Foundations of Artificial Intelligence

Final Project Report

B09902062

陳晏霆

Methods I've Tried:

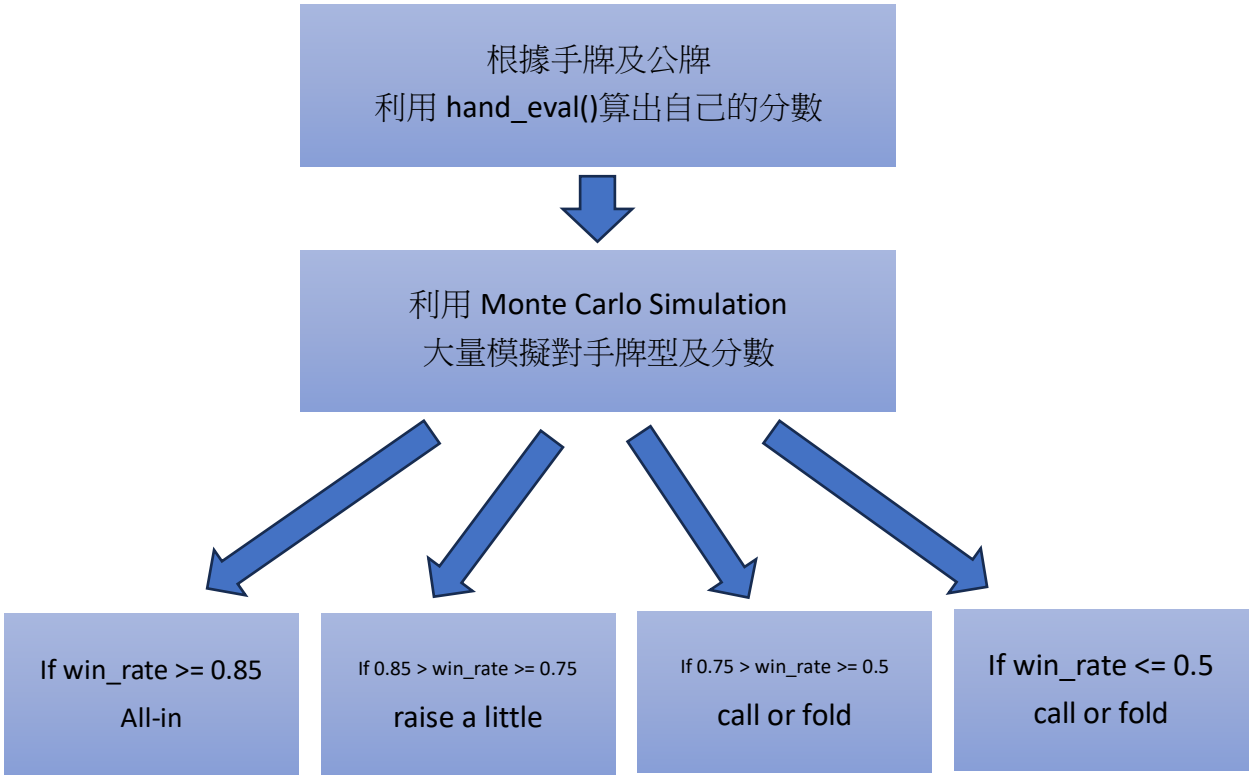
1. 使用 hand_eval()計算出自己的手牌與公牌所組成的最大分數，再依照此分數決定是否 raise, call, 或是 fold
   - ➔ **Result**: 效果不佳，在各個 baseline 上勝率都不超過五成。其中又以 baseline 3~5 為最差，勝率皆不及一成。
   - ➔ **Disadvantage**：只考慮到手牌及公牌的絕對大小，而沒有考慮到對手所有可能之手牌組合，造成容易在自己手牌分數不夠大時因為太保守而 fold，導致無法獲勝。
   - ➔ **Modification**: 在使用 hand_eval()時加上 Monte Carlo Simulation，計算出對手可能的手牌牌型及分數大小，經過多次模擬後依結果計算出勝場數所佔的比例，視為勝率(win rate)。

2. Based On Monte Carlo Simulations

   (Reference: https://www.data-blogger.com/pokerbot-create-your-poker-ai-bot-in-python/)

   每一輪都將場上的公牌及自身手牌透過 eval_hand()與對方所有有可能手牌分數進行多次模擬比較，計算出 win rate，再針對當下的 win rate 採取不同的對應措施。

➔ **Process**:



➔ **Adjustable Hyperparameters**:

- **nb_simulation (# of simulations)**

	100	500	1000	10000
baseline_0	85%(17/20)	90%(18/20)	90%(9/10)	exceed time limit
baseline_1	95%(19/20)	80%(16/20)	90%(9/10)	
baseline_2	60%(6/10)	70%(7/10)	70%(7/10)	
baseline_3	80%(8/10)	100%(10/10)	90%(9/10)	
baseline_4	10%(1/10)	20%(2/10)	20%(2/10)	
baseline_5	10%(1/10)	40%(4/10)	30%(3/10)	

◆ **Result**: 在 nb_simulation = 100 時，勝率仍未達到最佳值；nb_simulation = 500 及 1000 時有最佳勝率，但 1000 程式的執行速度明顯較慢：nb_simulation = 10000 時則超過時間上限。故以下的實驗採用 **nb_simulation = 500**。

- **Action when 0.75 > win_rate >= 0.5, call or fold?**

	call every time	fold every time	call when call_amount <= 100	call when call_amount <= 300	call when call_amount <= 50
baseline_0	60%(6/10)	0%(0/10)	90%(9/10)	90%(9/10)	90%(9/10)
baseline_1	40%(4/10)	0%(0/10)	80%(8/10)	50%(5/10)	80%(8/10)
baseline_2	60%(6/10)	10%(1/10)	70%(7/10)	80%(8/10)	60%(6/10)
baseline_3	50%(5/10)	0%(0/10)	100%(10/10)	100%(10/10)	80%(8/10)
baseline_4	40%(4/10)	0%(0/10)	40%(4/10)	60%(6/10)	10%(1/10)
baseline_5	0%(0/10)	0%(0/10)	40%(4/10)	20%(2/10)	30%(3/10)

  - **Result**: call every time 的情況下，會因為在勝率不夠高的情況下還一直盲目的跟注就輸錢；而 fold every time 的情況下，則會因為太保守、太早就 fold，所以一直輸小錢；後三者皆是在一定金額內 call，超過就 fold，當門檻為 50 或 100 時有最好的結果，其中又以 100 更佳；而當門檻為 300 時，部分 baseline 下的勝率有所下滑。故以下實驗採用 **call when call_amount <= 100, else fold.**

- **Action when win_rate < 0.5, call or fold?**

	fold every time	call if can_call	call if can_call and call_amount = 0
baseline_0	90%(9/10)	100%(10/10)	80%(8/10)
baseline_1	80%(8/10)	60%(6/10)	80%(8/10)
baseline_2	70%(7/10)	70%(7/10)	70%(7/10)
baseline_3	100%(10/10)	30%(3/10)	80%(8/10)

baseline_4	40%(4/10)	10%(1/10)	40%(4/10)
baseline_5	40%(4/10)	60%(6/10)	40%(4/10)

- Result: 當勝率低於五成時，直接 fold 的效果勝過小額 call 及無條件 call。故以下實驗採用 **fold every time**。

Final Method:

Method 方面，使用 Monte Carlo Simulations，將自己手牌與公牌之組合分數，與對手所有可能之分數做模擬比較，進而利用計算出的勝率決定下一步的 action。在參數方面，經過實驗結果之分析，採用：

```
If win_rate >= 0.85:
    All-in
else if 0.85 > win_rate >= 0.75:
    Raise by the minimum amount possible
else if 0.75 >= win_rate > 0.5:
    If call_amount <= 100:
        Call
    else:
        Fold
 else:   # win_rate < 0.5
    Fold
```

Discussion & Conclusion:

幾個實驗中遇到的問題，及未能解決的困難：

1. 無法透過 Monte Carlo Simulation 得知對方是否真的有很好的手牌，還是單純的在 bluff。
2. win rate 的分層，目前是以原本的 0.85, 0.75, 0.5 作為界線，但在嘗試調整這些數值後，勝率並沒有明顯規律的改變，故仍然採用原先數值。
3. 在 baseline 4 及 baseline 5 上，勝率仍然無法突破五成。