The Jungle Game

Group 8

16098537d	Yu Jing
17082705d	Qin Yaxue
17081996d	Gao Haorui
17083686d	Xia Jialu

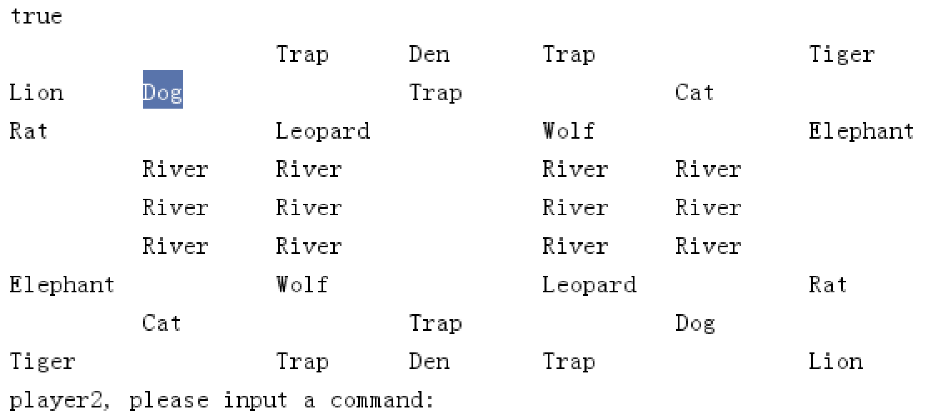
Team role

16098537d	Yu Jing	Controller, function
17082705d	Qin Yaxue	Function, GUI
17081996d	Gao Haorui	Function, testing
17083686d	Xia Jialu	Controller, GUI

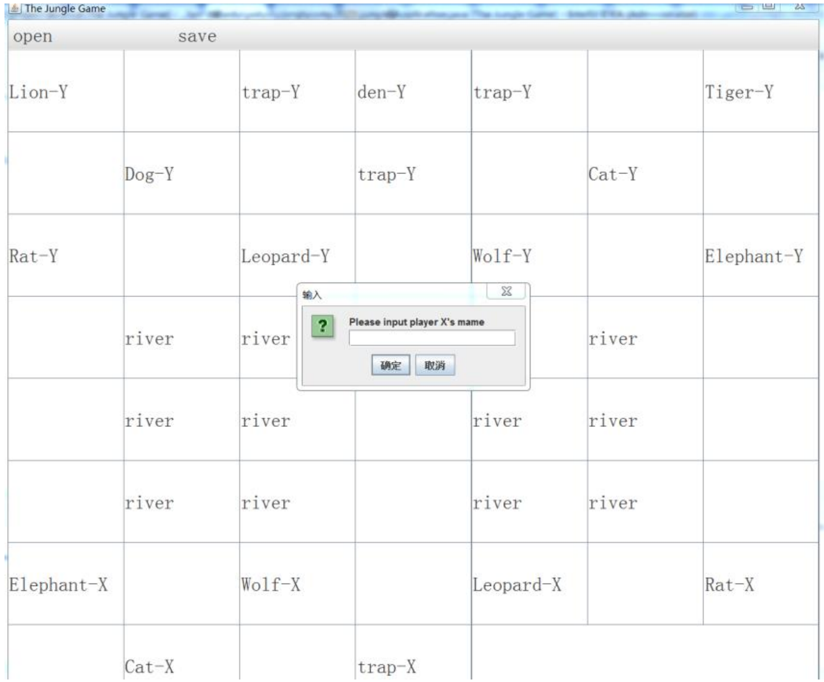
Architecture

Overall, Model- View -Controller pattern

View {command line, GUI}.



```
true
                    Trap      Den       Trap                Tiger
Lion      Dog                 Trap                Cat
Rat                 Leopard             Wolf                Elephant
          River     River               River     River
          River     River               River     River
          River     River               River     River
Elephant            Wolf                Leopard             Rat
          Cat                 Trap                Dog
Tiger               Trap      Den       Trap                Lion
player2, please input a command:
```



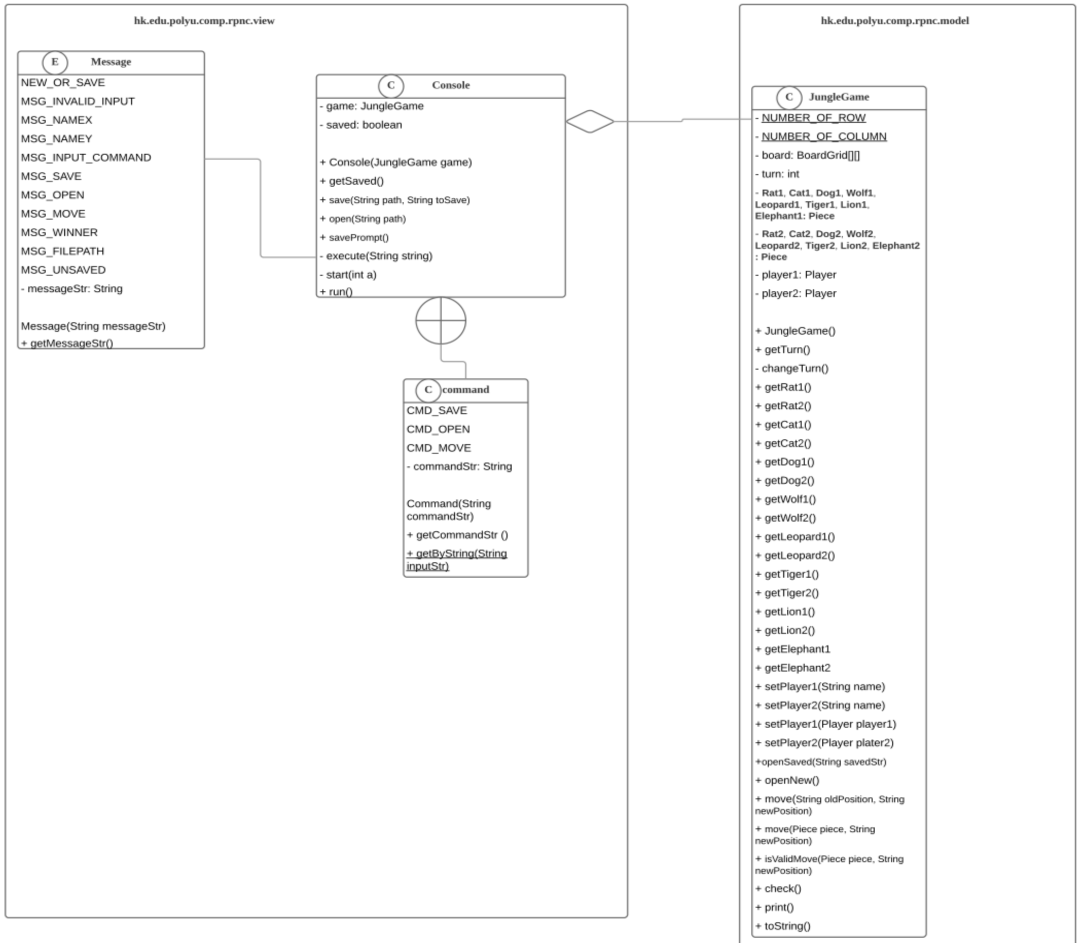
The Jungle Game

open save

Lion-Y		trap-Y	den-Y	trap-Y		Tiger-Y
	Dog-Y		trap-Y		Cat-Y	
Rat-Y		Leopard-Y		Wolf-Y		Elephant-Y
	river	river			river	
	river	river		river	river	
	river	river		river	river	
Elephant-X		Wolf-X		Leopard-X		Rat-X
	Cat-X		trap-X			

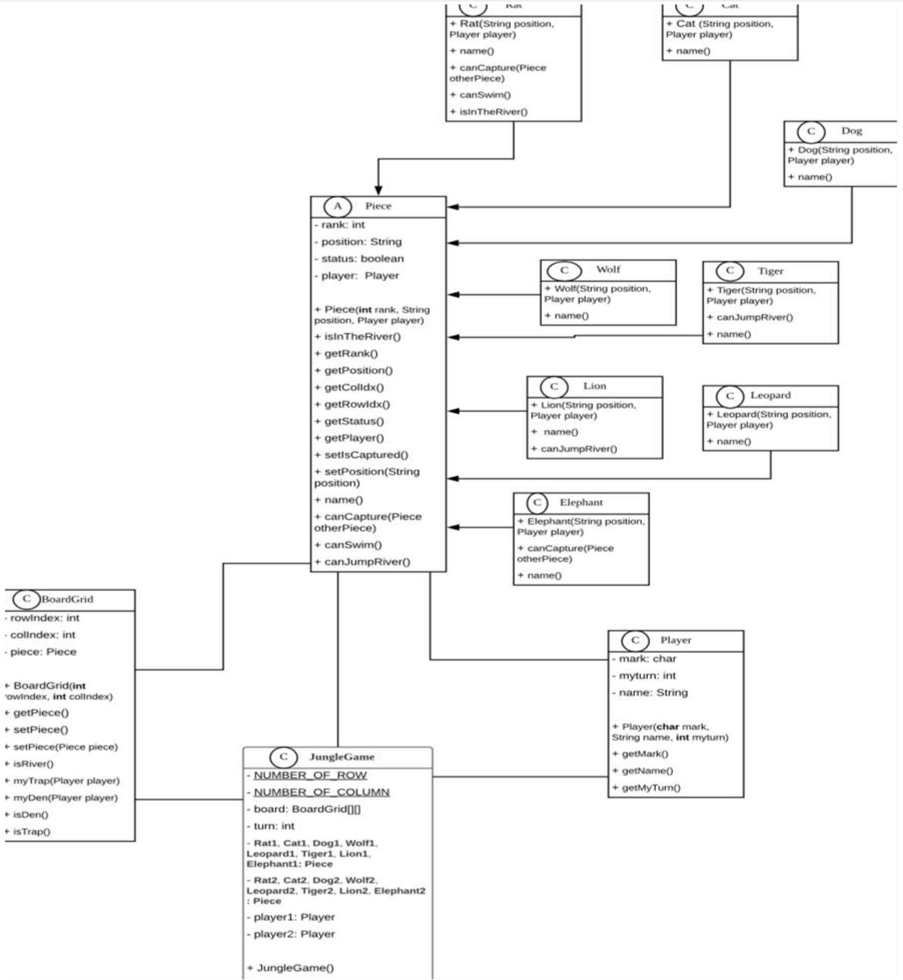
Architecture

/* The View part. */

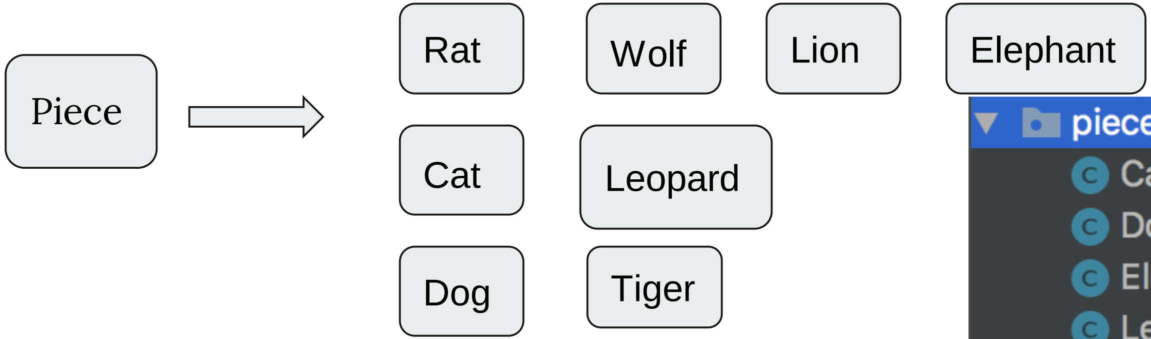


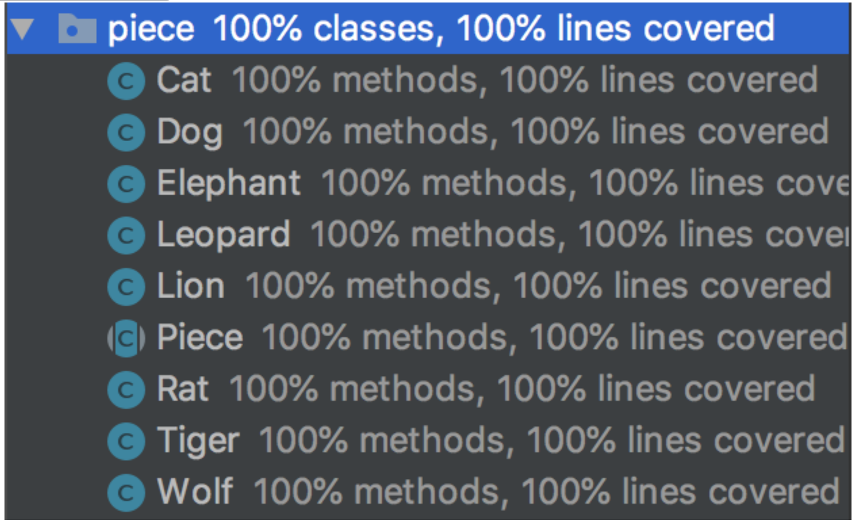
Architecture

/* The Model part. */



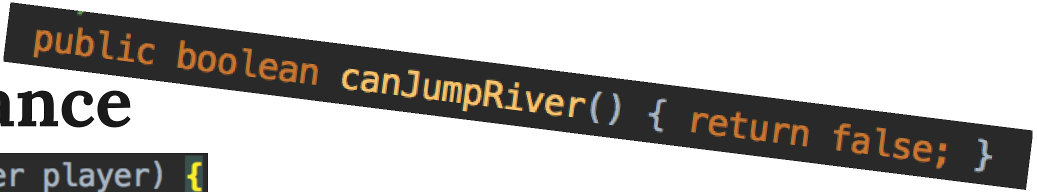
Design - Piece inheritance



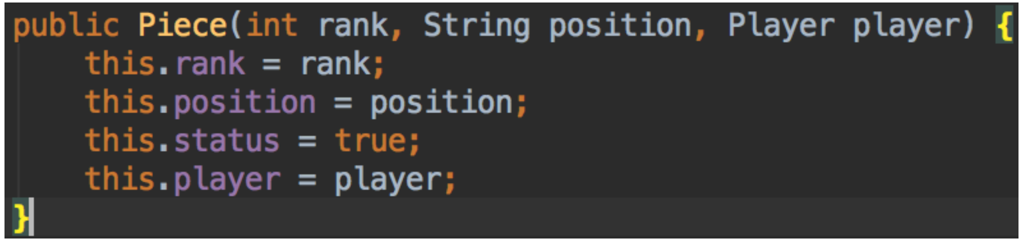

```
▼ piece  100% classes, 100% lines covered
    Cat  100% methods, 100% lines covered
    Dog  100% methods, 100% lines covered
    Elephant  100% methods, 100% lines cove
    Leopard  100% methods, 100% lines cove
    Lion  100% methods, 100% lines covered
    Piece  100% methods, 100% lines covered
    Rat  100% methods, 100% lines covered
    Tiger  100% methods, 100% lines covered
    Wolf  100% methods, 100% lines covered
```

The 8 pieces are inherited from the Class Piece.

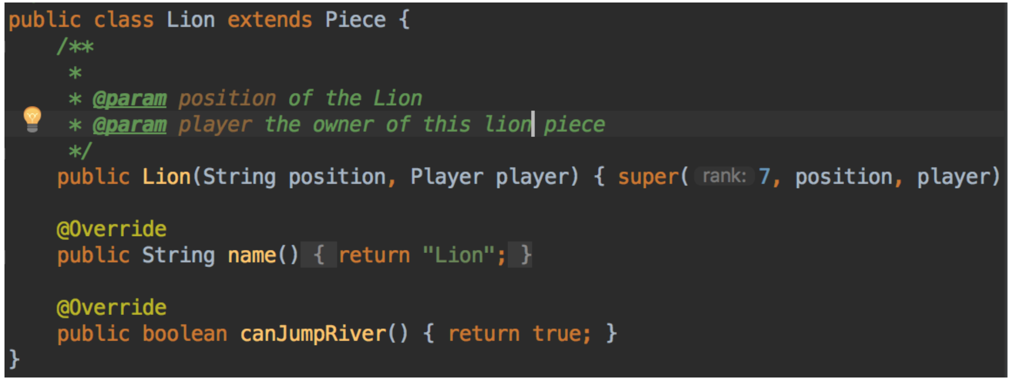
Design - Piece inheritance



```
public boolean canJumpRiver() { return false; }
```



```
public Piece(int rank, String position, Player player) {
    this.rank = rank;
    this.position = position;
    this.status = true;
    this.player = player;
}
```



```
public class Lion extends Piece {
    /**
     *
     * @param position of the Lion
     * @param player the owner of this lion piece
     */
    public Lion(String position, Player player) { super( rank: 7, position, player)

    @Override
    public String name() { return "Lion"; }

    @Override
    public boolean canJumpRiver() { return true; }
}
```

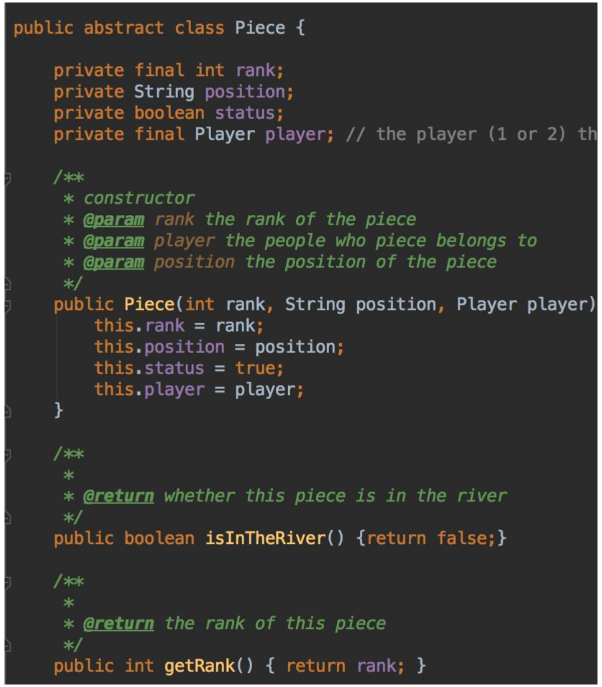
E.g: Lion inherit from Piece and get the attributes in super, in the meantime override the canJumpRiver.

1. Limit the redundant code
2. Could allow each “child” has own unique attribute.

Design - Encapsulation

Combine the data structure and the algorithm inside together in a “box”

Design - Piece Abstraction

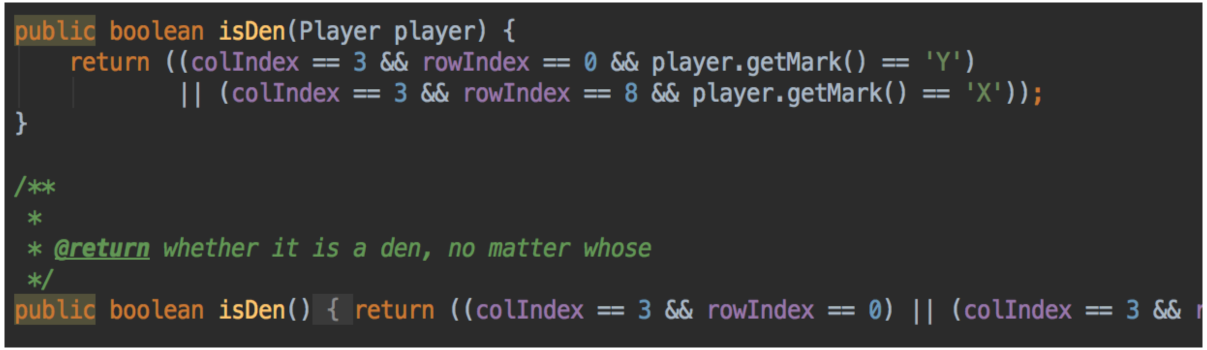


```java
public abstract class Piece {

    private final int rank;
    private String position;
    private boolean status;
    private final Player player; // the player (1 or 2) th

    /**
     * constructor
     * @param rank the rank of the piece
     * @param player the people who piece belongs to
     * @param position the position of the piece
     */
    public Piece(int rank, String position, Player player)
        this.rank = rank;
        this.position = position;
        this.status = true;
        this.player = player;
    }

    /**
     *
     * @return whether this piece is in the river
     */
    public boolean isInTheRiver() {return false;}

    /**
     *
     * @return the rank of this piece
     */
    public int getRank() { return rank; }
```

1) hide the unneccessary information

2) clear interface

Design - BoardGrid Polymorphism



```
public boolean isDen(Player player) {
    return ((colIndex == 3 && rowIndex == 0 && player.getMark() == 'Y')
            || (colIndex == 3 && rowIndex == 8 && player.getMark() == 'X'));
}

/**
 *
 * @return whether it is a den, no matter whose
 */
public boolean isDen() { return ((colIndex == 3 && rowIndex == 0) || (colIndex == 3 && r
```

E.g: For the BoardGrid, use the polymorphism, in which we call isDen() and isDen(Player player).

1) With a different signature, flexibility.

Discussion about the *Oop*

1) manageable, maintainable

It is easier to be modified, we could trace it to a small unit.

2) easier to design

Quite a big project, but we could start from making the frame.

3) reusable

You could create some frequently used classes, which could be used in different method.

4) scalable

Q&A