

INFO 6205 Program Structures & Algorithms

Final Project

Genetic Algorithm for Winning in Mahjong Game

Team Number: 102

Team Members:

Li Hua (001898699)

Gongdai Liu (001739979)

Background Introduction:

Mahjong is a tile-based game that was developed in China during the Qing dynasty and has spread throughout the world since the early 20th century. It is commonly played by four players. The game and its regional variants are widely played throughout Eastern and South Eastern Asia and have become popular in Western countries too. Similar to the Western card game rummy, Mahjong is a game of skill, strategy, and calculation and involves a degree of chance.

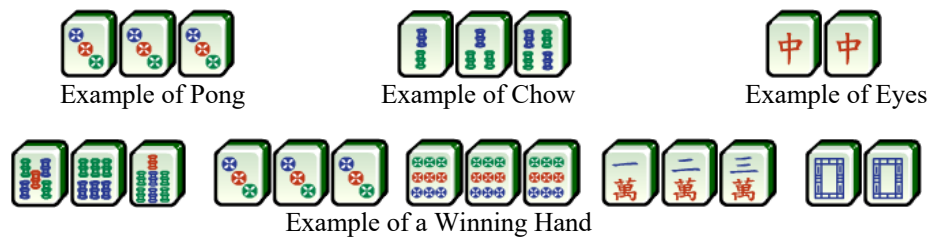


Rules:

The game is played with a set of 136 tiles based on Chinese characters and symbols. Each player begins by receiving 13 tiles, and then in turn draws a tile from the wall (tiles stack); as long as the tile drawn will not lead to a winning hand, the player proceeds to discard a tile (either the tile just drawn, or a tile in the hand) to maintain a hand of 13. The discarded tile is thrown into the center. The other players have an opportunity to seize the discarded tile; if no one takes it, the turn continues to the next player. Play continues this way until one player has a winning hand (using the 14th drawn tile to form 4 melds/sets and a pair/eye).

Melds consist of either a *Pong* (three identical tiles) or a *Chow* (three Simple tiles all of the same suit, in numerical sequence). Melds may be formed by drawing a tile from the wall, or by seizing another player's discard.

Eyes (also known as a pair) are two identical tiles which are an essential part of a legal winning hand.



Problem Inspiration:

When playing Mahjong, the most difficult part is to decide which tile to discard after drawing a tile either from wall or from other players. A good strategy will get you a winning hand faster than your competitors, which is the goal of this game.

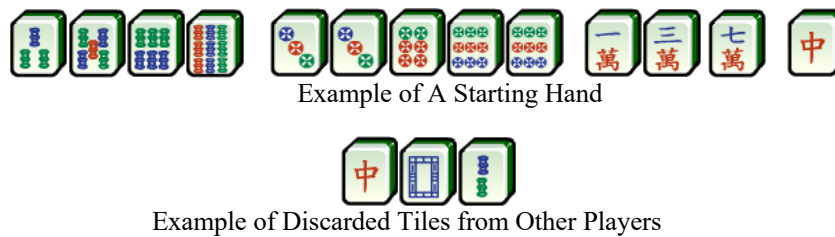
In this project, we simulated the process of Mahjong playing, including randomly assigning starting hands (parent), drawing tile from other players' discards (parent), drawing tiles from the wall (mutation), and checking how likely it will make a winning hand after a certain discard (fitness). We used genetic algorithm to select the most fitted tiles to keep (phenotype) for each round (generation) and decide which tile to discard accordingly (problem). For the next round of play, the selected hands will become the heritable gene. The process will repeat until there is a winning hand, or the available Mahjong stack (the wall) is empty.

As there are in total 136 tiles in a set of Mahjong, there are $(136 \times 135 \times \dots \times 124 \times 123) / (14!)$ different combinations of hands. We implemented a function which will generate all the possible winning hands and compared a random hand with each of the winning hands to check the fitness.

Implementation Design

Genotype:

At the start of the game, a random starting hand with 13 tiles (chromosomes) is one parent, the other parent is the tile discarded by other players which can be seized.



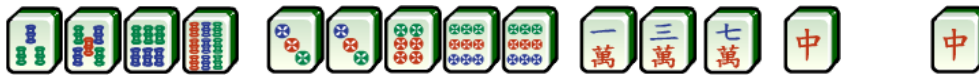
```

else if(i == 1){
    for(int j = 0; j < 13; j++){
        System.out.print("55555555555");
        int temp = rand.nextInt(size);
        MaJiang mj = maJiangList.getMaJiangList().get(temp);
        p2.inHand.add(mj);
        maJiangList.getMaJiangList().remove(mj);
        size = maJiangList.getMaJiangList().size();
    }
}

```

Crossing Over:

If a player seizes a tile from other players' discards, there is a crossing over. After seizing a tile, the player needs to discard a tile from hand. Part of the tiles from the previous hand and the seized tile are inherited, which consists the child.



Example of A Crossing Over (previous hand + seized tile)

```

public void start() throws IOException{
    Random rand = new Random();
    int first = 0;
    boolean condition = true;
    size = maJiangList.getMaJiangList().size();
    int position = 0;
    while(condition && size > 0){
        size = maJiangList.getMaJiangList().size();
        int temp = rand.nextInt(size);
        MaJiang mj = maJiangList.getMaJiangList().get(temp);
        maJiangList.getMaJiangList().remove(mj);
    }
}

```

```

if(first == 0){
    int k = rand.nextInt( bound: 2);
    if(onTable.size() != 0){
        if(k == 0)
            p1.inHand.add(position, mj);
        else{
            p1.inHand.add(position, onTable.get(onTable.size() - 1));
            onTable.remove( index: onTable.size() - 1);
        }
    }
    else
        p1.inHand.add(position, mj);
    File file = new File( pathname: "Output Result.txt");
    file.createNewFile();
    FileWriter fw = new FileWriter(file);
    sortInHand(p1.inHand);

    if(whetherHu(p1, getHu) == false){
        position = huOrThrow(p1.inHand, p1, getHu, fw);
        MaJiang lose = p1.inHand.get(position);

        onTable.add(lose);
        p1.deleteInHand(lose);
        first++;
    }else{
        condition = false;
        System.out.println("Congratulations!! Player1 is Win!!!!");
        break;
    }
    continue;
}
}

```

Selector:

After combining both of the parents, we have 14 tiles in hand. Then we suppose the tile with an index i is discarded and compare the rest tiles with all the possible winning hands. We count the number of overlapping tiles for each i , and discard the tile with the most similar hand.

Fitness function:

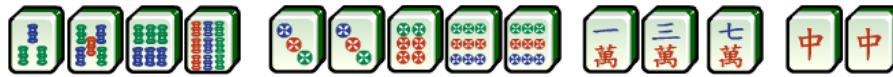
We set a variable 'recorder' to count the overlapping tiles between hand tile and winning hand. The larger 'recorder' is, the closer the player is to a winning hand. With a 'recorder' number of 14, the player wins the game.


```

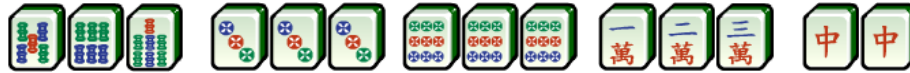
private int huOrThrow(ArrayList<MaJiang> inHand, Player p, GetHu getHu, FileWriter fw) throws IOException {
    MaJiang[][] reference = getHu.finalResArray;
    int n = reference.length;
    int index = 0;
    ArrayList<Integer> temp = p.getThrowList();
    for(int j = 0; j < 14; j++){
        for(int i = 0; i < n; i++){
            int recorder = 0;
            int pointer = 0;
            while(pointer < 14){
                MaJiang m1 = inHand.get(pointer);
                MaJiang m2 = reference[i][pointer];
                if(m1.getName().equals(m2.getName()) && m1.getNumber() == m2.getNumber() && pointer!=j){
                    recorder++;
                }
                pointer++;
            }
            System.out.println("Number of MaJiang: " + Integer.toString(j) + "-- " + "Recorder: "+Integer.toSt
            if(recorder != 0){
                String newLine = System.getProperty("line.separator");
                fw.write( str: "Number of MaJiang: " + Integer.toString(j) + "-- " + "Recorder: "+Integer.toStri
            }
            p.setScore(Math.max(p.getScore(), recorder));
        }
        temp.add(p.getScore());
    }
    int n1 = p.getThrowList().size();
    int or = n1 - 14;
    int i = or;
    int tempI = 0;
    while(i < n1){
        if(tempI <= p.getThrowList().get(i)){
            index = i;
            tempI = p.getThrowList().get(i);
        }
        i++;
    }
    return (index - or);
}

public boolean whetherHu(Player p, GetHu getHu){
    System.out.print("9999999999");
    MaJiang[][] reference = getHu.finalResArray;
    ArrayList<MaJiang> inHand = p.inHand;
    int n = reference.length;
    for(int i = 0; i < n; i++){
        for(int j = 0; j < 14; j++){
            MaJiang mj1 = reference[i][j];
            MaJiang mj2 = inHand.get(j);
            if(mj1 != mj2)
                return false;
        }
    }
    return true;
}

```



Example of A Crossing Over



Example of a Winning Hand


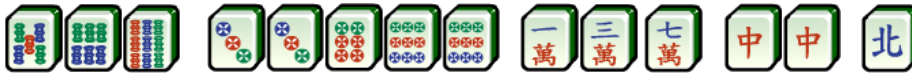
For the example hand, when $i = 0, 1, 2, 3, 6, 10, 11$, the candidate has the most similarity with the winning hand, with 7 overlapping tiles.

Mutation:

Every time a player draws a tile from the wall, it could bring new genes to the hands, which can be either good or bad for the evolution. And the possibility of mutation is $1/14$.



Example of A Candidate

When the player draws a tile from the wall as: 
The hand will be: 

As the new tile doesn't help the player gain more recorder points, it slows down the evolution.

Evolution:

Each round of play is one generation. The game will continue until either one of the players has a winning hand, or all the tiles in the 'wall' have been drawn. With either situation, the number of overlapping tiles increases after the evolution.

Unit Tests:

Our project builds and passes our unit test cases:

```

public static void main(String[] args){
    test t1 = new test();
    MaJiangList mjl = new MaJiangList();
    int n = mjl.getMaJiangList().size();
    Random r = new Random();
    int k = 0;
    while(k < 10){
        ArrayList<MaJiang> tempMJ = new ArrayList<>();
        for(int i = 0; i < 14; i++){
            int t = r.nextInt(n);
            tempMJ.add(mjl.getMaJiangList().get(t));
        }
        for(int j = 0; j < tempMJ.size(); j++){
            System.out.print(Integer.toString(tempMJ.get(j).getNumber()) + ":" + tempMJ.get(j).getName() + ", ");
        }
        System.out.println(".. After Sort:");
        t1.sortInHand(tempMJ);
        for(int j = 0; j < tempMJ.size(); j++){
            System.out.print(Integer.toString(tempMJ.get(j).getNumber()) + ":" + tempMJ.get(j).getName() + ", ");
        }
        System.out.println("..");
        k++;
    }
}

```

```

"C:\Program Files (x86)\Java\jdk1.8.0_181\bin\java.exe" ...
9:wan, 4:wan, 6:wan, 9:bing, 3:tiao, 9:bing, 8:wan, 8:wan, 0:fa, 2:wan, 5:wan, 4:tiao, 0:fa, 5:bing, .. After Sort:
5:bing, 9:bing, 9:bing, 0:fa, 0:fa, 3:tiao, 4:tiao, 2:wan, 4:wan, 5:wan, 6:wan, 8:wan, 8:wan, 9:wan, ..
9:wan, 2:wan, 0:fa, 8:bing, 9:bing, 1:tiao, 3:bing, 3:wan, 7:bing, 0:bai, 2:bing, 6:wan, 5:tiao, 2:bing, .. After Sort:
0:bai, 2:bing, 2:bing, 3:bing, 7:bing, 8:bing, 9:bing, 0:fa, 1:tiao, 5:tiao, 2:wan, 3:wan, 6:wan, 9:wan, ..
4:wan, 1:wan, 0:bai, 3:tiao, 0:dong, 5:bing, 4:wan, 6:bing, 5:tiao, 2:wan, 0:nan, 5:bing, 7:tiao, 0:xi, .. After Sort:
0:bai, 5:bing, 5:bing, 6:bing, 0:dong, 0:nan, 3:tiao, 5:tiao, 7:tiao, 1:wan, 2:wan, 4:wan, 4:wan, 0:xi, ..
6:bing, 0:bei, 3:tiao, 1:wan, 0:xi, 4:wan, 0:bei, 2:bing, 5:bing, 5:bing, 1:bing, 2:bing, 7:wan, 0:dong, .. After Sort:
0:bei, 0:bei, 1:bing, 2:bing, 2:bing, 5:bing, 5:bing, 6:bing, 0:dong, 3:tiao, 1:wan, 4:wan, 7:wan, 0:xi, ..
9:tiao, 3:tiao, 2:tiao, 9:tiao, 3:tiao, 8:bing, 3:tiao, 0:bai, 1:wan, 0:zhong, 1:bing, 3:tiao, 0:xi, 8:bing, .. After Sort:
0:bai, 1:bing, 8:bing, 8:bing, 2:tiao, 3:tiao, 3:tiao, 3:tiao, 3:tiao, 9:tiao, 9:tiao, 1:wan, 0:xi, 0:zhong, ..
4:wan, 0:nan, 5:bing, 3:tiao, 3:wan, 3:bing, 0:xi, 5:bing, 6:bing, 3:tiao, 2:wan, 1:bing, 0:zhong, 6:bing, .. After Sort:
1:bing, 3:bing, 5:bing, 5:bing, 6:bing, 6:bing, 0:nan, 3:tiao, 3:tiao, 2:wan, 3:wan, 4:wan, 0:xi, 0:zhong, ..
6:bing, 0:dong, 9:tiao, 2:wan, 1:tiao, 2:tiao, 4:tiao, 6:wan, 4:wan, 6:tiao, 9:bing, 4:tiao, 5:tiao, 8:wan, .. After Sort:
6:bing, 9:bing, 0:dong, 1:tiao, 2:tiao, 4:tiao, 4:tiao, 5:tiao, 6:tiao, 9:tiao, 2:wan, 4:wan, 6:wan, 8:wan, ..
0:nan, 9:tiao, 0:nan, 7:tiao, 0:bai, 2:bing, 3:bing, 0:zhong, 6:tiao, 0:bai, 0:fa, 8:bing, 1:wan, 0:fa, .. After Sort:
0:bai, 0:bai, 2:bing, 3:bing, 8:bing, 0:fa, 0:fa, 0:nan, 0:nan, 6:tiao, 7:tiao, 9:tiao, 1:wan, 0:zhong, ..
7:tiao, 0:zhong, 0:xi, 8:wan, 0:bei, 7:wan, 0:dong, 7:tiao, 2:bing, 0:xi, 1:tiao, 0:nan, 3:bing, 4:tiao, .. After Sort:
0:bei, 2:bing, 3:bing, 0:dong, 0:nan, 1:tiao, 4:tiao, 7:tiao, 7:tiao, 7:wan, 8:wan, 0:xi, 0:xi, 0:zhong, ..
5:bing, 9:bing, 9:bing, 0:fa, 3:bing, 0:xi, 6:tiao, 0:dong, 5:bing, 0:zhong, 0:zhong, 5:tiao, 1:tiao, 9:bing, .. After Sort:
3:bing, 5:bing, 5:bing, 9:bing, 9:bing, 9:bing, 0:dong, 0:fa, 1:tiao, 5:tiao, 6:tiao, 0:xi, 0:zhong, 0:zhong, ..

```

Process finished with exit code 0


```

public static void main(String[] args){
    ArrayList<MaJiang> temp = new ArrayList<>();
    MaJiang m3 = new MaJiang(); m3.setNumber(3); m3.setName("tiao");temp.add(m3);
    MaJiang m5 = new MaJiang(); m5.setNumber(5); m5.setName("tiao");temp.add(m5);
    MaJiang m6 = new MaJiang(); m6.setNumber(6); m6.setName("tiao");temp.add(m6);
    MaJiang m9 = new MaJiang(); m9.setNumber(9); m9.setName("tiao");temp.add(m9);
    MaJiang mB32 = new MaJiang(); mB32.setNumber(3); mB32.setName("bing");temp.add(mB32);
    MaJiang mB33 = new MaJiang(); mB33.setNumber(3); mB33.setName("bing");temp.add(mB33);
    MaJiang mB6 = new MaJiang(); mB6.setNumber(6); mB6.setName("bing");temp.add(mB6);
    MaJiang mB93 = new MaJiang(); mB93.setNumber(9); mB93.setName("bing");temp.add(mB93);
    MaJiang mB94 = new MaJiang(); mB94.setNumber(9); mB94.setName("bing");temp.add(mB94);
    MaJiang mW1 = new MaJiang(); mW1.setNumber(1); mW1.setName("wan");temp.add(mW1);
    MaJiang mW3 = new MaJiang(); mW3.setNumber(3); mW3.setName("wan");temp.add(mW3);
    MaJiang mW7 = new MaJiang(); mW7.setNumber(7); mW7.setName("wan");temp.add(mW7);
    MaJiang mZ52 = new MaJiang(); mZ52.setNumber(0); mZ52.setName("zhong");temp.add(mZ52);
    MaJiang mZ53 = new MaJiang(); mZ53.setNumber(0); mZ53.setName("zhong");temp.add(mZ53);

    huOrThrow(temp);
}

```

"C:\Program Files (x86)\Java\jdk1.8.0_181\bin\java.exe" ...

```

Number of Element: 0----Recorder: 7
Number of Element: 1----Recorder: 7
Number of Element: 2----Recorder: 7
Number of Element: 3----Recorder: 7
Number of Element: 4----Recorder: 6
Number of Element: 5----Recorder: 6
Number of Element: 6----Recorder: 7
Number of Element: 7----Recorder: 6
Number of Element: 8----Recorder: 6
Number of Element: 9----Recorder: 6
Number of Element: 10----Recorder: 7
Number of Element: 11----Recorder: 7
Number of Element: 12----Recorder: 6
Number of Element: 13----Recorder: 6

```

Process finished with exit code 0