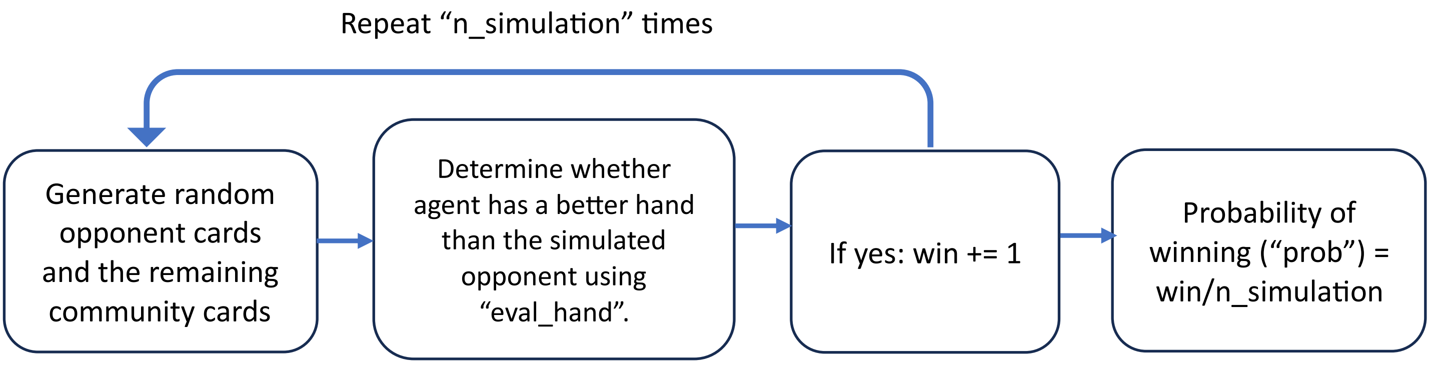
B09902085 – 郭雅美
Foundation of AI – Final Project Report

The method I decided to go for is Monte Carlo Simulations.

- When "declared_action" function is called, the agent will simulate "n_simulation" games. (will discuss more about the parameters later on)
- For every simulation game, it will generate random opponent cards and the remaining community cards. Eg: If the current round is flop, it will generate another 2 cards (since in flop, it only has 3 community cards, so it will generate the remaining 5 – 3 = 2 community cards).
- Note: there won't ever be repeated cards.
- Using the "eval_hand" function from "HandEvaluator" class that is available through game/engine/handevaluator.py, the agent can determine whether it has a better card than the simulated opponent (reminder: the opponent card is randomly generated). If yes, increase the "win" counter by one.
- After all the simulations are done, calculate the probability of the agent winning: "win/n_simulation"

Flowchart:



Note: the word opponent starting from this point on doesn't refer to the randomly generated one, but the actual opponent.

Configuration
The n_simulation that I decided to go for is 1000.
I tested various n_simulation, from 1000 to 3000 to 5000, but when I tested with baseline 3 and 4, the result does not really change. Therefore, I decided to go for 1000 to have a faster running time.

Once the agent obtained the probability of winning (or "prob" for short), the agent will need to decide whether to fold, call or raise.

The decision making is divided into three sections.

➔ Section 1: Keep folding
If the agent has enough stack to keep on folding for the rest of the rounds until the game is over and still has more stack than the opponent, the agent will keep on folding. This way the agent is guaranteed to win without any risk.

➔ Section 2: Reducing confidence:

- During the preflop street, the agent will determine whether the hole card it has is a good hole card or not.
  If it is a pair, or the highest rank is jack or above, or the difference between the hole cards' rank is less than 5, it is considered to be a good hole card.
  If the hole cards are not good and the opponent is doing raise, then the "prob "will be halved to reduce its confidence.
- For the remaining streets, if the "prob" < 0.8, the opponent is raising above 3% of agent's stack and the hole cards are not good, reduce "prob" by halved.
- The reason why I implemented this is because I observed that when baseline 4/5 raised, it is most likely to have a very good card. Therefore, I reduced the agent confidence so that it won't raise and won't call if the opponents raise with a huge sum of chips.

➔ Section 3: Call, fold or raise

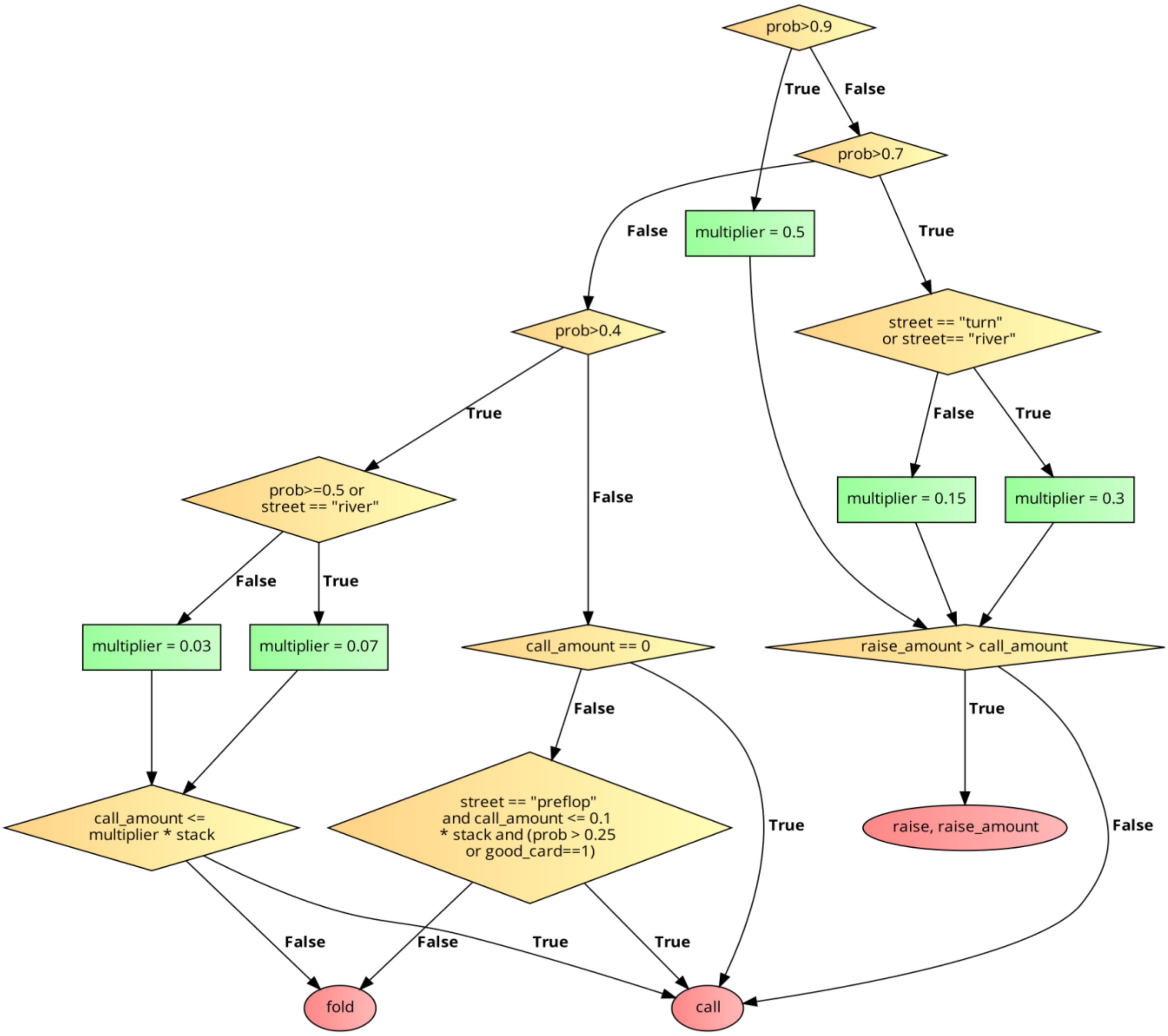
- The current value of "prob" will now determine if the agent will call, fold or raise.
- In short, if the agent has "prob" > 0.7, the agent will either raise or call. If "prob" <= 0.7, the agent can either call or fold.
- The raise amount is determined by the following calculation:

$$raise\_amount = prob * multiplier * \frac{maximum\ amount\ of\ raise}{2}$$

  The multiplier also depends on the "prob".
- I played around with the threshold and the multiplier until I am able to get a consistent 60% winning rate from baseline 1 – baseline 3.
- The agent will always call if the call amount is 0.
- In short, the higher the value of "prob", the more likely it will raise. As the "prob" got lower, it will call, then once it reaches to the point of "prob" < 0.4, it will fold if the call amount is not 0. Furthermore, if the street is "river" or "turn", it will more likely to have a higher multiplier which increases the agent likely chance to call when the opponent is doing raise.

- The <u>flowchart</u> on the next page gives the complete operation behind section 3.

Note: stack here refers to the initial stack from the beginning of the round, and good_card refers to whether it has a good hole card, which is determined from section 2.
(raise_amount is determined using the formula in section 3)



Conclusion:
Note: Results are on the next page

The agent is able to score consistently for baseline 1 to baseline 3. However, the agent is unable to beat baseline 4 and 5 (cannot get > 60% winning rate) no matter how much I changed the decision making. The limitation behind Monte Carlo Simulation is that it follows a strict algorithm in its decision, and lacks the ability to improve through learning.
A more effective approach for this homework would be to use reinforcement learning with a neural network. This can include a variant of Q-learning like deep Q-learning which is suitable for handling discrete dimension of actions.

Below is the result when I tried running 10x for each baseline.

Baseline 1 – 10/10

```
game: 1
baseline: 853.09375
player: 1145.9375
game: 2
baseline: 774.9761390000001
player: 1224.476139
game: 3
baseline: 769.9474
player: 1229.465112594
game: 4
baseline: 720.120479
player: 1279.1204790000002
game: 5
baseline: 982.345238785
player: 1016.7564165666968
game: 6
baseline: 889.8078488461326
player: 1109.8078488461326
game: 7
baseline: 982.2256
player: 1017.2256
game: 8
baseline: 992.14513118125
player: 1007.2511105855524
game: 9
baseline: 834.8028928606536
player: 1162.5515309544037
game: 10
baseline: 635.3050667553954
player: 1363.6121667553953
you win 10 games
```

baseline 2 – 10/10

```
game: 1
baseline: 526.9974455375
player: 1472.9974455375
game: 2
baseline: 693.18936050095
player: 1305.89586050095
game: 3
baseline: 718.3139500000001
player: 1281.3139500000002
game: 4
baseline: 857.595538523
player: 1141.595538523
game: 5
baseline: 741.4805852468583
player: 1256.2590852468581
game: 6
baseline: 856.4443214033109
player: 1143.4443214033108
game: 7
baseline: 834.19902889200
player: 1165.1990288920051
game: 8
baseline: 887.5
player: 1112.5
game: 9
baseline: 836.6772451605846
player: 1162.6772451605846
game: 10
baseline: 959.7116200465
player: 1039.7366963752006
you win 10 games
```

baseline 3 – 10/10

```
game: 1
baseline: 910
player: 1090
game: 2
baseline: 915
player: 1085
game: 3
baseline: 924.5
player: 1075
game: 4
baseline: 925
player: 1075
game: 5
baseline: 940
player: 1060
game: 6
baseline: 910
player: 1090
game: 7
baseline: 910
player: 1090
game: 8
baseline: 910
player: 1090
game: 9
baseline: 910
player: 1090
game: 10
baseline: 978.5687499999999
player: 1019.1129590266937
you win 10 games
```

baseline 4 – 4/10

```
game: 1
baseline: 1085.0
player: 909.3136413672219
game: 2
baseline: 1203.0
player: 790.9450540958704
game: 3
baseline: 0
player: 2000.0
game: 4
baseline: 97.65045992839768
player: 1897.4288839787919
game: 5
baseline: 1025.0
player: 968.6824935583675
game: 6
baseline: 1111.0
player: 883.9775912902385
game: 7
baseline: 1996.6188163134652
player: 0
game: 8
baseline: 1025.0
player: 973.8493660421086
game: 9
baseline: 387.6713679485537
player: 1607.5178034848484
game: 10
baseline: 0
player: 1999.0000000000002
you win 4 games
```

baseline 5 – 5/10

```
game: 1
baseline: 493.0
player: 1499.3681995517902
game: 2
baseline: 928.1337012454715
player: 1067.223425311368
game: 3
baseline: 928.6361788145259
player: 1068.1680447036315
game: 4
baseline: 397.0
player: 1601.945801810811
game: 5
baseline: 475.0
player: 1519.6940779171593
game: 6
baseline: 1303.9358361061463
player: 691.4956698713472
game: 7
baseline: 1000.0045328998849
player: 993.7190810820343
game: 8
baseline: 1099.3978420604283
player: 895.5095716638162
game: 9
baseline: 1010.3625249170836
player: 984.9371899182763
game: 10
baseline: 1247.3011300238531
player: 748.6965516295219
you win 5 games
```