Hw2

本次作業是將 UCB、UCT 以及 Progressive pruning 等實作在於九路圍棋上。並且以此產生的 AI 要能夠以高勝率贏過以隨機方式決定的 random AI。

作業中要求的 Principal variation 的 output 會輸出在 Principal_variation.txt 中。此檔案會在一局遊戲開始時刷新，並且記錄一局遊戲己方的出手。

Makefile command:

g++ -std=c++11 new_hw2.cpp -O3 -o execute_filename

UCB

此次作業最開始為實作 UCB，我將其實作在 MCS_pure_move 中，包含下列幾個部分：

1. Selection

每一步中，我們會針對所有可能的下一步，選取分數高的（在此即為勝率高者抑或是 UCB 分數高者）做接下來 Expansion 的動作。

2. Expansion

以每個盤面為 root，我們需要產生所有可能的下一步，這部分原本助教給的 code 裡面就已實作了 gen_legal_move() function，只要去以現在的盤面、出手權以及遊戲進行的步數，我們就可以得到目前的情況下，下一步可能的所有的合法步，並將其存於 MoveList 中。

3. Simulation

在得到所有可能的下一步後，我會將每一個可能的下一步，對其做 random simulate，也就是以隨機的方式將該盤面下完，並且將最後的結果存起來。

其中我們需要自己去實作 random simulate 的 function，此次作業中我將其 implement 在 rand_simulate() 中，透過原本提供的 function rand_pick_move 我使每個盤面以隨機的方式走到盤末並且記錄勝利者並回傳。

4. 選勝率最高者

最後選擇要走的 move 是以勝率最高者做選擇。

UCT

我將其實做在 MCTS_move 中。

UCT 和 UCB 最大的不同在於，他會將所有目前已探索過的點記錄下來，並且生成一顆 Tree，在每次 simulation 過後，會多一步叫做 back propagation，目標是為了將 visit 和 score 做回傳，以利於計算 UCB 的分數。為了使實作更容易，我這次定義的 node 結構為：

```
typedef struct mcsNode{
    struct mcsNode* parent;
    vector <struct mcsNode> child;
    double score;
    double ucb_value;
    double square_score;
    int visit;
    int move;
    int turn;
    int Board[BOUNDARYSIZE][BOUNDARYSIZE];
}MCSNODE;
```

其中，每個 node 由於需要做 back propagation 所以需要記錄自己的 parent，另外做完 expansion 後，會將自己所有的 child 也都記錄在 child 的 vector 中。另外，square_score 是存此盤面得分的平方，目的是為了使在實作 progressive pruning 時更方便。

計算 UCB 的方式為使用公式計算，在實作中我將它實作在 getUCB 這個 function 中。公式如下，child.wins / child.visits + c * sqrt(log(N) /child.visits)

所以，UCT 中的主要步驟如下

1. Selection

每一步中，我們會針對所有可能的下一步，選取分數高的（在此即是 UCB 分數高者）做接下來 Expansion 的動作。

2. Expansion

以每個盤面為 root，我們需要產生所有可能的下一步，利用 gen_legal_move 所得到的 MoveList，我們可以將這個下一步產生的盤面，分數等等資訊都存進一個新的 child_node 中，並且 push_back 進 current 的 child vector 中。

3. Simulation

在得到所有可能的下一步後，我會將每一個可能的下一步，對其做 random simulate，也就是以隨機的方式將該盤面下完，並且將最後的結果存起來。其中與 UCB 不同，我實作的另一個 function random_simulate_score，他可以用隨機的方式產生這個盤面繼續下下去之後最後的黑白子勝分，其中需要注意的是因為勝分皆以黑方的角度呈現，所以白方的勝分會表示為負數，做 back propagation 需要額外小心。

4. Back Propagation

每次從 root 以 random_simulate_score 得到的分數，我們需要將其 parent 的 visit 以及 score 去做更新。

5. 選擇分數最大者

這裡的分數我們是選擇 score / visit 後的最大者，他所代表的 move 做回傳。

其他優化方式

剪枝優化

此次作業中有要求要使用 progressive pruning 做優化，因此在每次計算一個節點的 score 時，我便會存另一項數據，也就是 score 的平方，來方便之後做 progressive pruning 時的計算。progressive pruning 中有兩樣重要的常數可以做調整，也就是 gamar_d 與 sigma_e，這些參數我去參考了講義中所提到的文獻，並且多次的嘗試不一樣的數字。

UCB 參數調整

UCB 計算方式中有一樣常數 C 可以去做調整，我試過的參數有此常數的理論值sqrt(2) 也就是 1.414、2、3 和 4，但是實際去讓不同的常數的 AI 去做比較後，得到的結果發現常數為理論值，1.414 時勝率最高。附上，他們之間對戰的數據

2 vs 1.414 (2為白)

#GAME	RES_B	RES_W	RES_R	ALT	DUP	LEN	TIME_B	TIME_W	CPU_B	CPU_W	ERR	ERR_MSG
0	0	0	?	0	-	89	276.5	254.8	0	0	0	
1	B+50	B+50	?	1	-	120	289.7	339.3	0	0	0	
2	W+6	W+6	?	0	-	102	277.1	281.7	0	0	0	
3	B+6	B+6	?	1	-	116	299	312.8	0	0	0	
4	B+22	B+22	?	0	-	114	288.6	337.1	0	0	0	
5	B+88	B+88	?	1	-	132	304.7	432.3	0	0	0	
6	B+50	B+50	?	0	-	107	246.5	375.4	0	0	0	
7	B+36	B+36	?	1	-	109	188.2	286	0	0	0	
8	W+34	W+34	?	0	-	141	371.2	333.9	0	0	0	
9	B+20	B+20	?	1	-	90	243	278.4	0	0	0	

3 vs 1.414 (3為白)

#GAME	RES_B	RES_W	RES_R	ALT	DUP	LEN	TIME_B	TIME_W	CPU_B	CPU_W	ERR	ERR_MSG
0	B+6	B+6	?	0	-	109	317	301.6	0	0	0	
1	W+12	W+12	?	1	-	105	291.9	283.6	0	0	0	
2	W+8	W+8	?	0	-	110	265.9	293.2	0	0	0	
3	B+14	B+14	?	1	-	101	292.9	301.9	0	0	0	
4	B+20	B+20	?	0	-	116	254.8	358.2	0	0	0	
5	B+14	B+14	?	1	-	113	291.9	324.6	0	0	0	
6	W+30	W+30	?	0	-	90	272.1	244.1	0	0	0	
7	B+26	B+26	?	1	-	102	306.3	303.6	0	0	0	
8	W+20	W+20	?	0	-	96	283.3	277	0	0	0	
9	W+40	W+40	?	1	-	107	335.2	270.5	0	0	0	

4 vs 1.414 (4為白)

#GAME	RES_B	RES_W	RES_R	ALT	DUP	LEN	TIME_B	TIME_W	CPU_B	CPU_W	ERR	ERR_MSG
0	W+22	W+22	?	0	-	110	260.2	219.6	0	0	0	
1	B+8	B+8	?	1	-	90	275.2	253.6	0	0	0	
2	W+88	W+88	?	0	-	150	407.3	307.4	0	0	0	
3	B+20	B+20	?	1	-	90	270.7	283.2	0	0	0	
4	W+30	W+30	?	0	-	109	303.5	283.6	0	0	0	
5	B+10	B+10	?	1	-	92	257.2	277.6	0	0	0	
6	B+16	B+16	?	0	-	96	235.5	306.6	0	0	0	
7	B+6	B+6	?	1	-	152	339.3	354.9	0	0	0	
8	W+18	W+18	?	0	-	102	310.6	262.7	0	0	0	
9	B+26	B+26	?	1	-	100	285.7	293	0	0	0	

加入領域知識

這次作業中，我使 AI 的前五步中，不會去下最外層那一圈的棋盤，並且由於觀察到 AI 有很多時候因為不去提子而輸掉一場比賽，因此，我在 MCT 中做 Expansion 時多加入了一些判斷，若下一步能夠產生提子的結果，我便會使該步有更高的機率被選中。

結果

和隨機對戰的結果，我方執黑

#GAME	RES_B	RES_W	RES_R	ALT	DUP	LEN	TIME_B	TIME_W	CPU_B	CPU_W	ERR	ERR_MSG
0	B+22	B+22	?	0	-	91	229.4	0	0	0	0	
1	B+28	B+28	?	1	-	107	284.7	0	0	0	0	
2	B+42	B+42	?	0	-	97	252	0	0	0	0	
3	B+58	B+58	?	1	-	108	246.9	0	0	0	0	
4	B+74	B+74	?	0	-	131	309.7	0	0	0	0	
5	B+46	B+46	?	1	-	101	259.8	0	0	0	0	
6	B+38	B+38	?	0	-	99	244.3	0	0	0	0	
7	B+32	B+32	?	1	-	107	269	0	0	0	0	
8	B+74	B+74	?	0	-	125	302.7	0	0	0	0	
9	B+24	B+24	?	1	-	91	257.2	0	0	0	0	

#GAME	RES_B	RES_W	RES_R	ALT	DUP	LEN	TIME_B	TIME_W	CPU_B	CPU_W	ERR	ERR_MSG
0	B+36	B+36	?	0	-	99	330.1	0	0	0	0	
1	B+42	B+42	?	1	-	101	340.9	0	0	0	0	
2	B+42	B+42	?	0	-	104	357	0	0	0	0	
3	B+28	B+28	?	1	-	102	341.2	0	0	0	0	
4	B+32	B+32	?	0	-	102	307.6	0	0	0	0	
5	B+32	B+32	?	1	-	101	269.9	0	0	0	0	
6	B+42	B+42	?	0	-	116	378.8	0	0	0	0	
7	B+48	B+48	?	1	-	98	293	0	0	0	0	
8	B+74	B+74	?	0	-	139	396	0	0	0	0	
9	B+30	B+30	?	1	-	103	347.7	0	0	0	0	

#GAME	RES_B	RES_W	RES_R	ALT	DUP	LEN	TIME_B	TIME_W	CPU_B	CPU_W	ERR	ERR_MSG
0	B+32	B+32	?	0	-	95	314.4	0	0	0	0	
1	B+44	B+44	?	1	-	99	276	0	0	0	0	
2	0	0	?	0	-	99	320.2	0	0	0	0	
3	B+62	B+62	?	1	-	100	313.9	0	0	0	0	
4	B+74	B+74	?	0	-	117	279.3	0	0	0	0	
5	B+54	B+54	?	1	-	110	333.9	0	0	0	0	
6	B+38	B+38	?	0	-	103	366.9	0	0	0	0	
7	B+36	B+36	?	1	-	109	353.6	0	0	0	0	
8	B+30	B+30	?	0	-	95	312.1	0	0	0	0	
9	B+40	B+40	?	1	-	96	287.6	0	0	0	0	