4인용 **할리갈리** 게임

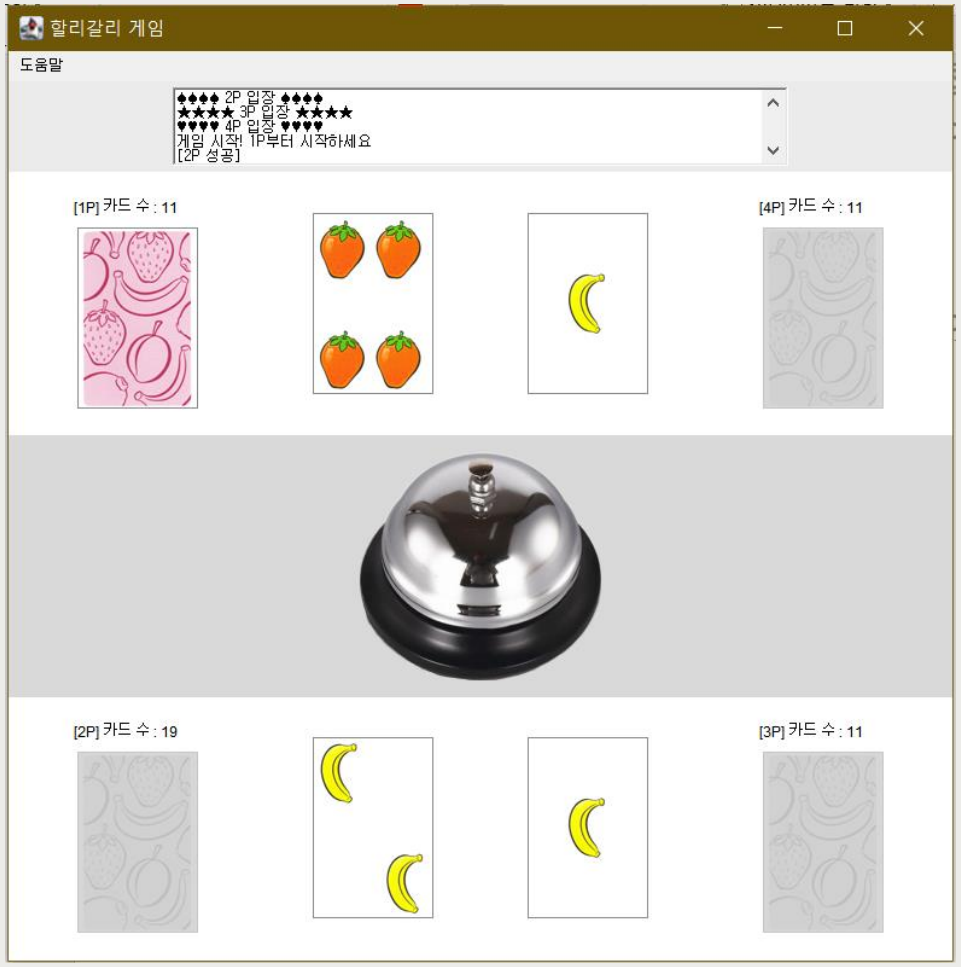
비트캠프 1차 JAVA 프로젝트

소켓통신을 활용한 게임

개발기간 : 7일

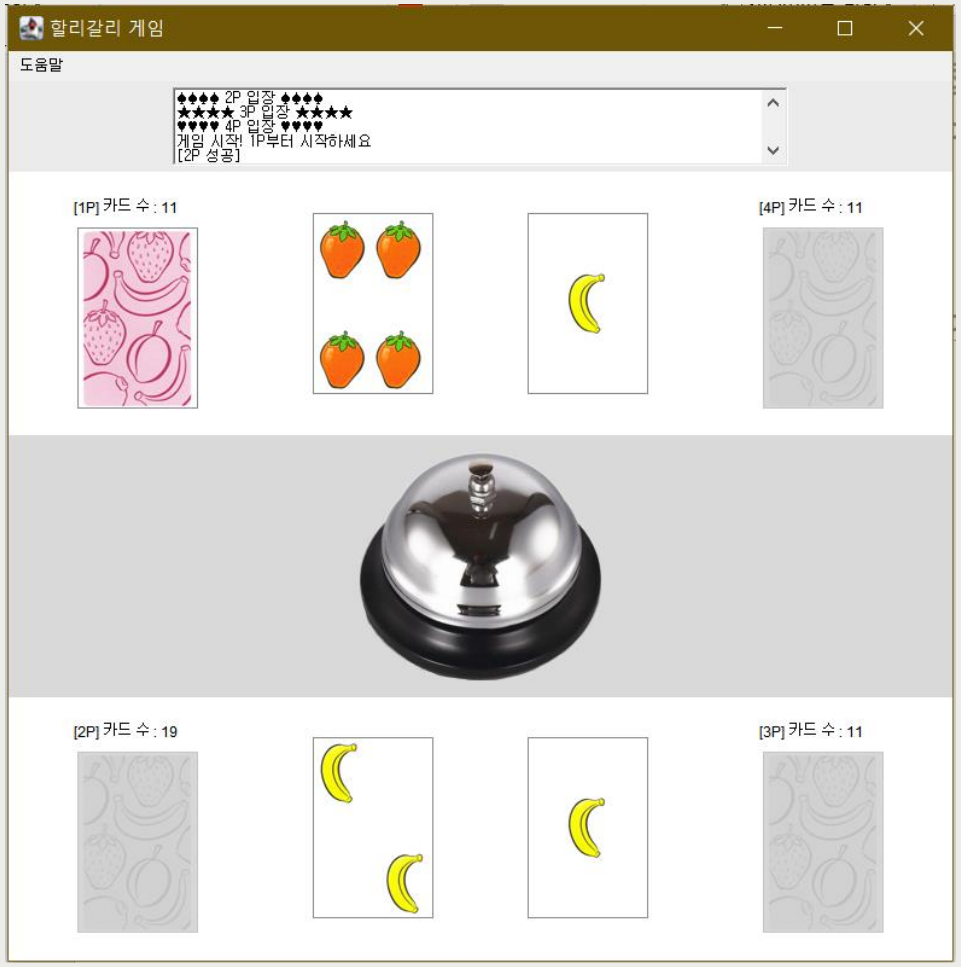
전형진, 김민주, 김재연

할리갈리 UI



- Java awt와 swing 사용
- 카드 이미지
- 공지사항 텍스트박스
- 남은 카드 수 표시
- 도움말 메뉴 : 게임설명

할리갈리 규칙



- 1p~4p 순서대로 돌아가면서 카드 오픈한다.
- 오픈 된 카드 중에서 같은 종류 과일 5개가 되면 종을 친다.
- 먼저 친사람이 오픈된 카드를 모두 가져간다.
- 종을 잘못 치면 다른 플레이어들에게 카드를 1장씩 나눠준다.
- 카드를 전부 잃으면 패배한다.
- 마지막까지 살아남으면 승리한다.

서버 : 소켓통신

```
public synchronized void startThread() {
    ServerSocket svSocket = null;

    try {
        // System.out.println("server open");
        svSocket = new ServerSocket(PORT, 4);

        while (true) {// infinity loop

            System.out.println("waiting for accessing");
            Socket socket = svSocket.accept();
            // System.out.println(socket.getInetAddress() + "님이 입장하셨습니다. ");

            TestThread th = new TestThread(socket, position);
            th.start(); // player thread

            playerSu.put(cnt + "key", socket);
            cnt++;// 0123 키값.
            //들어오는 순서대로 "0key", "1key" , "2key", "3key"부여, 소켓저장

            // System.out.println("player : " + playersu.size());
            if (playerSu.size() == 4) {
                System.out.println("server is full");
            }

            if ((playerSu.size() - 1) == position) {
                position++;
            }
        } // while
    } catch (IOException e) {
        position = 0;
        e.printStackTrace();
    }

}// startThread
```

- Port와 접속인원 수 4
- New ServerSocket 생성
- 클라이언트가 accept 되면 thread 생성한다.

서버 : 카드 준비

```
static final int PORT = 8111;
static int cnt = 0;
static int position = 0;
static HashMap<String, Socket> playerSu = new HashMap◇();
static ArrayList<BufferedWriter> playerStream = new ArrayList◇();
static int numberOfPlayers = 4;
static int[][] allCard = new int[4][14];    //카드번호 생성
static String[] card2;                      //allCard를 과일종류 0123 붙여서 String타입으로 변환
static String[] card3 = new String[56];     //card2를 랜덤으로 섞기 전에 정렬된 상태로 복사.
Vector<String> player01 = new Vector<String>();//플레이어가 처음 받을 카드
Vector<String> player02 = new Vector<String>();
Vector<String> player03 = new Vector<String>();
Vector<String> player04 = new Vector<String>();
static ArrayList<String> openedCards = new ArrayList◇();//오픈된 4장의 카드
static Vector<String> allOpenedCards = new Vector◇();//플레이어가 가져가기 전까지 쌓이는 오픈된 카드
```

- 필요한 변수와 자료구조를 미리 선언해둔다.
- Player는 게임유저가 가질 카드 자료구조이다.
- 카드를 셔플해서 vector에 미리 담는다.

서버 : 카드 준비

```
public void cardShuffle(int[][] arr, int count) {
    String[][] card = new String[4][14];
    card2 = new String[56];
    int card2Length = 0;
    for (int i = 0; i < 4; i++) {// 인자앞에 과일종류붙여서 문자열변환, 배열복사.
        for (int j = 0; j < 14; j++) {
            card[i][j] = i + "" + arr[i][j];
        }
    } // outforend

    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 14; j++) {
            card2[card2Length] = card[i][j];
            card2Length++;
        }
    } // outforend
    System.arraycopy(card2, 0, card3, 0, 56);// 랜덤으로섞기전에 배열복사.
    // 클라이언트 과일이미지랑 비교용

    for (int k = 0; k < count; k++) {
        int i = (int) (Math.random() * card2.length);
        int j = (int) (Math.random() * card2.length);
        String tmp = card2[i];
        card2[i] = card2[j];
        card2[j] = tmp;
    }
}// cardShuffle
```

- 필요한 변수와 자료구조를 미리 선언해둔다.
- Player는 게임유저가 가질 카드 자료구조이다.
- 카드를 셔플해서 vector에 미리 담는다.

```java
public void cardShuffle(Vector<String> arr, int count) {
    String[] temp = new String[arr.size()];
    for (int i = 0; i < arr.size(); i++) {
        temp[i] = arr.get(i);
    }
    for (int k = 0; k < count; k++) {
        int i = (int) (Math.random() * temp.length);
        int j = (int) (Math.random() * temp.length);
        String tmp = temp[i];
        temp[i] = temp[j];
        temp[j] = tmp;
    }
    arr.removeAllElements();
    for (int i = 0; i < temp.length; i++) {
        arr.add(temp[i]);
    }
}// cardShuffle

public void cardDivide() {

    String[] player1 = new String[14];
    String[] player2 = new String[14];
    String[] player3 = new String[14];
    String[] player4 = new String[14];

    int idx = 0;
    for (int i = 0; i < 14; i++) {
        player1[idx] = card2[i];
        player2[idx] = card2[i + 14];
        player3[idx] = card2[i + 28];
        player4[idx] = card2[i + 42];
        idx++;
    }

    for (int j = 0; j < 14; j++) {
        player01.add(player1[j]);
        player02.add(player2[j]);
        player03.add(player3[j]);
        player04.add(player4[j]);
    }

}// cardDivide
```

- 생성한 카드를 Math.random()을 이용해 배열에 섞어서 담는다.
- 섞은 카드배열을 플레이어가 가질 배열에 나눠담는다.
- 이미 섞여 있으므로 i번째부터 14장씩 index번호만 늘려서 차례대로 담아준다.
- 분배가 완료된 뒤 플레이어 배열을 vector에 다시 옮겨 담는다.
- 카드 준비를 마친다.

서버 : 게임 시작

- 플레이어의 클릭이벤트로 문자열을 inputStream으로 받는다.
- 1카드오픈~4카드오픈
- 서버에 있는 플레이어 카드 vector 상태를 확인해 게임 오버인지 아닌지 판별한다.
- 플레이어 카드에서 꺼낼 카드가 없어 indexOutOfBound 예외가 발생하면 패배처리 한다.

```
String msg = null;

while ((msg = br.readLine()) ≠ null) {
    synchronized (msg) {

        System.out.println(socket.getInetAddress());
        System.out.println("쌓인 카드 수 " + allOpenedCards.size());
        OutputStreamWriter bellosw = null;
        BufferedWriter bellbfw = null;
        if (msg.equals("1카드오픈")) {
            System.out.println("1p카드수 " + player01.size());
            try {
                String card = player01.get(0);
                openedCards(card);
                player01.remove(0);

                for (int i = 0; i < playerStream.size(); i++) {
                    bw = playerStream.get(i);
                    bw.write(findCardIndex(card) + "1카드오픈"
                            + player01.size());
                    bw.newLine();
                    bw.flush();
                } // forend
            } catch (IndexOutOfBoundsException e) {

                bellosw = new OutputStreamWriter(playerSu.get(
                        "0key").getOutputStream());
                bellbfw = new BufferedWriter(bellosw);
                bellbfw.write("패배1카드오픈");
                bellbfw.newLine();
                bellbfw.flush();
                playerStream.remove(bellbfw);

                for (int i = 0; i < playerStream.size(); i++) {
                    bw = playerStream.get(i);
                    bw.write("[1P 패배] 1P의 남은 카드가 없습니다.");
                    bw.newLine();
                    bw.flush();
                } // forend
                playerSu.remove("0key");
                numberOfPlayers--;
            } // catch
```

서버 : 게임 종료

- 카드 size()=0인 플레이어는 제거한다.
- 플레이어 카드 vector들의 사이즈를 구해 정렬한다.
- 1명 남은게 누구인지 win[]에 카드 수와 일치하는 플레이어를 찾는다.
- 1명 남았을 때 텍스트박스에 승리자 메시지를 출력한다.

```
    try {
        if (player01.size() == 0) {
            playerSu.remove("0key");
        }
        if (player02.size() == 0) {
            playerSu.remove("1key");
        }
        if (player03.size() == 0) {
            playerSu.remove("2key");
        }
        if (player04.size() == 0) {
            playerSu.remove("3key");
        }
    } catch (Exception e) {
    }
```

```
int winner = 0;
int p1 = player01.size();// 0
int p2 = player02.size();// 0
int p3 = player03.size();// 26
int p4 = player04.size();// 0

int[] win = { p1, p2, p3, p4 };
Arrays.sort(win);
System.out.println(Arrays.toString(win));

winner = win[3];

if (win[3] == player01.size()) {
    winner = 1;
} else if (win[3] == player02.size()) {
    winner = 2;
} else if (win[3] == player03.size()) {
    winner = 3;
} else if (win[3] == player04.size()) {
    winner = 4;
}
```

```
if (playerSu.size() == 1) {

    System.out.println("체크44444444");
    for (int i = 0; i < playerStream.size(); i++) {
        bw = playerStream.get(i);
        bw.write("***** [ " + winner + "P ] 승리 *****");
        bw.newLine();
        bw.flush();
    }
    break;

}// if
```

클라이언트 : 카드 준비

```
/*
 * 00 01 02 03 04 05 06 07 08 09 010 011 012 013 바나나
 * 10 11 12 13 14 15 16 17 18 19 110 111 112 113 레몬
 * 20 21 22 23 24 25 26 27 28 29 210 211 212 213 자두
 * 30 31 32 33 34 35 36 37 38 39 310 311 312 313 딸기
 *
 *
 * */

public void cardImages() {

    String imageFile = "";
    ImageIcon icon = null;
    Image img = null;
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 14; j++) {
            imageFile = ".\\cardimages2\\" + i + "" + j + ".jpg";
            img = tool.createImage(imageFile);
            icon = new ImageIcon(img);
            cardImages.add(icon);// 0~55
        }
    } // for
}// cardImages
```

- 카드 이미지 이름을 규칙성 있게 숫자로 정하고 숫자에 해당하는 카드 이미지를 불러와서 자료구조에 담는다.
- 내부적으로는 숫자로 카드를 구별한다.
- 누가 카드를 얼마나 가지고 있는지는 서버가 처리한다.

클라이언트 : 게임 시작

- Thread를 시작하면서 HaliUI 생성자를 생성한다. (게임창 활성화)
- 서버에서 전체 플레이어 수를 받으면서 입장할 때 자신이 몇번 Player인지 텍스트 박스에서 확인 가능하다.
- 4p까지 입장하면 자동으로 게임이 시작되며 1p부터 카드 오픈가능하다.

```
@Override
public void run() {
    // System.out.println("thread start");

    try {
        isr = new InputStreamReader(socket.getInputStream());
        br = new BufferedReader(isr);
        osw = new OutputStreamWriter(socket.getOutputStream());
        bw = new BufferedWriter(osw);

        int position = Integer.parseInt(br.readLine());

        Haliui haliui = new Haliui(socket, position, true);
        bw.write("대기");
        bw.newLine();
        bw.flush();
        String numberOfPlayers = null;

        while ((numberOfPlayers = br.readLine()) ≠ null) {

            if (numberOfPlayers.equals("4")) {
                ta.append("♥♥♥♥ 4P 입장 ♥♥♥♥\n");
                ta.append("게임 시작! 1P부터 시작하세요\n");
                break;
            } else if (numberOfPlayers.equals("3")) {
                ta.append("★★★★ 3P 입장 ★★★★\n");
            } else if (numberOfPlayers.equals("2")) {
                ta.append("♠♠♠♠ 2P 입장 ♠♠♠♠\n");
            } else if (numberOfPlayers.equals("1")) {
                ta.append("◆◆◆◆ 1P 입장 ◆◆◆◆\n");
            } else {
                ta.append("PLAYER 를 기다리는 중입니다 ... \n");
            }
            ta.revalidate();
        }
        bw.write("시작");
        bw.newLine();
```

클라이언트 : 카드 준비

- Position 0~3 = 1p~4p의 자리이다.
- Position별로 1p~4p가 없을 시 다음 차례의 플레이어의 카드를 활성화 시켜준다.
- 텍스트박스에서 몇 번 P패배 문자열을 검색해서 패배한 플레이어를 인식한다.
- 패배한 플레이어의 카드는 비활성화되어 클릭이 되지 않는다.
- 1p부터 차례대로 오픈하며 자기 차례에서만 카드버튼이 활성화 된다.

```
String msg = null;
while ((msg = br.readLine()) ≠ null) {

    if (position == 0) {

        if (!ta.getText().contains("1P 패배")
                && msg.contains("4카드오픈")) {
            jbtns[0].setEnabled(true);
        } else if (ta.getText().contains("4P 패배")
                && msg.contains("3카드오픈")) {
            jbtns[0].setEnabled(true);
        } else if (ta.getText().contains("3P 패배")
                && ta.getText().contains("4P 패배")
                && msg.contains("2카드오픈")) {
            jbtns[0].setEnabled(true);
        }

        if (ta.getText().contains("1P 패배")) {
            jbtns[4].setEnabled(false);
            jla1.setEnabled(false);
            la6.setEnabled(false);
            break;
        }

    } else if (position == 1) {// 2p

        if (!ta.getText().contains("2P 패배")
                && msg.contains("1카드오픈")) {
            jbtns[2].setEnabled(true);
        } else if (ta.getText().contains("1P 패배")
                && msg.contains("4카드오픈")) {
            jbtns[2].setEnabled(true);
        } else if (ta.getText().contains("1P 패배")
                && ta.getText().contains("4P 패배")
                && msg.contains("3카드오픈")) {
            jbtns[2].setEnabled(true);
        }
        if (ta.getText().contains("2P 패배")) {
            jbtns[4].setEnabled(false);
            jla3.setEnabled(false);
            la5.setEnabled(false);
            break;
```

클라이언트 : 카드 준비

- 1p가 카드 버튼을 누르면 카드가 오픈되고 서버에 1카드오픈 문자열이 전송된다.
- 그리고 다시 클라이언트에게도 같은 문자열이 오며 누가 카드를 오픈했는지 인식한다.
- 서버에서 남은 카드 수와 종을 눌렀을 때 성공인지 실패인지 등의 결과를 받아 클라이언트가 화면에 나타낸다.

```
if (msg.contains("1카드오픈")) {
    jbtns[0].setEnabled(false);
    String left = msg.substring(msg.lastIndexOf("픈") + 1);
    System.out.println(left);
    la6.setText("    [1P] 카드 수 : " + left);
    la6.setBackground(color5);
    Thread.sleep(70);
    la6.setBackground(null);

    if (msg.contains("패배") || left.equals("0")) {
        ta.append("[1P 패배]\n");
        jla1.setIcon(new ImageIcon("bincard.jpg"));
        jbtns[0].setEnabled(false);
        // jbtns[4].setEnabled(false);
        jla1.setEnabled(false);
        la6.setVisible(false);
    } else {
        System.out.println(msg.substring(0, msg.indexOf("1카")));
        String cardImage = msg.substring(0, msg.indexOf("1카"));
        int idx = Integer.parseInt(cardImage);
        jla1.setIcon(cardImages.get(idx));
        // p3.add(jla1, haliui.gridcons(1, 0));
    }
} else if (msg.contains("4카드오픈")) {
    jbtns[1].setEnabled(false);
    String left = msg.substring(msg.lastIndexOf("픈") + 1);
    System.out.println(left);
    la7.setText("    [4P] 카드 수 : " + left);
    la7.setBackground(color5);
    Thread.sleep(70);
    la7.setBackground(null);

    if (msg.contains("패배") || left.equals("0")) {
        ta.append("[4P 패배]\n");
        jbtns[1].setEnabled(false);
        // jbtns[4].setEnabled(false);
        jla2.setEnabled(false);
        la7.setVisible(false);
    } else {
        String cardImage = msg.substring(0, msg.indexOf("4카"));
        int idx = Integer.parseInt(cardImage);
        jla2.setIcon(cardImages.get(idx));
        // p3.add(jla2, haliui.gridcons(3, 0));
```

소감

소켓통신보다 게임 규칙을 자바로 표현하는 것이 어려웠다. 약속된 문자열을 게임 UI의 클릭이벤트로 서버로 보내고 서버는 이를 이용해서 게임을 처리하고 플레이어들의 카드 정보를 갱신한다. 다시 서버는 클라이언트로 약속된 문자열을 전달하고 클라이언트도 받은 문자열로 화면처리를 한다. 복잡한 게임이지만 기본은 채팅과 다를 게 없다는 것을 느꼈고 소켓통신, 쓰레드, 자료구조, 자바, 스트림 등 배운 것들을 종합적으로 활용할 수 있었다.