



HEAD SOCCER

Matilde Simonini & Alessandro Taboni

INTRODUZIONE

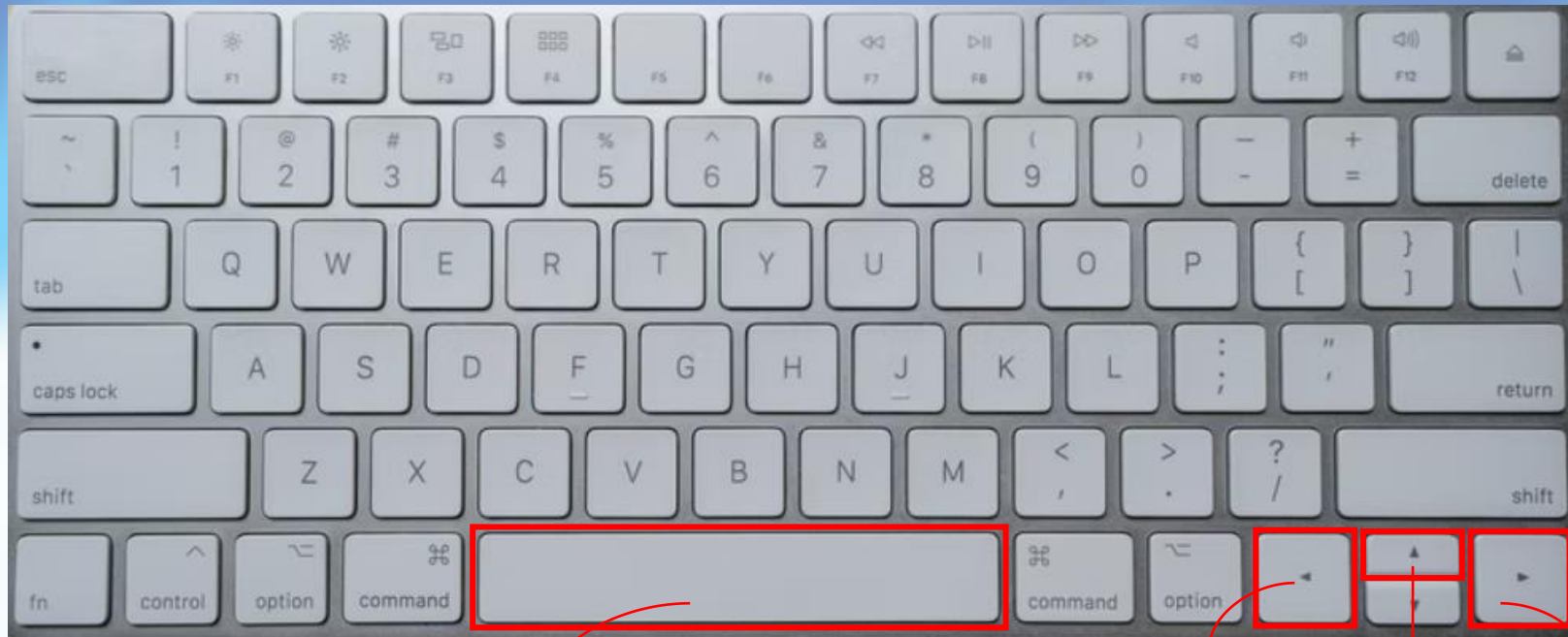
Head Soccer è un gioco di calcio multiplayer che coinvolge due giocatori che si sfidano in una partita di calcio al meglio dei 3 goal.



REGOLAMENTO

- Numero di giocatori: 2
- I giocatori possono: muoversi a destra e a sinistra, saltare, calciare.
- Ogni volta che un giocatore fa goal, i giocatori e la palla ripartono dalle posizioni iniziali.
- Ogni volta che si verifica un fuori campo (palla ferma sopra la traversa) i giocatori e la palla ripartono dalle posizioni iniziali.
- Vince il giocatore che raggiunge per primo i 3 goal.

COMANDI



KICK

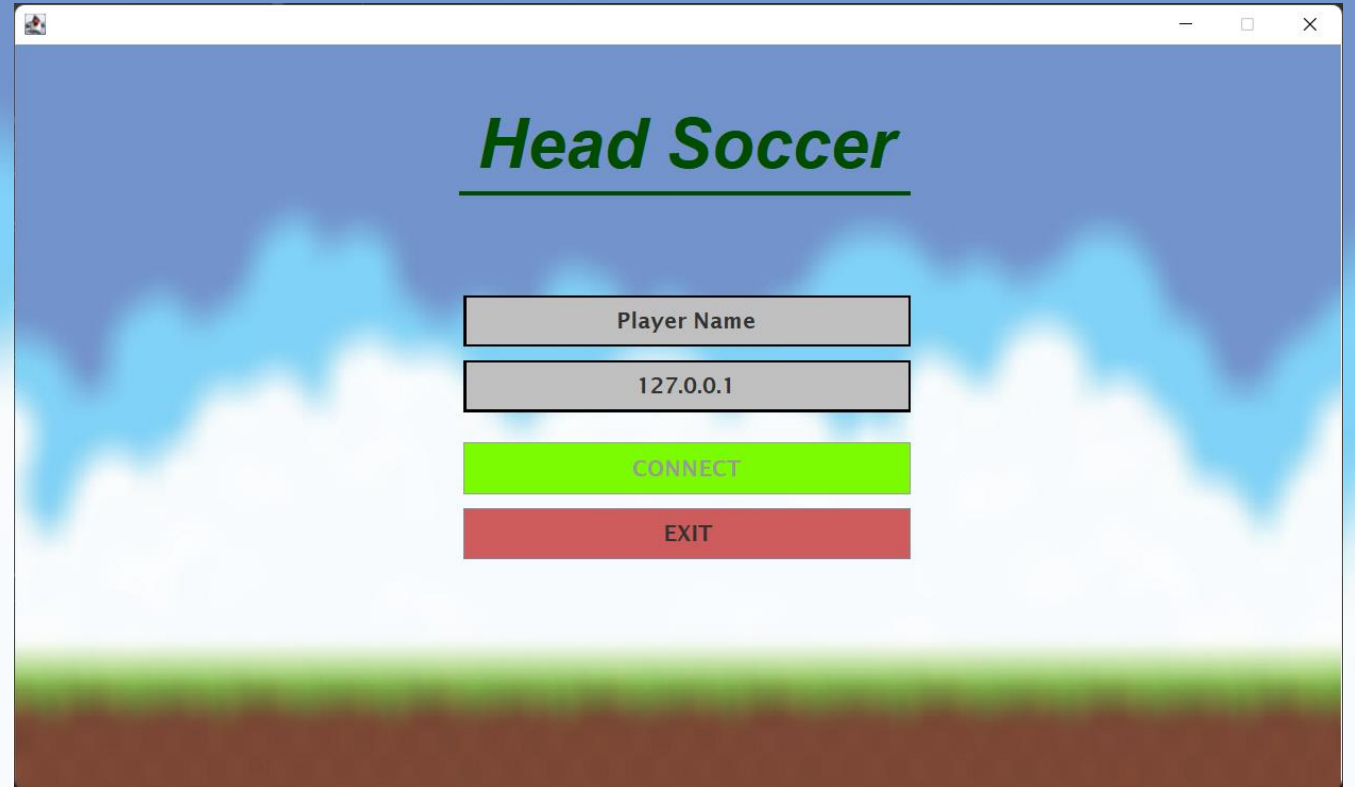
MOVE LEFT

JUMP

MOVE RIGHT

AVVIO DEL GIOCO

All'avvio del gioco viene chiesto di inserire il nome del player e l'indirizzo IP del server al quale collegarsi.



Head Soccer

Player Name

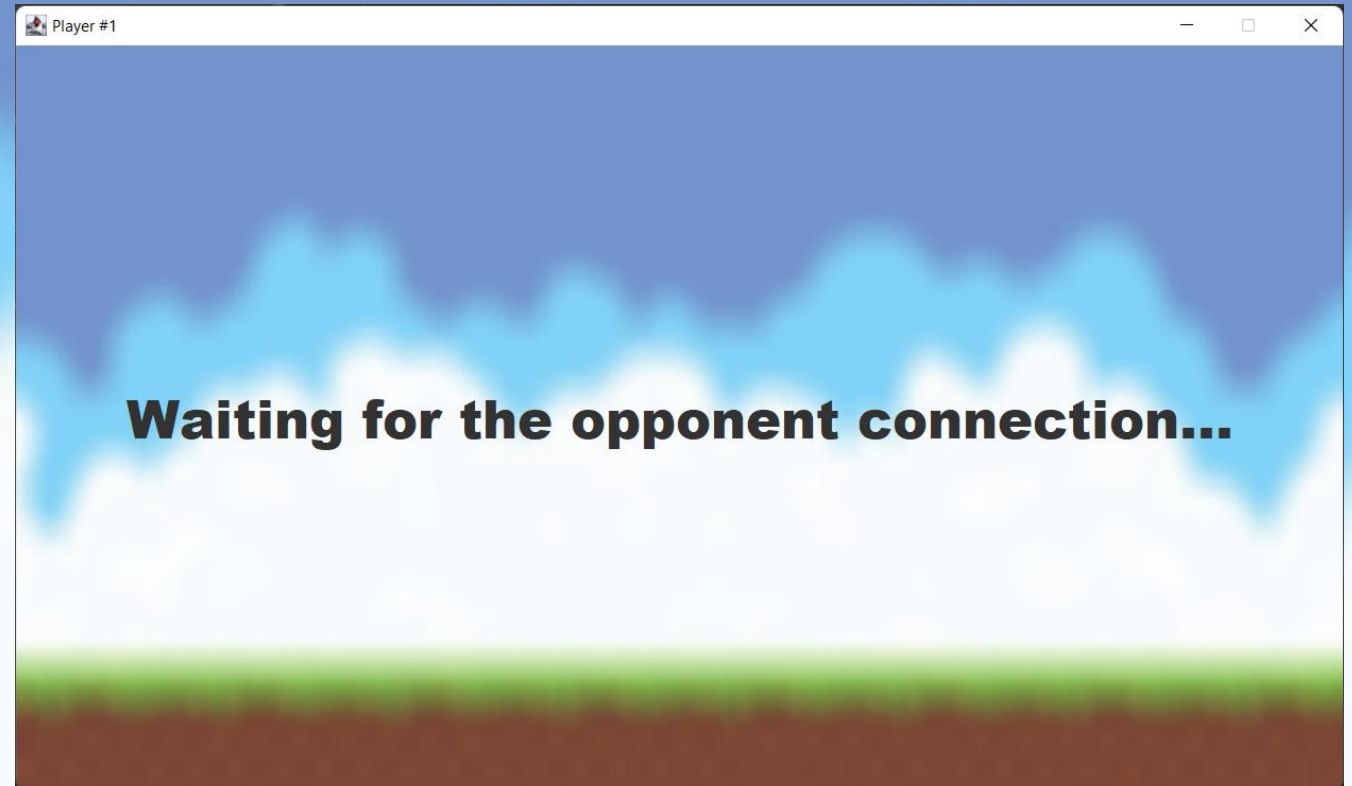
127.0.0.1

CONNECT

EXIT

SALA D'ATTESA

Una volta premuto il tasto «connect» si entra nella schermata di attesa nella quale si rimane fino all'arrivo del secondo giocatore.



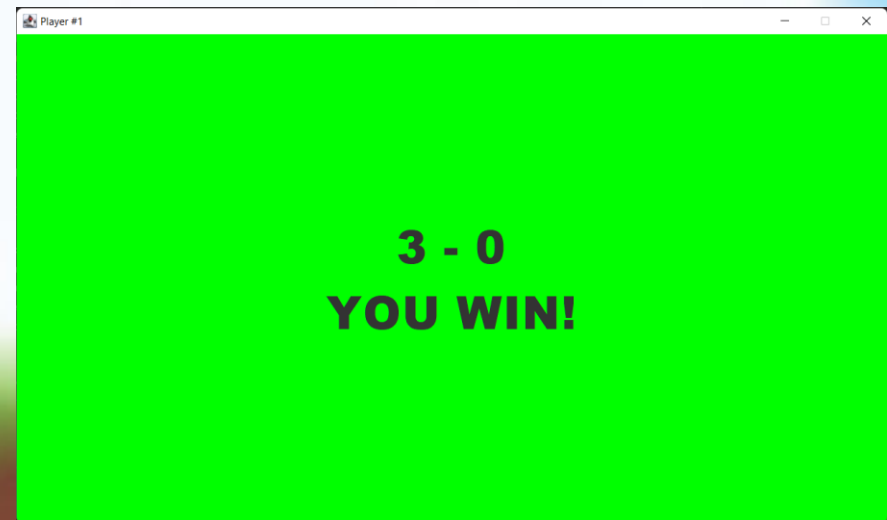
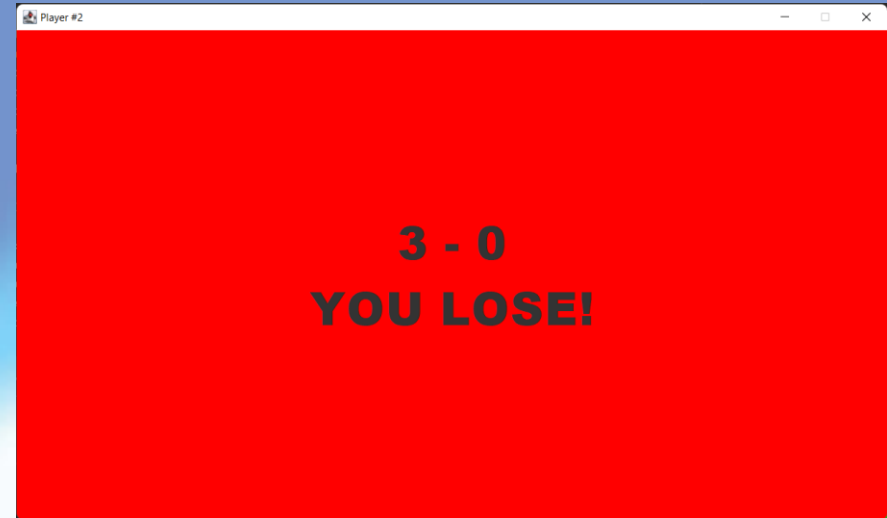
SCHERMATA DI GIOCO

Una volta che il secondo giocatore si connette, si entra nella schermata di gioco (ora i giocatori possono comandare i propri personaggi).



FINE PARTITA

Appena uno dei due giocatori segna il terzo goal (vittoria), il gioco termina e viene mostrato il risultato.

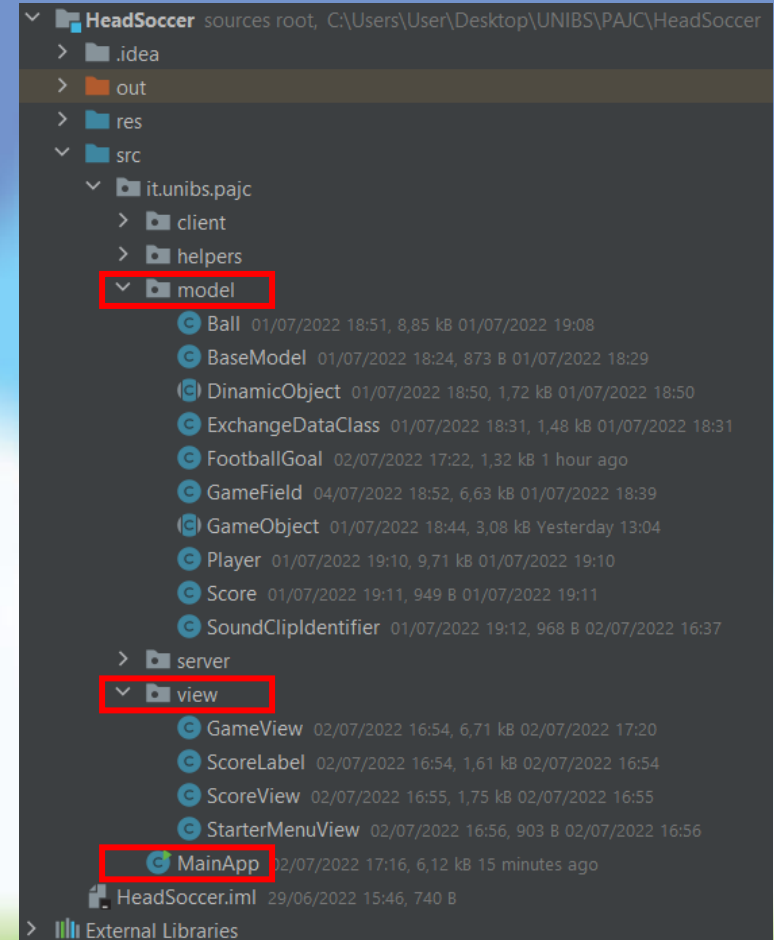


REQUISITI PROGETTO

- Implementazione pattern MVC
- Client / Server
- GUI (Graphic User Interface)
- Gestione delle collisioni
- Gestione dei suoni

MVC PATTERN

Abbiamo implementato il pattern MVC (Model View Controller) attraverso la suddivisione delle classi del progetto in package diversi.



CLIENT / SERVER

```
public Client(MainApp frame, String serverIp) {
    this.frame = frame;
    this.serverIp = serverIp;
}

/**
 * Metodo che tenta di avviare la connessione al server di ip fornito dall'utente nel costruttore della classe Client
 */
public void startServerConnection() {
    try {
        serverConnection = new Socket();
        serverConnection.connect(new InetSocketAddress(serverIp, Server.PORT), timeout: 1000);

        HelperClass.importImages(); //si scaricano le immagini necessarie

        ObjectOutputStream out = new ObjectOutputStream(serverConnection.getOutputStream());
        ObjectInputStream in = new ObjectInputStream(serverConnection.getInputStream());

        playerID = Integer.parseInt((String) in.readUnshared()); //come prima cosa riceviamo dal server l'ID del player
        System.out.println("You are player #" + playerID);

        if(playerID == 1){
            System.out.println("Waiting for Player #2 to connect...");
        }

        //creazione classi di scrittura e lettura
        readerFS = new ReadFromServer(in);
        writerTS = new WriteToServer(out);
    }
}
```

```
public Server(){
    System.out.println("=====GAME SERVER =====");

    try {
        serverSocket = new ServerSocket(PORT);

        gameField = new GameField();

        commandApplierPl1 = new CommandApplier(gameField.getPlayer1());
        commandApplierPl2 = new CommandApplier(gameField.getPlayer2());

    }catch (IOException e){
        System.out.println(e.getMessage());
    }
}

//metodo che avvia il server e permette di accettare le connessioni con i client
public void startServer() {
    try {

        while (numPlayers < maxPlayers) {
            Socket client = serverSocket.accept();

            ObjectOutputStream out = new ObjectOutputStream(client.getOutputStream());
            ObjectInputStream in = new ObjectInputStream(client.getInputStream());

            numPlayers++;
            out.writeUnshared( obj: "" + numPlayers); //Si manda al player il suo ID ---
        }
    }
}
```

GUI

```
//DISEGNO PALLA
g2.drawImage(HelperClass.getImageFromName( nameImage: "Ball.png"), (int) modelData.getBall().getPosX(), (int) modelData.getBall().getPosY(), observer: null);

//DISEGNO PERSONAGGI
BufferedImage leftPlayerImage = null;
switch (modelData.getPlayer1().getCurrentImageIndex()) {
    case 0 -> leftPlayerImage = HelperClass.getImageFromName( nameImage: "LeftMan.png");
    case 1 -> leftPlayerImage = HelperClass.getImageFromName( nameImage: "WalkingLeftMan.png");
    case 2 -> leftPlayerImage = HelperClass.getImageFromName( nameImage: "KickLeftMan.png");
}

g2.drawImage(leftPlayerImage, (int) modelData.getPlayer1().getPosX(), (int) modelData.getPlayer1().getPosY(), observer: null);

BufferedImage rightPlayerImage = null;
switch (modelData.getPlayer2().getCurrentImageIndex()) {
    case 0 -> rightPlayerImage = HelperClass.getImageFromName( nameImage: "RightMan.png");
    case 1 -> rightPlayerImage = HelperClass.getImageFromName( nameImage: "WalkingRightMan.png");
    case 2 -> rightPlayerImage = HelperClass.getImageFromName( nameImage: "KickRightMan.png");
}

if(!modelData.getPlayer2().getKickStatus()) {
    g2.drawImage(rightPlayerImage, (int) modelData.getPlayer2().getPosX(), (int) modelData.getPlayer2().getPosY(), observer: null);
}
else {
    g2.drawImage(rightPlayerImage, x: (int) modelData.getPlayer2().getPosX() - 12, (int) modelData.getPlayer2().getPosY(), observer: null);
}
```

```
public void draw(Graphics2D g) {
    g.setColor(Color.WHITE);
    g.fillRect(x, y, width, height);

    namePl1.draw(g);
    namePl2.draw(g);
    scorePl1.draw(g);
    scorePl2.draw(g);
}

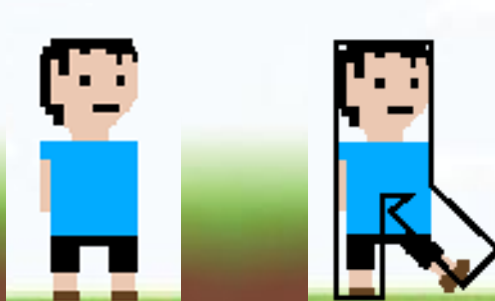
public void setScore(Score score) {
    if(score != null) {
        if(!score.getPlayer1().getPlayerName().equals(namePl1.getText())) {
            namePl1.setText(score.getPlayer1().getPlayerName());
            namePl2.setText(score.getPlayer2().getPlayerName());
        }

        scorePl1.setText(""+score.getScorePl1());
        scorePl2.setText(""+score.getScorePl2());
    }
}
```

COLLISION DETECTION

Sviluppo della gestione delle collisioni attraverso i seguenti punti:

- Ogni oggetto di gioco è caratterizzato da una Shape (non visibile a schermo) che ne costituisce lo scheletro
- Ad ogni update del gioco vengono controllate le eventuali intersezioni tra le Shape degli oggetti
- `collisionDetected()` specifico per ogni oggetto
- Modellizzazione fisica del gioco (forza di gravità, attrito, urti)
- Supponiamo che i giocatori abbiano massa molto maggiore rispetto alla palla per semplicità



GESTIONE SUONI

```
Sound.java
1 package it.unibs.pajc.helpers;
2
3 import ...
4
5 /**
6  * Classe necessaria per gestire i suoni nel gioco
7  */
8
9 public class Sound {
10
11     Clip clip;
12     URL soundUrl[] = new URL[30];
13     public static final int KICK_OFF = 0;
14     public static final int KICK_BALL = 1;
15
16     public Sound() {
17         soundUrl[0] = getClass().getResource(name: "/sounds/kick-off.wav");
18         soundUrl[1] = getClass().getResource(name: "/sounds/kick-ball.wav");
19     }
20
21     public void setFile(int i) {
22         try {
23             AudioInputStream ais = AudioSystem.getAudioInputStream(soundUrl[i]);
24             clip = AudioSystem.getClip();
25             clip.open(ais);
26         } catch (Exception e) {
27             e.printStackTrace();
28         }
29     }
30
31     public void play() { clip.start(); }
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

SoundClipIdentifier.java
1
2
3
4
5
6 * Classe che gestisce il singolo suono tra client e server.
7 */
8
9 public class SoundClipIdentifier implements Serializable {
10
11     private int clipNumber; //identifica il suono fra quelli presenti nel gioco
12     private boolean isClipActive; //indica se il suono è attualmente attivo
13
14     public SoundClipIdentifier() {
15         clipNumber = -1;
16         isClipActive = true;
17     }
18
19     public SoundClipIdentifier(int clipNumber, boolean isClipActive) {
20         this.clipNumber = clipNumber;
21         this.isClipActive = isClipActive;
22     }
23
24     public int getClipNumber() { return clipNumber; }
25
26     public boolean isClipActive() { return isClipActive; }
27
28     public void setClipActive(boolean clipActive) { isClipActive = clipActive; }
29
30     public void setClipNumber(int clipNumber) { this.clipNumber = clipNumber; }
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```