. 操縱變因

(1) UCT 的係數：更改 exploration parameter 觀察結果(預設為√2)

(2) Policy：

2. 應變變因

(1) 勝率

結果分析

一、 UCT 的 exploration parameter

C=	1.2	1.41	1.6
勝率	0.6	0.3	0.75

一開始都使用理論值 1.41(√2)來跑，發現結果不佳，反而在 1.6 及 1.4 表現更好，後來近一步嘗試 1.7 但勝率又明顯下降了，因此決定使用 1.6 作為之後調正 policy 的基礎設定。

二、 Probability 權重的使用

probabilities	Filter1	Filter2	Filter3
勝率	0.55	0.7	0.4

(filter 1)

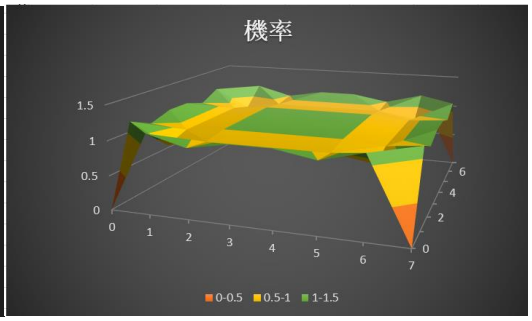
```
{ 0    ,1.10,1    ,1.05 ,1.05 ,1    ,1.10,0    ,
  1.10,0.85,0.95,0.95,0.95,0.95,0.8,1.10,
  1    ,0.95,1    ,1    ,1    ,1    ,0.95,1    ,
  1.05 ,0.95,1    ,1    ,1    ,1    ,0.95,1.05
  1.05 ,0.95,1    ,1    ,1    ,1    ,0.95,1.05
  1    ,0.95,1    ,1    ,1    ,1    ,0.95,1    ,
  1.10,0.85,0.95,0.95,0.95,0.95,0.85,1.10,
  0    ,1.10,1    ,1.05 ,1.05 ,1    ,1.10,0 };
```

(filter2)

```
= {0    ,1.20,1    ,1.1 ,1.1 ,1    ,1.20,0    ,
  1.20,0.80,0.95,0.95,0.95,0.95,0.80,1.15,
  1    ,0.95,1    ,1    ,1    ,1    ,0.95,1    ,
  1.1 ,0.95,1    ,1    ,1    ,1    ,0.95,1    ,
  1.1 ,0.95,1    ,1    ,1    ,1    ,0.95,1    ,
  1    ,0.95,1    ,1    ,1    ,1    ,0.95,1    ,
  1.20,0.80,0.95,0.95,0.95,0.95,0.80,1.15,
  0    ,1.20,1    ,1.1 ,1.1 ,1    ,1.20,0    };
```

(filter3)

```
= { 0    ,1.2,1    ,1.1 ,1.1 ,1    ,1.2,0    ,
    1.2,0.8,0.9,0.9,0.9,0.9,0.8,1.2,
    1    ,0.9,1    ,1    ,1    ,1    ,0.9,1    ,
    1.1 ,0.9,1    ,1    ,1    ,1    ,0.9,1    ,
    1.1 ,0.9,1    ,1    ,1    ,1    ,0.9,1    ,
    1    ,0.9,1    ,1    ,1    ,1    ,0.9,1    ,
    1.2,0.8,0.9,0.9,0.9,0.9,0.8,1.2,
    0    ,1.2,1    ,1.1 ,1.1 ,1    ,1.2,0 };
```



機率分布圖

雖然沒有像 AlphaGo 一樣使用 NN，但是我們仿造他，加上 action probabilities，起初是將每次計算 UCT value 時都將分數再乘上機率權重，但是發現到遊戲快結束時，這樣的權重反而影響了整個決定的平衡(起初比較有影響的位置已經改變不大了，且幾乎可以跑完整個

Search Tree)，因此在遊戲結束前五步(仍有五部空格時)，我們將 action probabilities 的加權取消，勝率也有提高。

依照上面的機率模型，我們改變權重的敏感度(差距大小)來測試結果，其中第二個的勝率最佳，第三個最差，我們認為是因味權重差距大，造成 agent 會很執著要先填哪些位置，導致去許多機會。

三、 Heuristic function

因為每個階段(state)，對於每一個位子被下的機率會有所改變，因此我們決定將 probabilities array 寫入 function，例如：當(1,0)已經放入自己的棋子時，(2,0)的權重就變高(因為不可能被翻轉)，以及當(1,0)和(0,1)都放了自己的棋子時(1,1)的位置並沒有原本差等條件。

測試完的結果發現並沒有明顯的優勢，可能是因為他的對手(最陽春的 MCTS)，並沒有用 Filter 因此在 simulation 的時候比較不能預測到對手的情況。

討論

一、 結果探討

根據上面的實驗，最好的結果就是能在 UCT 的係數為 1.6 且加上 Filter 2。再測試的過程中，嘗試多加一些自己的想法，似乎不見的對整個 agent 的能力有提升，因為改變規則的關係，整個遊戲又多了更多可能性，包含可以佈局或是先占位子等等，可能需要更好的遊戲經驗，或是只能靠 NN 來優化他了。

二、 未來展望

對原本的下棋規則來說，下棋的方式，不是取決於這次能夠吃多少旗子，而是下完了這次能夠爭取到更多可下的步數(行動力)，但如今的下棋規則改了，6*6 裡頭的都是可下步數，而在實驗方面，還是可以去試試看是否增加可下步數（ 在角落、邊上 ），同時在 simulation 的時候都以能夠增加自己的可下步數、阻擋對手的可下步數。

- 主要採取控制中心策略：上網查到減少對手行動力（尤其是没有大大減少自己行動力）的方法是避免翻轉過多的的邊緣子。擁有內部子（即不與空位相鄰的棋子）比擁有邊緣子更好，在最後剩下五步直接使用 MIN-MAX，因為時間足夠跑完所有的可能性，可以高度提升最後五步的獲勝率。