CS 151

Final Project

Group 5

Xinken Zheng

Hao Tu

Shengda zhang

Zhiyuan Xie

GOMOKU(Connect Six)

Introduction

this assignment is designed to play a GoMoKu(Connect Six)game of writing a basic graphical user interface(GUI) in Java. The key concept are Swing components by using JFrame to create an interface and MouseListener. MouseListener is listener interface for receiving "interesting" mouse events (press, release, click, enter, and exit)on a component. (To track mouse moves and mouse drags, use the MouseMotionListener), MouseEvent for x and y coordinate , graphics-fillOval( ),and exception handling.

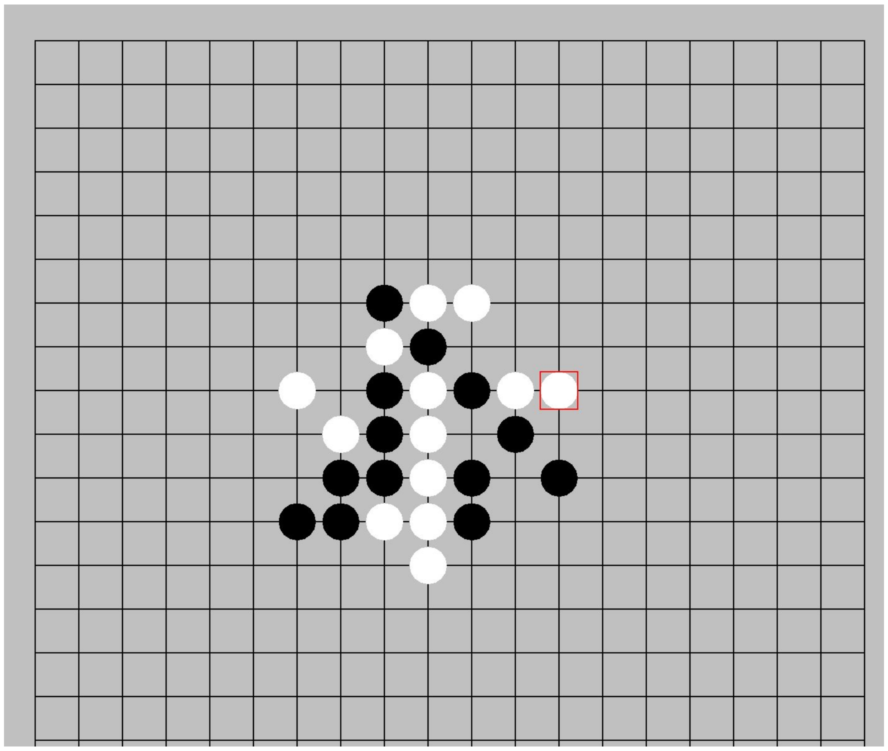
Gomoku, also called *Five in a Row*, but we create Connect Six which enhanced the difficulty of game. Gomoku is an abstract strategy board game. It is traditionally played with Go pieces (black and white stones) on a Go board.This game lets two players play Gomoku against each other.Black starts the game .When a player gets Connect Six,that player wins.The game ends in a draw if the board is filled before either played wins

1.1 Use Cases

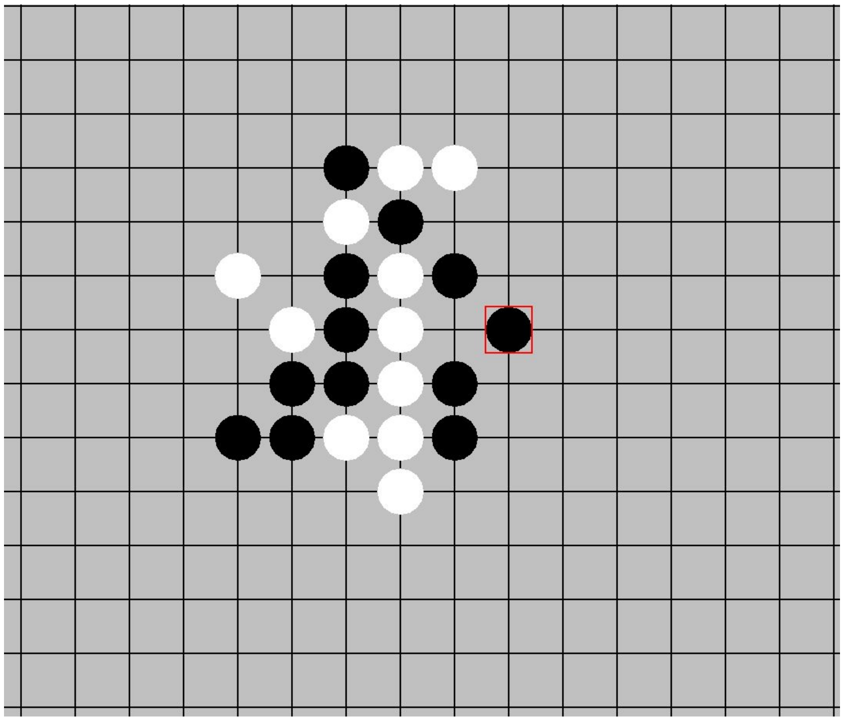
Use case name: 2-players of **Connect six** game
Context of use: one or two player at a computer to play one or more games of Connect Six. The computer manages the display of the game. enforces the rules, indicates the winner of each game.
Software Product Name: **Connect Six**
Primary Actor: player
Trigger: One player initiates the program.
Main Success condition:
(Actors are two Players. One player is designated Player1 with Black chess pieces and the other is designated Player2 with white chess pieces).
Steps 1 - 6 occur in sequence until players indicc
nnn ate they are finished.
1. The system displays an empty board.
2. The current player selects move.
3. The system validates the move.
4. System updates board with move and redisplays board.
5. The system validates that a win has occurred.
6. The system announces winner.

Extensions:
4a. Cell already occupied or out of bounds.
   4a1. The system informs players, prompts.
   4a2. Continue at 3 above.
6a. No win has occurred (drew).
   6a1. The system informs players.

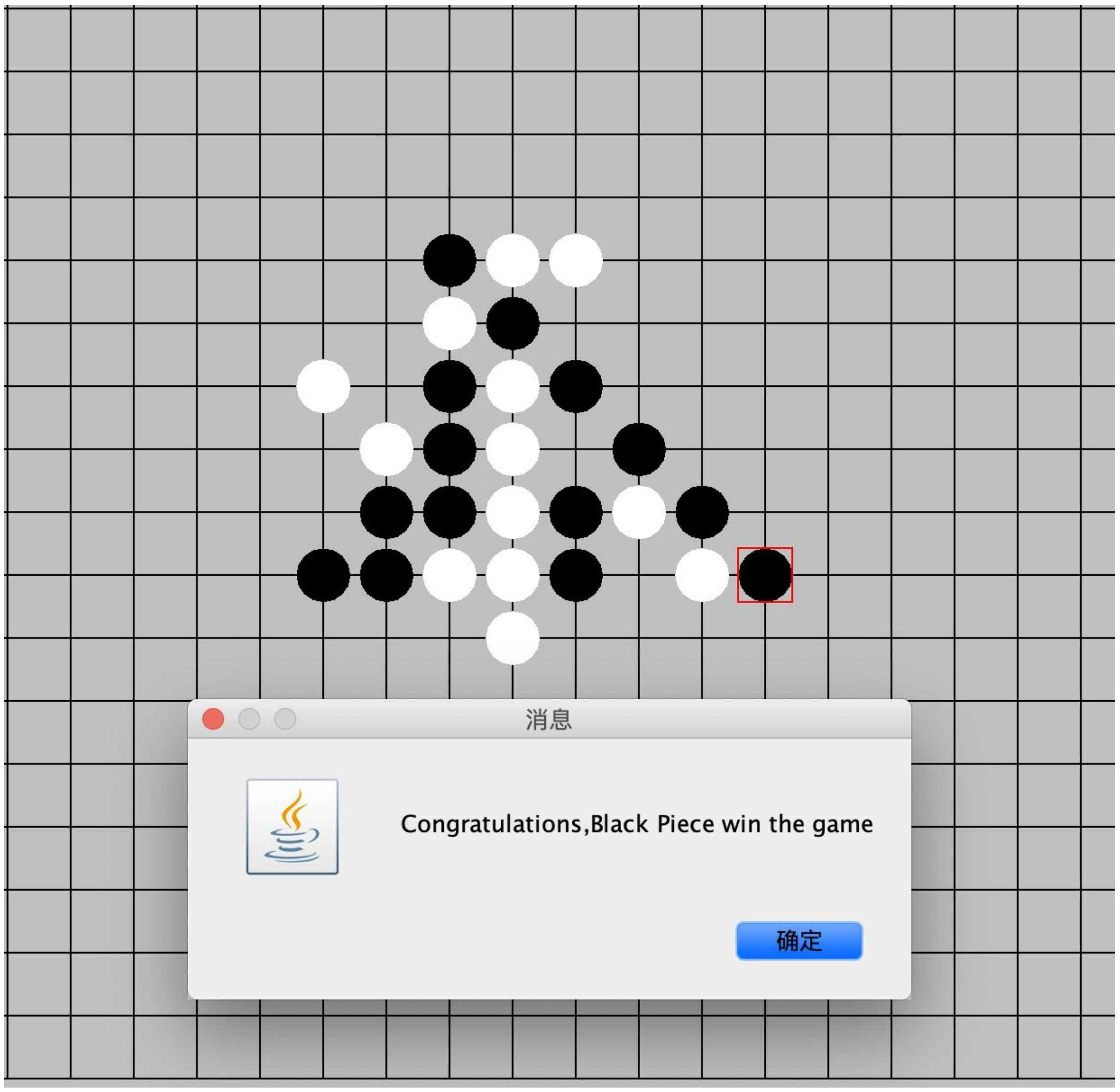
Case 1 .Undo



After click Undo button, the black chess pieces from 5 in rows changed to four in rows .



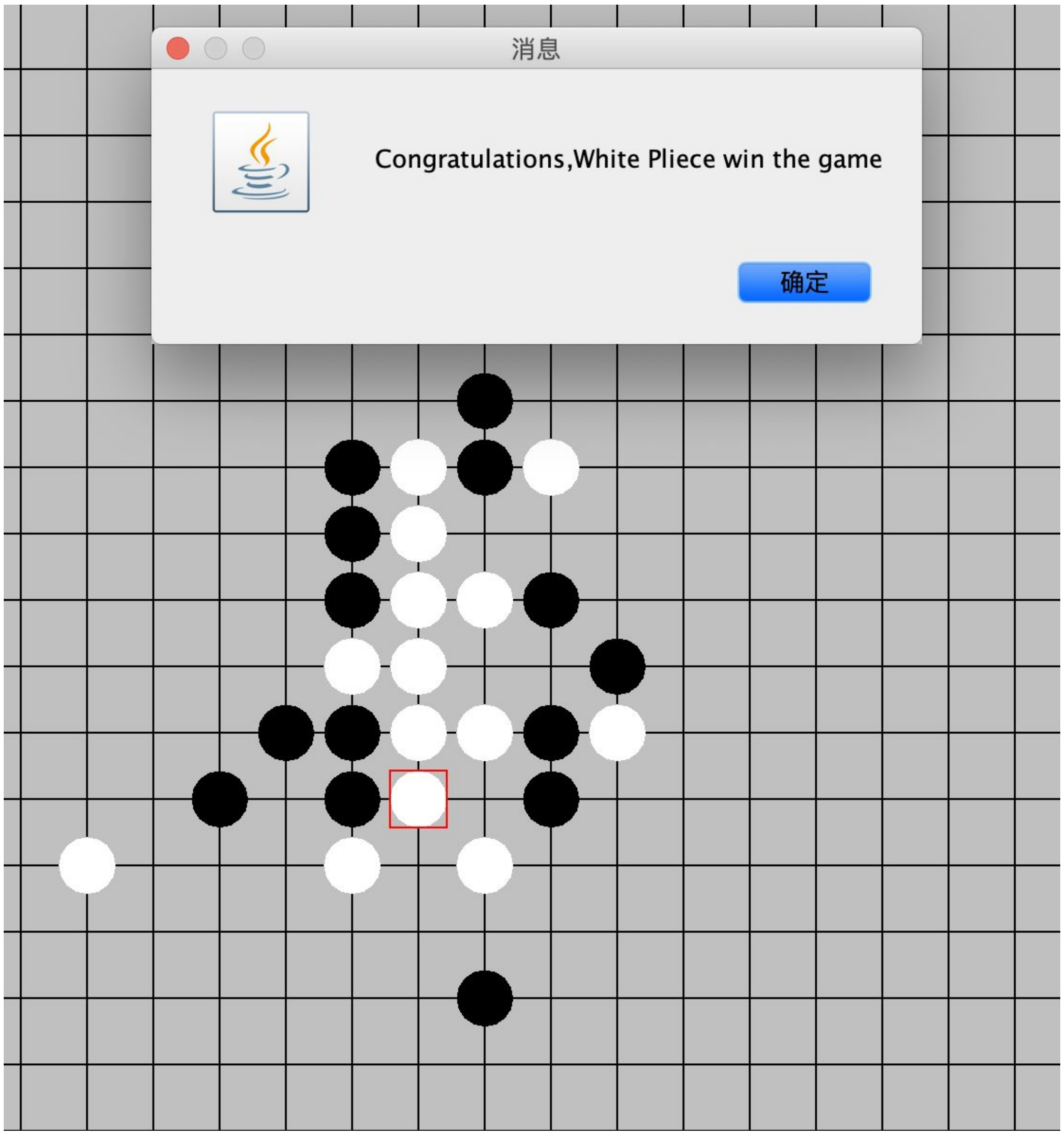
Case 2
Black win (black chess pieces six in rows)



Congratulations,Black Piece win the game

Case 3

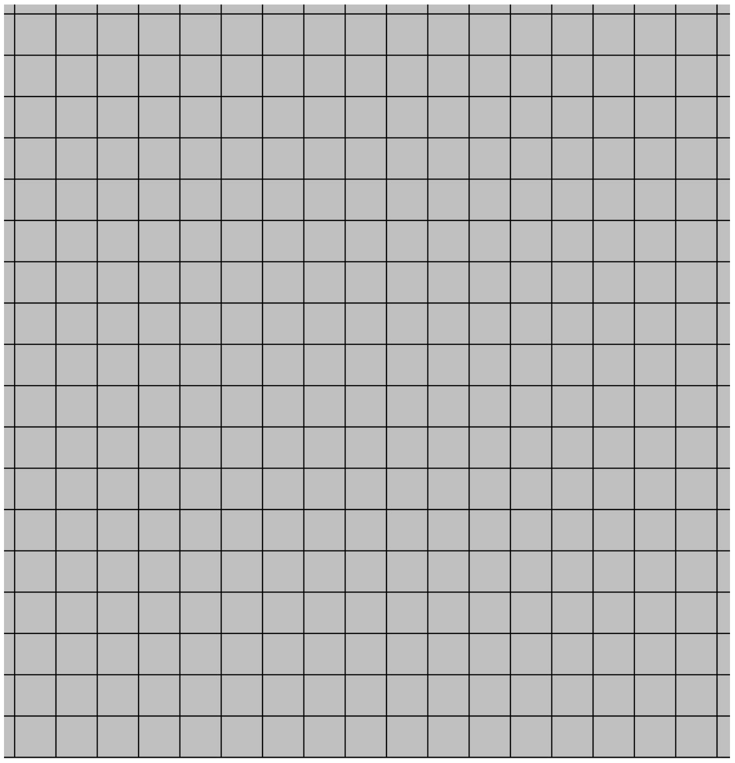
White win (white chess pieces six in rows)



Congratulations,White Pliece win the game

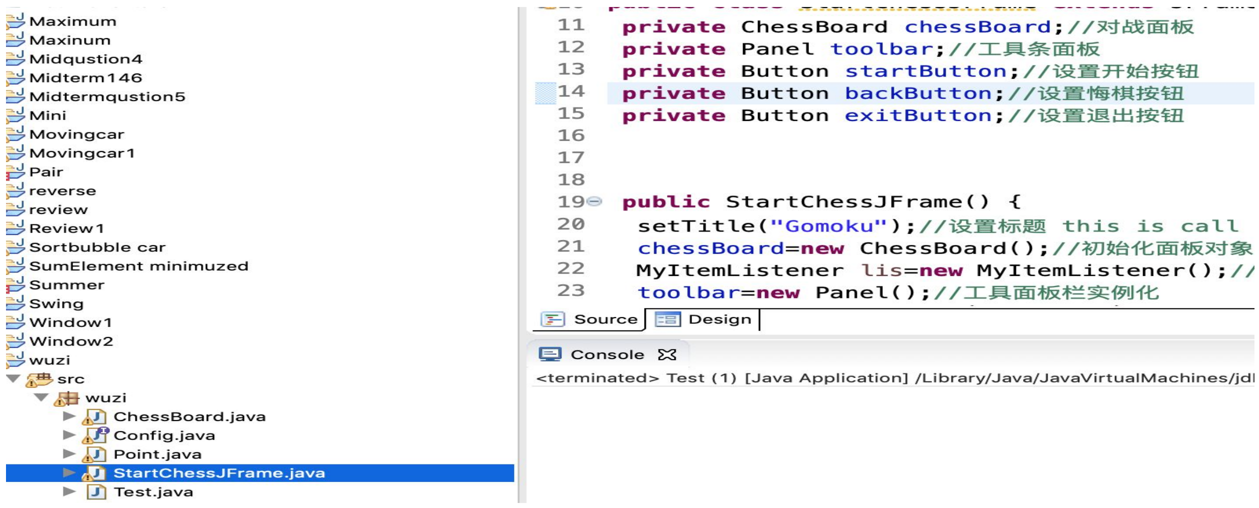
Case 4

restart (chess board will back to original empty board)



Case 5 quit

The Connect Six game JFrame window will close and back to the eclipse app



```
Maximum
Maxinum
Midqustion4
Midterm146
Midtermqustion5
Mini
Movingcar
Movingcar1
Pair
reverse
review
Review1
Sortbubble car
SumElement minimuzed
Summer
Swing
Window1
Window2
wuzi
  src
    wuzi
      ChessBoard.java
      Config.java
      Point.java
      StartChessJFrame.java
      Test.java
```

```
private ChessBoard chessBoard;//对战面板
private Panel toolbar;//工具条面板
private Button startButton;//设置开始按钮
private Button backButton;//设置悔棋按钮
private Button exitButton;//设置退出按钮



public StartChessJFrame() {
 setTitle("Gomoku");//设置标题 this is call
 chessBoard=new ChessBoard();//初始化面板对象
 MyItemListener lis=new MyItemListener();//
 toolbar=new Panel();//工具面板栏实例化
```

Source | Design

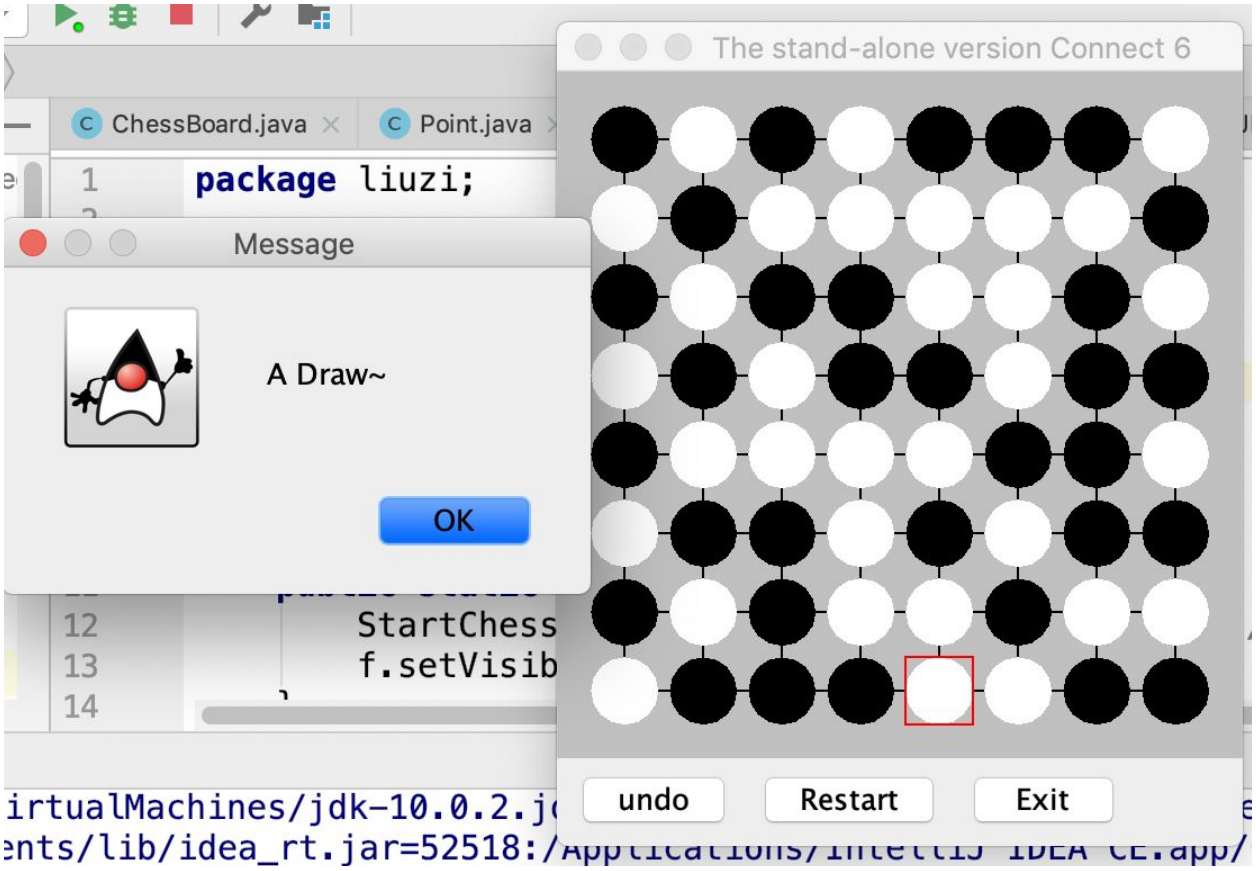
Console

```
<terminated> Test (1) [Java Application] /Library/Java/JavaVirtualMachines/jdl
```

Case 6

Game ended in a draw

In order to test this case faster and easier, we decrease the size of chess board. Below the



```
package liuzi;
    StartChess
    f.setVisib
```

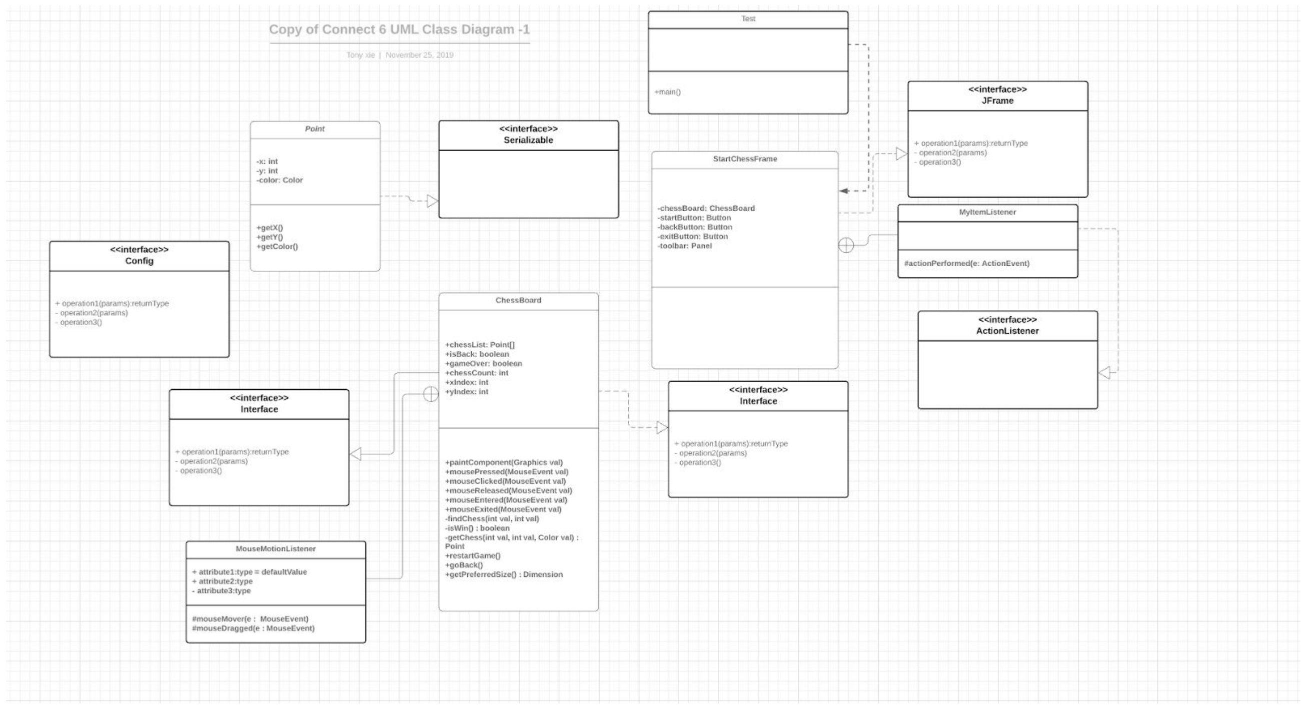
A Draw~

```
irtualMachines/jdk-10.0.2.j
ents/lib/idea_rt.jar=52518:/Applications/IntelliJ IDEA CE.app/
```

1.2 UML Diagrams

We used the LucidChart online UML tool to design and generate out UML diagrams.

1.2.1 Class Diagrams(Draft)



Link from LucidChart:
https://www.lucidchart.com/invitations/accept/a36c0bac-1e87-4f45-bb8e-a9dad48597c6

1.2.2 UML Sequence Diagram
Link from LucidChart:
https://www.lucidchart.com/invitations/accept/76ecbb56-9ab2-4141-9549-6a3cee8af705

1.3 Design pattern applied

1. **Decorator Design Pattern** -attaching additional responsibilities to an object dynamically to extend its functionality

   Most of our project heavily relies on Decorator Design pattern, since we have to make the graphics shown for the game. In our project, we used GUI programming to make a chess board, this is shown in the StartChessJFrame.java class. In our chessboard, we added several toolbars to enhance the functionalities for our game, including the Restart, Back, and Exit button. We also painted our board to be yellow to enhance the visual for the game.

2. **Observer Pattern** - A software design pattern in which an object called the subject, maintains a list of its dependents, called servers, and notifies them automatically of a state changes, usually by calling on of their methods.

   In our project, we demonstrated this pattern in the ChessBoard.java class. In this class, the chess board acts as the subject, that has its dependents, in this case, win or lose which are the servers. When there is a new chess added on the board, the ChessBoard will called the method isWin(), to notify the changes on the board, and checks for which color is the winner according to their color chess counts in a row. To keep track of new chess that are being added, we implemented a MouseListener to notifies and make changes to the program when the mouse is clicked on the chess board.

1.4 components

This assignment is designed to play a GoMoKu(Connect Six)game of writing a basic graphical user interface(GUI) in Java. The key concept are Swing components by using JFrame to create an interface and MouseListener. MouseListener is listener interface for receiving "interesting" mouse events (press, release, click, enter, and exit)on a component. (To track mouse moves and mouse drags, use the MouseMotionListener), MouseEvent for x and y coordinate , graphics-fillOval( ),and exception handling.

Gomoku, also called *Five in a Row*, but we create Connect Six which enhanced the difficulty of game. Gomoku is an abstract strategy board game. It is traditionally played with Go pieces (black and white stones) on a Go board.This game lets two players play Gomoku against each other.Black starts the game .When a player gets Connect Six,that player wins.The game ends in a draw if the board is filled before either played wins.This game defined by five classes : Test.java, StartchessJFrame.java, Point.java, Config.java, Chessboard.java.

Five class

1. Test - main class for creating chessFrame and setvisible.
2. StartChessJFrame - Connect Six JFrame the program startup class (created restart ,back ,exit buttons).
3. Point - the size, color, and x and y coordinates of point.
4. Config- the size of board and space between each row and column.
5. ChessBoard- The main Gomoku board (created board , point, and mouse listener).

1.5 Implementation

ChessBoard.java

```
Packa
ge
liuzi;


        import javax.swing.*;


        import java.awt.*;

        import java.awt.event.MouseListener;

        import java.awt.event.MouseMotionListener;

        import java.awt.event.MouseEvent;

        /*Connect Six*/

        public class ChessBoard extends JPanel implements MouseListener{


         Point[] chessList=new Point[(Config.ROWS+1)*(Config.COLS+1)];//Initialize each array element to
        null

         boolean isBack=true;//Black piece first by default

         boolean gameOver=false;//Whether the game is over

         int chessCount;//Number of pieces on the current board

         int xIndex,yIndex;//The index of the current piece

         public ChessBoard(){

          setBackground(Color.LIGHT_GRAY);//Set the background color to yellow

          addMouseListener(this);//Add event listener

          addMouseMotionListener(new MouseMotionListener() {//Anonymous inner class
  @Override
  public void mouseMoved(MouseEvent e) {
            int x1=(e.getX()-Config.MARGIN+Config.GRID_SPAN/2)/Config.GRID_SPAN;
            int y1=(e.getY()-Config.MARGIN+Config.GRID_SPAN/2)/Config.GRID_SPAN;//Convert
mouse click coordinate position to grid index
            if(x1<0||x1>Config.ROWS||y1<0||y1>Config.COLS||gameOver||findChess(x1,y1)){
   setCursor(new Cursor(Cursor.DEFAULT_CURSOR));//Set to default shape
            }else{
   setCursor(new Cursor(Cursor.HAND_CURSOR));//
            }
  }

  @Override
  public void mouseDragged(MouseEvent e) {
  }
 });
}
/*draw*/
public void paintComponent(Graphics g){
 super.paintComponent(g);//Draw chess board
 for(int i=0;i<=Config.ROWS;i++){//Draw horizontal lines
  g.drawLine(Config.MARGIN, Config.MARGIN+i*Config.GRID_SPAN,
Config.MARGIN+Config.COLS*Config.GRID_SPAN, Config.MARGIN+i*Config.GRID_SPAN);
 }
 for(int i=0;i<=Config.COLS;i++){//Draw a straight line
```

```
   g.drawLine(Config.MARGIN+i*Config.GRID_SPAN, Config.MARGIN,
Config.MARGIN+i*Config.GRID_SPAN,Config.MARGIN+Config.ROWS*Config.GRID_SPAN);

  }

  /*Draw piece*/

  for(int i=0;i<chessCount;i++){

   int xPos=chessList[i].getX()*Config.GRID_SPAN+Config.MARGIN;//X-coordinates of the grid
crossing

   int yPos=chessList[i].getY()*Config.GRID_SPAN+Config.MARGIN;//Y coordinate of grid crossing

   g.setColor(chessList[i].getColor());//Set color

   g.fillOval(xPos-Point.DIAMETER/2, yPos-Point.DIAMETER/2, Point.DIAMETER,
Point.DIAMETER);

   if(i==chessCount-1){

              g.setColor(Color.red);//Mark the last pieces in red

              g.drawRect(xPos-Point.DIAMETER/2, yPos-Point.DIAMETER/2, Point.DIAMETER,
Point.DIAMETER);

   }

  }

 }




 @Override

 public void mousePressed(MouseEvent e) {//Invoked when the mouse button is pressed on a
component

  if(gameOver)//The game is over and cannot be played

   return ;

  String colorName=isBack ? "Black Piece" : "White Piece";
  xIndex=(e.getX()-Config.MARGIN+Config.GRID_SPAN/2)/Config.GRID_SPAN;

  yIndex=(e.getY()-Config.MARGIN+Config.GRID_SPAN/2)/Config.GRID_SPAN;//Convert mouse
click coordinate position to grid index

  if(xIndex<0||xIndex>Config.ROWS||yIndex<0||yIndex>Config.COLS)//pieces fall outside the
chessboard and cannot be played

   return ;

  if(findChess(xIndex,yIndex))//There are already pieces in the x and y positions.

   return ;



  Point ch=new Point(xIndex,yIndex,isBack ? Color.black : Color.white);

  chessList[chessCount++]=ch;

  repaint();//Notify the system to repaint

  if(isWin()){

   String msg=String.format("Congratulations,%s win the game", colorName);

   JOptionPane.showMessageDialog(this, msg);

   gameOver=true;

  }

  else if(chessCount==(Config.COLS+1)*(Config.ROWS+1))

  {

   String msg=String.format("A Draw");

   JOptionPane.showMessageDialog(this,msg);

   gameOver=true;

  }

  isBack=!isBack;

 }
```

```
 @Override

 public void mouseClicked(MouseEvent e) {//Called when the mouse button is clicked (pressed and
released) on a component

 }


 @Override

 public void mouseReleased(MouseEvent e) {////Invoked when the mouse button is released on a
component

 }


 @Override

 public void mouseEntered(MouseEvent e) {//Called when the mouse enters the component

 }


 @Override

 public void mouseExited(MouseEvent e){//Called when the mouse leaves the component

 }


 private boolean findChess(int x,int y){

  for(Point c:chessList){

   if(c!=null&&c.getX()==x&&c.getY()==y)

              return true;

  }

  return false;

 }
```

```
/*Judge which side wins the game*/

private boolean isWin(){

 int continueCount=1;//Number of pieces

 for(int x=xIndex-1;x>=0;x--){//Look left

  Color c=isBack ? Color.black : Color.white;

  if(getChess(x,yIndex,c)!=null){

            continueCount++;

  }else

            break;

 }

 for(int x=xIndex+1;x<=Config.ROWS;x++){//Look right

  Color c=isBack ? Color.black : Color.white;

  if(getChess(x,yIndex,c)!=null){

   continueCount++;

  }else

            break;

 }

 if(continueCount>=6){//Judging that the number of records is greater than or equal to six, that
means the party wins

  return true;

 }else

  continueCount=1;

 //

 for(int y=yIndex-1;y>=0;y--){//look up

  Color c=isBack ? Color.black : Color.white;
  if(getChess(xIndex,y,c)!=null){
   continueCount++;
  }else
            break;
 }
 for(int y=yIndex+1;y<=Config.ROWS;y++){//look down
  Color c=isBack ? Color.black : Color.white;
  if(getChess(xIndex,y,c)!=null){
   continueCount++;
  }else
            break;
 }
 if(continueCount>=6){
  return true;
 }else
  continueCount=1;
 //
 for(int x=xIndex+1,y=yIndex-1;y>=0&&x<=Config.COLS;x++,y--){
  Color c=isBack ? Color.black : Color.white;
  if(getChess(x,y,c)!=null){
            continueCount++;
  }else
            break;
 }
 for(int x=xIndex-1,y=yIndex+1;y<=Config.ROWS&&x>=0;x--,y++){
  Color c=isBack ? Color.black : Color.white;

  if(getChess(x,y,c)!=null){

   continueCount++;

  }else

            break;

 }

 if(continueCount>=6){

  return true;

 }else

  continueCount=1;

 //

 for(int x=xIndex-1,y=yIndex-1;y>=0&&x>=0;x--,y--){

  Color c=isBack ? Color.black : Color.white;

  if(getChess(x,y,c)!=null){

   continueCount++;

  }else

            break;

 }

 for(int x=xIndex+1,y=yIndex+1;y<=Config.ROWS&&x<=Config.COLS;x++,y++){

  Color c=isBack ? Color.black : Color.white;

  if(getChess(x,y,c)!=null){

   continueCount++;

  }else

            break;

 }
```

```
 if(continueCount>=6){

  return true;

 }else

  continueCount=1;

 return false;

}

private Point getChess(int xIndex,int yIndex,Color color){

 for(Point c:chessList){

  if(c!=null&&c.getX()==xIndex&&c.getY()==yIndex&&c.getColor()==color)

   return c;

 }

 return null;

}

public void restartGame(){//Clear all the pieces in the board

 for(int i=0;i<chessList.length;i++)

  chessList[i]=null;

 /*Restore game-related variables*/

 isBack=true;

 gameOver=false;//Whether the game is over

 chessCount=0;//Number of pieces on the current board

 repaint();

}

public void goback(){

 if(chessCount==0)

  return ;
  chessList[chessCount-1]=null;

  chessCount--;

  if(chessCount>0){

   xIndex=chessList[chessCount-1].getX();

   yIndex=chessList[chessCount-1].getY();

  }

  isBack=!isBack;

  repaint();

 }

 //Dimension:rectangle

 public Dimension getPreferredSize(){

  return new Dimension((Config.MARGIN)
*2+Config.GRID_SPAN*Config.COLS,Config.MARGIN*2+Config.GRID_SPAN*Config.ROWS);

 }




}
```

Point.java

```
package
liuzi;

import java.awt.*;

import java.io.Serializable;


public class Point implements Serializable{

 private int x;

 private int y;

 private Color color;//color of the point

 public static int DIAMETER=30;//diameter of the
point

 public Point(int x,int y,Color color){

  this.x=x;

  this.y=y;

  this.color=color;

 }


 public int getX(){

  return x;

 }


 public int getY(){

  return y;

 }


 public Color getColor(){
  return color;
 }
}



package liuzi;

import java.awt.*;
import java.io.Serializable;

/**
 * This class represents the points
 * Author:Group 5
 * CS 151 Final Project
 */
public class Point implements Serializable {

        /**
        *
    */
        private int x;//the x coordinate
        private int y;//the y coordinate
        private Color color;//the color of chess white
or black
        public static int DIAMETER = 30;//the
diameter
```

```
/**

* construct the point

* @param x the value of x coordinate

* @param y the value of y coordinate

* @param color the color of chess

*/

public Point(int x, int y, Color color) {

this.x = x;

this.y = y;

this.color = color;

}


/**

* Get the x coordinate

*

* @return x the value of x coordinate

*/

public int getX() {

 return x;

}



/**

* Get the y coordinate
```

```
     *
     * @return y the value of the vertical
coordinate of chess
     */
     public int getY() {
     return y;
     }

     /**
     * get the color of chess
  * @return the color of chess
     */
     public Color getColor() {
     return color;
     }
}
```

Config.java

```
package
liuzi;

import java.awt.Dimension;

/**
```

```
*This interface stores the four major numbers

*/

public interface Config {

        public static int MARGIN=30;//The value of the marginal distance

         public static int GRID_SPAN=35;//the value of the sepparation distance
between griddings

        int ROWS = 19;  // the number of rows



        int COLS = 19;  //  the number of rows





}
```

StartChsessJFrame.java

```
package liuzi;


import javax.swing.*;

import java.awt.*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;


/*
```

```
 * This class is main frame of Gomuku game

 *It's main functions is to initialize the game

 *CS 151 final project

 * Author: group5

 */


public class StartChessJFrame extends JFrame {


        /**

        * instance variables

        */

        private ChessBoard chessBoard;//the main chess board

        private Panel toolbar;//the tool bar

        private Button startButton;//the start button

        private Button backButton;//set the undo button

        private Button exitButton;//set the exit button


        /**

        * This is the constructor

        */

        public StartChessJFrame() {

    setTitle("The stand-alone version Connect 6");// set the title of this game

    chessBoard = new ChessBoard();//intialize the chess board

        MyItemListener lis = new MyItemListener();//construct and initialize the inner action
listener class
```

```
toolbar = new Panel();//the instantiation of the tool bar
startButton = new Button("Restart");
    backButton = new Button("undo");
exitButton = new Button("Exit");//initialize the three buttons
toolbar.setLayout(new FlowLayout(FlowLayout.LEFT));
toolbar.add(backButton);
toolbar.add(startButton);
toolbar.add(exitButton);//add three buttons into the toolbar
startButton.addActionListener(lis);
backButton.addActionListener(lis);
exitButton.addActionListener(lis);//add each button an ActionListener
add(toolbar, BorderLayout.SOUTH);//use the Border lay out and add the toolbar
add(chessBoard);//add the main chess board to the frame
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);//set the close operation
pack();//self-adapted the size of frame
    }

    /**
    * this inner class is to specifying the actionListener
    */
    private class MyItemListener implements ActionListener {

/**
    * handle the actionEvent
    *
```

```
        * @param e the ActionEvent after clicked certain button

        */

        public void actionPerformed(ActionEvent e) {

        Object obj = e.getSource();//get the source of the actionEvent

        if (obj == startButton) {

         // if the restartButton was clicked

         System.out.println("Restarting...");

         //JFiveFrame.this Inner class references outer class

         chessBoard.restartGame();

         //Restart the whole game

        } else if (obj == exitButton) {

         // the exit button was clicked

         System.out.println(0);// exit the program

         } else if (obj == backButton) {

         //the back button was clicked

         System.out.println("Undo...");//undo the last step

         chessBoard.goback();

         //now it's back to the last step

         }

        }

    }


}
```

Test.java

```
package
liuzi;

                    /**

                    *This clss is for testing the Gomuku game code.

                    *Author Group 5

                    *CS151

                    */



                    public class Test {



                              public static void main(String[] args) {

                                StartChessJFrame f=new StartChessJFrame();//construct the frame
                    of the game

                                f.setVisible(true);//show the frame

                               }

                    }
```

Summary

During the whole process of developing this project, we have learned and applied lots of new things. At the beginning, all the team members were confused about what we can do. We also felt very hard to identify each class's responsibility. It required a large amount of research and reading to understand how to apply the Object-Oriented Design concept. Besides, we found out that communication is very important for the success of developing. Obviously, our project only satisfies the basic functions. It still needs to improve on many aspects such as the looking. It's a very interesting process for developing a game that you are familiar with.