Japanese Mahjong AI on Android

Source Code:

Team12Andjong.zip

Final Report:

AI Final Project Report

Team 12

· Topics: AI Mahjong

· Name/ID of team members

n 蔣佳航 **B98505005 <- ME**

Job: Tiles on hand(手牌) management, Rounds to reach(聽牌) calculation, Referenced code research and analysis, Modify referenced code to our team approach, Paper Report

n 張偉哲 **B98902018**

Job: Historical game record(牌譜) analysis, Degree of dangerous(危險度) calculation, Called(叫牌) decision making, Referenced code research and analysis, Paper Report

n 王盛平 **B98902062**

Job: Tree search for mahjong algorithm design, Dangerous point(危險分數) calculation, Expectation value calculation, Strategy selection

n **Collaboration/communication mechanism**

· Problem Definition

Our goal is to create a mahjong agent that can be the first place of a Japanese mahjong match with a very high percentage. The reason is, in Japanese mahjong, the first place can get extra rewards than what the player wins in the game. That is, we want to let our agent can win points stably during games.

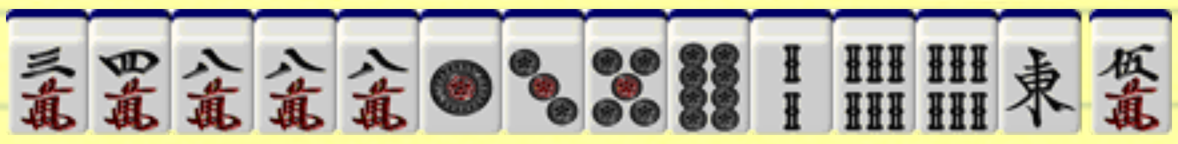


Figure 1. which tile should be discarded?

· Proposed method

First of all, the hand tiles must be some form or ron(和牌). So no matter which method it is, at least an agent must have to count how many rounds needed to ron/reach and whether that format can earn points. Hence, all of the method we proposed need to do these thing:

(1) Using tree search to make up the tiles on hand becomes a combination, which includes of

(a) 3-in-a-kind(刻子)

(b) 3-in-a-row(順子)

The 2 above can be called a “menzu”(面子)

(c) a pair(對子)

(d) 3-in-a-row but lack of one(斷順)

The following should be considered tile-by-tile separately

(e) stand alone tile(孤張).

(2) Compute how many rounds need at least to reach, which means only 1 tile away from a ron. There are 3 kinds of numbers need to be count because except for the normal ron format(4 menzu and 1 pair), there are still some special formation, they are: Kokushi(國士無雙) and 7-pairs(七對子).

(3) Because called tile(叫牌) may broke the ron form, how to decide whether to call or not is also a main topic. Our method is to check that whether this call breaks our current form or not, if the Han(飜) is still there or our call may gain a Han immediately, then it would not be a problem. Otherwise, be silence.

The followings are our proposed method in this project:

1. **Purely decrease how many rounds needed(進聽數) to reach(和牌)**

That is, for each turn a player got 14 tiles on hand, discard a tile that can reduce the number of rounds needed to reach, otherwise throw the tile a player just got away.

The method sounds simple and direct, but it is not smart. This method has the highest efficiency, which leads to a fluent performance when program running, that can let human to watch it play comfortably. Due to its low difficulty, we add some features so that it won’t be too stupid. One of them is to count the points our hand tiles worth each round. This may help us to get a better result when picking tiles that can replace of each other’s position but not destroying the menzu and pair the player already had. Actually this method is also be implemented on Andjong, but we think the way it implement just not good enough, therefore we enhanced it to fit our team requirements and become one of our method to solve the problem we picked.

2. **Purely counting the maximum expected value we can earn**

This might also sounds simple, but the truth is it isn’t. It depends on how many turns we want to predict. And the best way is using an iteration function to run through a tree, which each rounds compute the best value we can achieved if we throw a card from our hands and draw another one “until a ron format is done”.

The algorithm can be written as the following:

```
Algorithm outcome(Tile root[], int size = 13, int max_depth, int tsumo_left, pai_left[34])


 if(max_depth <= 0)
   return 0


 S = 0
 for valid_tiles && pai_left[Y] > 0: Add it to our hands: Y
   P = pai_left[Y] / tsumo_left
   Tile X = root + Y
   if X.ron
     S += P * X.score
   else
     pai_left[Y] -= 1
```

```
        S' = 0
        for j = 0 to size && X[j] != Y && (the rounds to reach doesn't decreased)
            Tile x' = X - X[j]
            S' = max(S' , P * outcome(x', size, max_depth-1
                        , tsumo_left-1, pai_left) )
        pai_left[Y] += 1
        S += S'
return S
```

where the definition of valid tiles(有效牌) is the tile that can be used in our hands to combine with and to form some combination(alone cards not included). This help us to reduce the size of tree.

A problem of this method is the depth of this tree is definitely big and thus time- and space-consuming. So we limit the depth with a max_depth that can be used to prevent this tree from growing forever.

3. **Using degree of dangerous(危險度) and expected value together to count a score, then used this score to make decision**

This is the enhancement version of the 2nd method. Since mahjong is a multiplayer game that each player are heading for a ron, each discarded tile can possibly helping other player get score. Therefore, adding the score that other player gain from the tile we discarded as a part of our degree of dangerous evaluation seems reasonable here.

We've collected over 60,000 games record to help us do some basic analysis and use statistic method to compute the percentage between any 2 tiles. That is, when a player discard a tile, for each of 34 kinds of tile, what is the probability that that player may ron that. In short, try to predict other's hand by their discarded tile. After we made a 34x34 table by tracing all of 60,000 games, we can use the probability table with us by the algorithm:

```
    double dangerous_value[3][34]
for player = 1 to 3
    reach = false
    for turn = 1 to last
        X = discard[player][turn]: dangerous_value [player][X] = 0
                if(discard[player][turn] is reach) reach = true
        a = 1 + 0.025 * turn, if reach == false
          = 1.1, if reach == true
        b = 1, if tile is not numbers
          = 1.1, if tile is 1 or 9
          = 1.15, if 5
          = 1.2, if tile is other numbers
        c = 1, if X appeared for 1st time
          = 0.25, if X appeared for 2nd time
          = 0.125, if X appeared for 3rd time
          = 0.0625, if X appeared for 4th time
for tile = 0 to 33
    dangerous_value [player][tile] += a * b * c * table[有X][tile]
```

Then we can added this value to our expectation value calculation. This may help our agent to have a lower chance to lose points to other while keeping our ways to get a ron.

· Result

We evaluate our agent by competing with the AI that Andjong implements, and calculate the average value of points get and the probability of getting first place.

By using BlueStacks to run our games for 800 times, and keeping records of how many times we got each places and our average score, then we can do the recording jobs very easily. The figure showing below are a sample image of running our code with recorded value showing out.

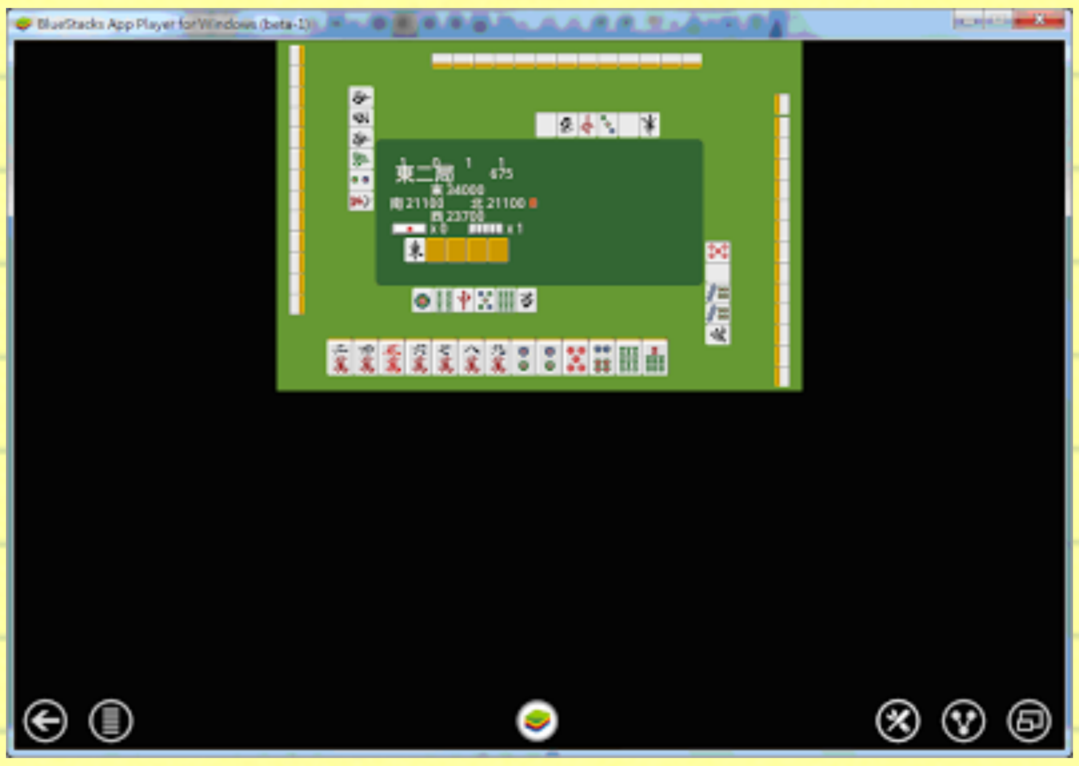


Figure 2. running our program on BlueStacks

.

	Method 1	Method 2	Method 3
Games played	500	200	100
1st place game	287	122	64
1st place probability	57.4%	61.0%	64.0%
2nd place game	130	36	23
2nd place probability	26.0%	18.0%	23.0%
3rd place game	44	31	11
3rd place probability	8.8%	15.5%	11.0%
4th place game	39	11	2
4th place probability	7.8%	5.5%	2.0%
Average scoring	34149.11	34610.92	36721.29

· Reference

1. http://cs229.stanford.edu/proj2009/Loh.pdf

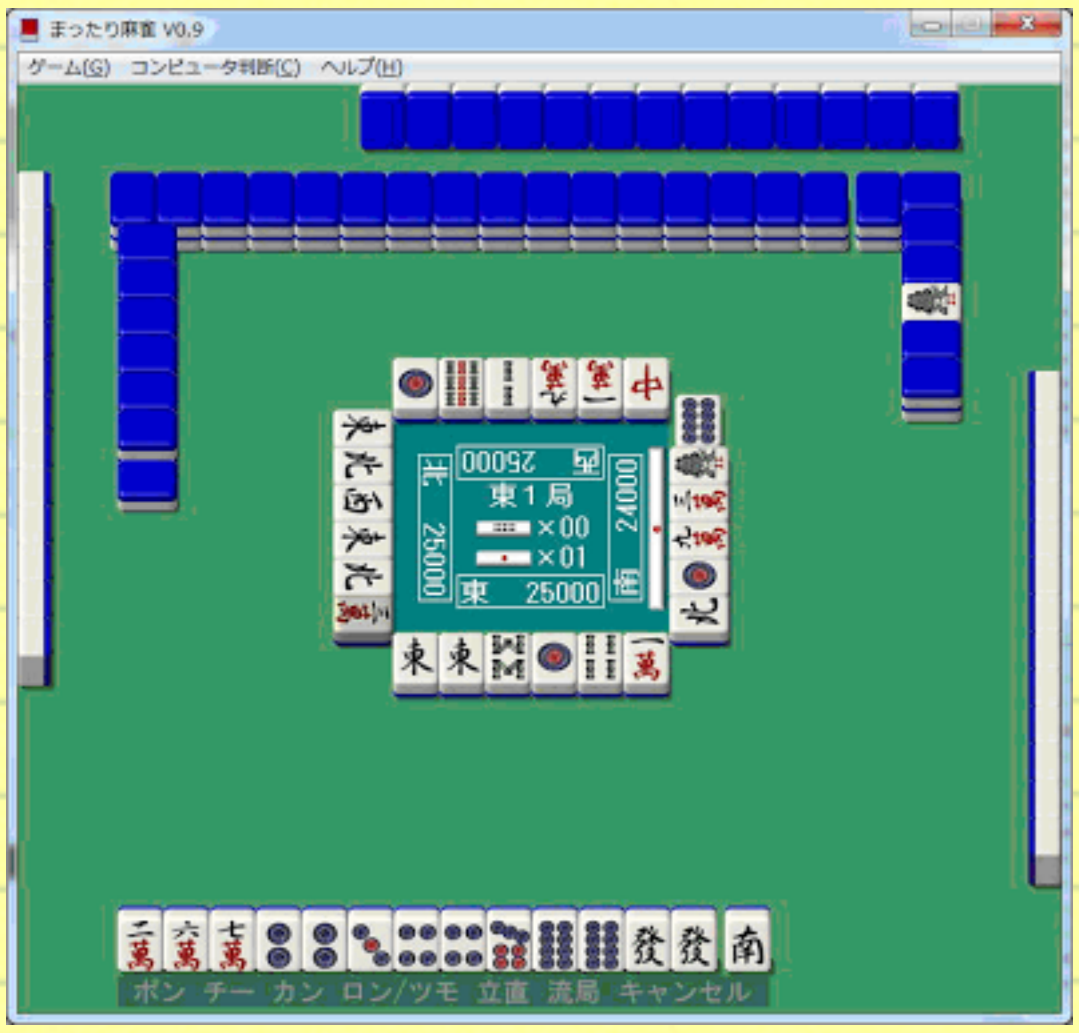
A referenced paper that also implements a AI mahjong agent.

2. http://en.sourceforge.jp/projects/andjong/

Andjong Project. This is the program that we decided to implement our AI agent on.

3. http://homepage2.nifty.com/kmo2/

A referenced implementation that using a similar algorithm as ours, though it doesn't provide its source code, but it's still a good reference.



评论

您没有权限添加评论。