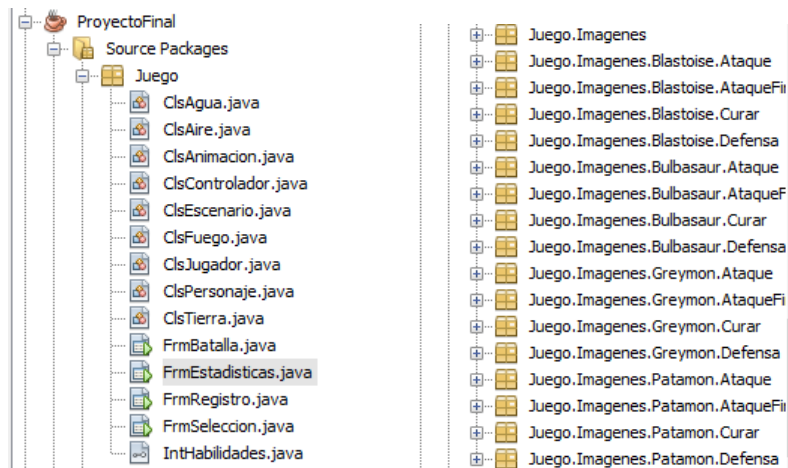


Objetivo General:

Realizar un pequeño juego de peleas mediante el lenguaje java, el IDE Netbeans y el paradigma de la programación orientada a objetos (POO).

1. El proyecto consta de varias clases que se irán detallando a continuación, además de varias carpetas que contienen las imágenes para las animaciones.



2. La clase jugador corresponde al usuario que jugará, el jugador tiene un id, nombre, apellido, cedula y un nickname, además de victorias y se usa herencia por agregación al implementar un personaje en el primer constructor. El segundo constructor está vacío, el toString devuelve el nombre y el apellido, se implementa los respectivos setters y getters, además de hacer la clase serializable para almacenarla en archivos.dat.

```
Start Page x ClsJugador.java x IntHabilidades.java x
Source History
1 package Juego;
2
3 import java.io.Serializable;
4
5 public class ClsJugador implements Serializable{
6     private int id;
7     private String nombre;
8     private String apellido;
9     private String cedula;
10    private String usuario;
11    private int victorias;
12    ClsPersonaje personaje;
13
14    public ClsJugador(int id, String nombre, String apellido, String cedula, String usuario)
15    {
16        this.id = id;
17        this.nombre = nombre;
18        this.apellido = apellido;
19        this.cedula = cedula;
20        this.usuario = usuario;
21        victorias=0;
22        personaje=new ClsPersonaje();
23    }
24
25    public ClsJugador() {
26    }
27
28    @Override
29    public String toString() {
30        return nombre + " " + apellido;
31    }
32
33 }
```

3. La clase personaje igualmente implementa el “Serializable”, tiene un id, nombre, tipo, una imagen, vida, estamina, ataque, ataque final, un escudo y contador para los turnos, el primer constructor recibe el id, nombre, tipo y una imagen, además de inicializar el escudo en false y el turno en cero.

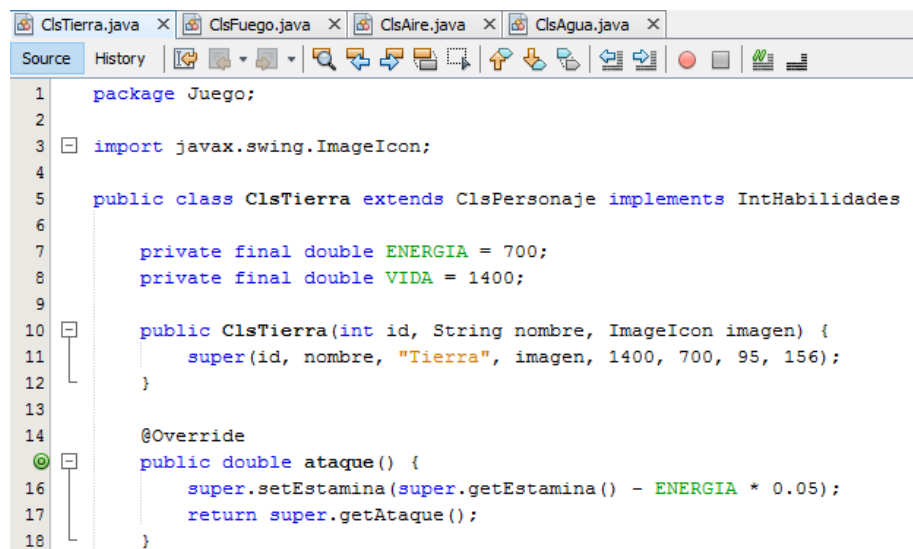
```
ClPersonaje.java X
Source History
1 package Juego;
2
3 import java.io.Serializable;
4 import javax.swing.ImageIcon;
5
6
7 public class ClsPersonaje implements Serializable{
8
9     private int id;
10    private String nombre;
11    private String tipo;
12    private ImageIcon imagen;
13    private double vida;
14    private double estamina;
15    private double ataque;
16    private double ataqueFinal;
17    private boolean escudo;
18    private int turno;
19
20    public ClsPersonaje(int id, String nombre, String tipo, ImageIcon imagen) {
21        this.id = id;
22        this.nombre = nombre;
23        this.tipo = tipo;
24        this.imagen = imagen;
25        escudo=false;
26        turno=0;
27    }
28 }
```

El segundo constructor además de los atributos anteriores recibe la vida, estamina, el ataque y el ataque final, el tercer constructor se encuentra vacío y el método daño disminuye la vida dependiendo si el escudo se encuentra activo o no, además que devuelve un string que se mostrará en el área de texto de la batalla. La clase personaje al igual que las demás implementa los respectivos setters y getters.

```
28 public ClsPersonaje(int id, String nombre, String tipo, ImageIcon imagen, double vida,
29                     double estamina, double ataque, double ataqueFinal) {
30     this.id = id;
31     this.nombre = nombre;
32     this.tipo = tipo;
33     this.imagen = imagen;
34     this.vida = vida;
35     this.estamina = estamina;
36     this.ataque = ataque;
37     this.ataqueFinal = ataqueFinal;
38     escudo=false;
39     turno=0;
40 }
41
42 public ClsPersonaje(){
43 }
44
45 public String daño(double ataque){
46     if(this.isEscudo()==true){
47         this.setVida(this.getVida()-ataque*0.75);
48         return String.valueOf(ataque*0.75);
49     }else{
50         this.setVida(this.getVida()-ataque);
51         return String.valueOf(ataque);
52     }
53 }
54 }
```

4. Las clases tierra, fuego, aire y agua son similares, todas heredan de la clase personaje y utilizan herencia múltiple mediante la *interface* "IntHabilidades", los métodos son parecidos lo único que cambia es la energía y la vida, que se definen como final ya que son constantes sobre las que se realizará la mayoría de los cálculos, los constructores de cada tipo de personaje son diferentes ya que poseen distintas capacidades de ataque, de estamina y de vida.

El método ataque disminuye el 5% de la estamina y devuelve el atributo ataque de la clase, el método defensa depende del turno, si el turno se encuentra en cero se resta el 25% de la estamina, se activa el escudo y se aumenta el turno, el turno sigue incrementándose hasta que llega a tres y el escudo se desactiva, el método curar aumenta la vida del personaje en un 20% y el ataque final realiza lo mismo que ataque pero generando más daño y reduciendo mayor estamina, el aumento de estamina 10, permite aumentar la estamina en un 10% en cada turno.



```
1 package Juego;
2
3 import javax.swing.ImageIcon;
4
5 public class ClsTierra extends ClsPersonaje implements IntHabilidades {
6
7     private final double ENERGIA = 700;
8     private final double VIDA = 1400;
9
10    public ClsTierra(int id, String nombre, ImageIcon imagen) {
11        super(id, nombre, "Tierra", imagen, 1400, 700, 95, 156);
12    }
13
14    @Override
15    public double ataque() {
16        super.setEstamina(super.getEstamina() - ENERGIA * 0.05);
17        return super.getAtaque();
18    }
```

```

20      @Override
21      public void defensa() {
22          switch (super.getTurno()) {
23              case 0:
24                  super.setEstamina(super.getEstamina() - ENERGIA * 0.25);
25                  super.setEscudo(true);
26                  super.setTurno(super.getTurno() + 1);
27                  break;
28              case 3:
29                  super.setEscudo(false);
30                  super.setTurno(0);
31                  break;
32              default:
33                  super.setTurno(super.getTurno() + 1);
34                  break;
35          }
36      }
37
38      @Override
39      public void curar() {
40          super.setEstamina(super.getEstamina() - ENERGIA * 0.20);
41          super.setVida(super.getVida() + VIDA * 0.20);
42      }
43
44
45      @Override
46      public double ataqueFinal() {
47          super.setEstamina(super.getEstamina() - ENERGIA * 0.50);
48          return super.getAtaqueFinal();
49      }
50
51      @Override
52      public void aumentoEstamina10() {
53          super.setEstamina(super.getEstamina() + ENERGIA * 0.10);
54      }

```

```

ClsTierra.java x ClsFuego.java x ClsAire.java x ClsAgua.java x
Source History
1  package Juego;
2
3  import javax.swing.ImageIcon;
4
5  public class ClsFuego extends ClsPersonaje implements IntHabilidades {
6
7      private final double ENERGIA = 500;
8      private final double VIDA = 1300;
9
10     public ClsFuego(int id, String nombre, ImageIcon imagen) {
11         super(id, nombre, "Fuego", imagen, 1300, 500, 110, 230);
12     }

```

```
Clstierra.java x Clsfuego.java x Clsaire.java x Clsagua.java x
Source History
1 package Juego;
2
3 import javax.swing.ImageIcon;
4
5 public class ClsAire extends ClsPersonaje implements IntHabilidades {
6
7     private final double ENERGIA = 400;
8     private final double VIDA = 950;
9
10    public ClsAire(int id, String nombre, ImageIcon imagen) {
11        super(id, nombre, "Aire", imagen, 950, 400, 140, 200);
12    }
```

```
Clstierra.java x Clsfuego.java x Clsaire.java x Clsagua.java x
Source History
1 package Juego;
2
3 import javax.swing.ImageIcon;
4
5 public class ClsAgua extends ClsPersonaje implements IntHabilidades {
6
7     private final double ENERGIA = 600;
8     private final double VIDA = 1200;
9
10    public ClsAgua(int id, String nombre, ImageIcon imagen) {
11        super(id, nombre, "Agua", imagen, 1200, 600, 100, 210);
12    }
```

5. La clase escenario permite colocar una imagen de fondo para el campo de batalla, tienen un nombre y una imagen, esta clase permitirá cargar los escenarios en un combo box.

```
Clsanimacion.java x Clscontrolador.java x Clsescenario.java x
Source History
1 package Juego;
2
3 import javax.swing.ImageIcon;
4
5 public class ClsEscenario {
6
7     private String nombre;
8     private ImageIcon fondo;
9
10    public ClsEscenario(String nombre, ImageIcon fondo) {
11        this.nombre = nombre;
12        this.fondo = fondo;
13    }
14
15    @Override
16    public String toString() {
17        return nombre;
18    }
```

6. La clase controlador permite almacenar la información en archivos.dat.

```

1 package Juego;
2
3 import java.io.EOFException;
4 import java.io.FileInputStream;
5 import java.io.FileNotFoundException;
6 import java.io.FileOutputStream;
7 import java.io.IOException;
8 import java.io.ObjectInputStream;
9 import java.io.ObjectOutputStream;
10 import java.util.ArrayList;
11 import javax.swing.JOptionPane;
12
13 public class ClsControlador {
14
15     public void crearFichero(String nombreFichero) {
16
17         try {
18             ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(nombreFichero));
19             oos.close();
20         } catch (FileNotFoundException ex) {
21             JOptionPane.showMessageDialog(null, ex.getMessage());
22
23         } catch (IOException ex) {
24             JOptionPane.showMessageDialog(null, ex.getMessage());
25         }
26     }
27 }

```

7. En la clase animación se realizó varios cambios como crear un string url para pasar la dirección de la imagen que servirá para la animación, además en el método start animation se incrementó a 22 el número de imágenes y en el caso de que llegue a la imagen 22, esta automáticamente se cambia a la imagen 23, la cual es la imagen que aparece al inicio de la batalla.

```

13 boolean run = false;
14 String url;
15
16 public ClsAnimacion(Dimension d) {
17     this.setSize(d);
18 }
19
20 public void setImage(int n) {
21     this.setIcon(new javax.swing.ImageIcon(getClass().getResource(url + n + ".PNG")));
22 }
23
24 public void setVelocidad(int v) {
25     this.speed = v;
26 }
27
28 public void startAnimation() {
29     frame=0;
30     run = true;
31     tiempo = new Timer();
32     task = new TimerTask() {
33
34         public void run() {
35             frame++;
36             if (frame <= 22) {
37                 setImage(frame);
38                 if(frame==22){
39                     setImage(23);
40                 }
41             }
42         }
43     };

```

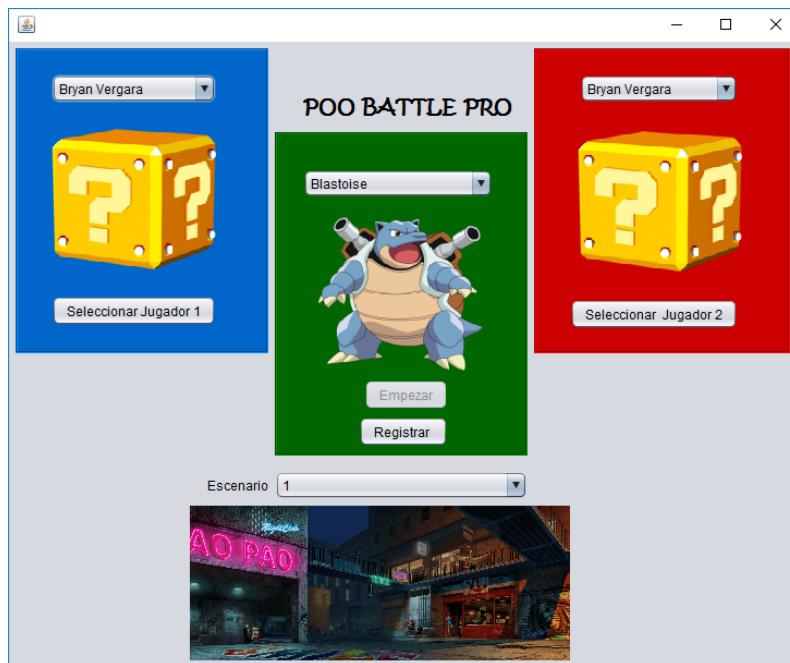
8. La *interface* usada se llama "IntHabilidades" y corresponde a la aplicación de polimorfismo, ya que cada personaje se comporta diferente.

```

1  package Juego;
2
3
4  public interface IntHabilidades
5
6      double ataque();
7      void defensa();
8      void curar();
9      double ataqueFinal();
10
11 }

```

9. En el frame form de selección de personajes se utilizó varios paneles, combo box, botones, además de poder seleccionar el escenario que se desee.



Para el código se creó un array list para los jugadores, personajes y escenarios además del controlador para los archivos.dat, también dos jugadores, un escenario, dos personajes, enteros para traspasar la información a la siguiente ventana, además de dos booleanos.

```

1 package Juego;
2
3 import java.awt.Image;
4 import java.util.ArrayList;
5 import javax.swing.ImageIcon;
6 import javax.swing.JOptionPane;
7
8 public class FrmSeleccion extends javax.swing.JFrame {
9     ArrayList<Object> jugadores = new ArrayList<>();
10     ClsControlador control = new ClsControlador();
11     ArrayList<Object> personajes = new ArrayList<>();
12     ArrayList<ClsEscenario> escenarios = new ArrayList<>();
13
14     ClsPersonaje personajeSeleccionado;
15     ClsJugador player1, player2;
16     int p1,p2,c1,c2,esc;
17     boolean p1Ready=false, p2Ready=false;
18     ClsPersonaje person=new ClsPersonaje();
19     ClsEscenario background;

```

Al inicio del programa se extrae los jugadores del archivo.dat y se pasa su contenido al array list de jugadores, se llenan las imágenes de los jugadores con una imagen específica, se llena los combos con los jugadores y se crean los personajes, se llenan los escenarios y se pasan al combo box respectivo.

```

21 public FrmSeleccion() {
22     initComponents();
23
24     this.setLocationRelativeTo(null);
25     jugadores = control.extraerObjeto("jugadores.dat");
26     jugadoresImagenVacia();
27     llenarCombos();
28     crearPersonajes();
29     String nombreFondo="";
30     for (int i = 1; i <=13; i++) {
31         nombreFondo=String.valueOf(i);
32         ImageIcon base= new ImageIcon(getClass().getResource("Imagenes/"+nombreFondo+".gif"));
33         ClsEscenario escenar= new ClsEscenario(nombreFondo,base);
34         escenarios.add(escenar);
35     }
36     cbbEscenario.setModel(new javax.swing.DefaultComboBoxModel(escenarios.toArray()));
37     cbbEscenario.setSelectedIndex(0);
38
39 }

```

El método jugadoresImagenVacia coloca una imagen de interrogación hasta que se elija un personaje, el método llenar combos, los llena con los jugadores registrados, el método “jugadores listos” verifica que los dos jugadores estén listos para activar el botón de empezar, el método agregar imagen recibe el nombre de la imagen, que busca en la ruta específica y retorna la imagen con esa ruta, el método imagenEtiqueta se adecua al tamaño del label y conforme a esto devuelve una imagen.


```

41 public void jugadoresImagenVacía() {
42     ImageIcon empty= new ImageIcon(getClass().getResource("Imagenes/inter.png"));
43     ImageIcon vacio= new ImageIcon(empty.getImage().getScaledInstance(lblP1.getWidth(),lblP1.getHeight(),
44         Image.SCALE_DEFAULT));
45     lblP1.setIcon(vacio);
46     lblP2.setIcon(vacio);
47 }
48
49 public void llenarCombos() {
50     cbbP1.setModel(new javax.swing.DefaultComboBoxModel(jugadores.toArray()));
51     cbbP2.setModel(new javax.swing.DefaultComboBoxModel(jugadores.toArray()));
52 }
53
54 public void jugadoresListos() {
55     if(p1Ready==true && p2Ready==true){
56         btEmpezar.setEnabled(true);
57     }
58 }
59
60 public ImageIcon agregarImagen(String nombreImagen){
61     ImageIcon base= new ImageIcon(getClass().getResource("Imagenes/"+nombreImagen+".png"));
62     return base;
63 }
64
65 public ImageIcon imagenEtiqueta() {
66     ImageIcon personaje= new ImageIcon(personajeSeleccionado.getImage().getScaledInstance(
67         lblP1.getWidth(),lblP1.getHeight(), Image.SCALE_DEFAULT));
68     return personaje;
69 }

```

El método crearPersonajes instancia objetos de diferentes personajes, luego los pasa al array list respectivo, los almacena en el archivo.dat y los pasa al combo box de personajes. El método personajeJugadorIguales determina si está escogiendo un personaje o un usuario igual, y si es así se desactiva el botón de empezar hasta que el jugador elija una opción válida.

```

71 public void crearPersonajes() {
72     ClsPersonaje blastoise = new ClsAgua(0,"Blastoise",agregarImagen("Blastoise"));
73     ClsPersonaje bulbasaur = new ClsTierra(1,"Bulbasaur",agregarImagen("Bulbasaur"));
74     ClsPersonaje patamon = new ClsAire(2,"Patamon",agregarImagen("Patamon"));
75     ClsPersonaje greymon = new ClsFuego(3,"Greymon",agregarImagen("Greymon"));
76     personajes.add(blastoise);
77     personajes.add(bulbasaur);
78     personajes.add(patamon);
79     personajes.add(greymon);
80     control.escribirObjeto("personajes.dat",personajes);
81     cbbPersonaje.setModel(new javax.swing.DefaultComboBoxModel(personajes.toArray()));
82     cbbPersonaje.setSelectedItem(blastoise);
83 }
84
85 public boolean personajeJugadorIguales() {
86     if(player1==player2 || player1.getPersonaje()==player2.getPersonaje()){
87         JOptionPane.showMessageDialog(null,"Los personajes y usuarios deben ser diferentes, "
88             + "cambiélos e intente de nuevo");
89         btEmpezar.setEnabled(false);
90         return false;
91     }
92     return true;
93 }

```

El botón registrar conduce a la ventana de registro, el combo box de personaje realiza un casting y lo pasa a "personajeSeleccionado" el mismo que usa el método "imagenEtiqueta". El botón para seleccionar al jugador uno usa el método "imagenEtiqueta", si el jugador no tiene un personaje seleccionado toma el primer personaje del combo box, pasa el personaje a "player1" y luego toma los ids del personaje y del jugador, le coloca a listo al jugador y verifica que los dos se encuentren listos, lo mismo se realiza para el botón para seleccionar al jugador dos.

```

341 private void btRegistrarActionPerformed(java.awt.event.ActionEvent evt) {
342     FrmRegistro registro = new FrmRegistro();
343     this.setVisible(false);
344     registro.setVisible(true);
345 }
346
347 private void cbbPersonajeActionPerformed(java.awt.event.ActionEvent evt) {
348     personajeSeleccionado=(ClsPersonaje) cbbPersonaje.getSelectedItem();
349     lblPersonaje.setIcon(imagenEtiqueta());
350 }
351
352 private void btP1ActionPerformed(java.awt.event.ActionEvent evt) {
353     lblP1.setIcon(imagenEtiqueta());
354     if(player1==null){
355         player1=(ClsJugador)jugadores.get(0);
356     }
357     player1.setPersonaje(personajeSeleccionado);
358     p1=player1.getId();
359     c1=player1.getPersonaje().getId();
360     p1Ready=true;
361     jugadoresListos();
362 }
363
364 private void btP2ActionPerformed(java.awt.event.ActionEvent evt) {
365     lblP2.setIcon(imagenEtiqueta());
366     if(player2==null){
367         player2=(ClsJugador)jugadores.get(0);
368     }
369     player2.setPersonaje(personajeSeleccionado);
370     p2=player2.getId();
371     c2=player2.getPersonaje().getId();
372     p2Ready=true;
373     jugadoresListos();
374 }

```

El combo box de los jugadores se realiza un casting para colocar el jugador a “player1” luego se llena el combo box del otro jugador con los jugadores y se elimina el que el actual usuario seleccionó para que no aparezcan los mismos usuarios en el combo box del otro jugador. Lo mismo se realiza para el otro combo box del jugador, pero al revés. El botón empezar corre el método “personaJugadoresIguales” pasa todos los ids a la ventana de batalla y la hace visible. En el combo box del escenario simplemente se cambia la imagen para que el usuario elija en qué escenario jugar.

```

376 private void cbbP1ActionPerformed(java.awt.event.ActionEvent evt) {
377     player1=(ClsJugador) cbbP1.getSelectedItem();
378     cbbP2.setModel(new javax.swing.DefaultComboBoxModel(jugadores.toArray()));
379     cbbP2.removeItem(player1);
380     cbbP2.setSelectedItem(player2);
381 }
382
383
384 private void cbbP2ActionPerformed(java.awt.event.ActionEvent evt) {
385     player2=(ClsJugador) cbbP2.getSelectedItem();
386     cbbP1.setModel(new javax.swing.DefaultComboBoxModel(jugadores.toArray()));
387     cbbP1.removeItem(player2);
388     cbbP1.setSelectedItem(player1);
389 }
390
391
392 private void btEmpezarActionPerformed(java.awt.event.ActionEvent evt) {
393     if(personajeJugadorIguales()){
394         FrmBatalla battle = new FrmBatalla(p1,c1,p2,c2,esc);
395         this.setVisible(false);
396         battle.setVisible(true);
397     }
398 }
399
400 private void cbbEscenarioActionPerformed(java.awt.event.ActionEvent evt) {
401     background=(ClsEscenario) cbbEscenario.getSelectedItem();
402     esc= Integer.parseInt(background.getNombre());
403     ImageIcon fondo= new ImageIcon(background.getFondo().getImage().getScaledInstance(
404         lblFondo.getWidth(),lblFondo.getHeight(), Image.SCALE_DEFAULT));
405     lblFondo.setIcon(fondo);
406 }

```

En el frame de registro permite ingresar usuarios para que puedan jugar y su puntaje quede almacenado en el programa a través de un archivo.dat.

id	Nombre	Apellido	Cédula	Usuario	Victorias
4	Gred	Fred	9999	Fred	0
7	asd	asd	asd	asd	0

```

FrmRegistro.java x FrmBatalla.java x FrmEstadisticas.java x
Source Design History
1 package Juego;
2
3 import java.util.ArrayList;
4 import javax.swing.table.DefaultTableModel;
5
6 public class FrmRegistro extends javax.swing.JFrame {
7     ArrayList<Object> jugadores = new ArrayList<>();
8
9     int cont = 0;
10    ClsControlador controller = new ClsControlador();
11
12    public FrmRegistro() {
13        initComponents();
14        this.setLocationRelativeTo(null);
15        jugadores = controller.extraerObjeto("jugadores.dat");
16        llenarCombo();
17        llenarGrid();
18        if (jugadores.size() > 0) {
19            ClsJugador ultimo = (ClsJugador) jugadores.get(jugadores.size()-1);
20            cont = ultimo.getId()+1;
21        }
22        btnmodificar.setEnabled(false);
23        btneliminar.setEnabled(false);
24    }
25

```

En las modificaciones realizadas esta la creación de un array list de jugadores, y añadir las victorias a la tabla mediante el arreglo de tipo objeto.

```

276 private void btnregistrarActionPerformed(java.awt.event.ActionEvent evt) {
277     ClsJugador jugador = new ClsJugador(cont,txfNombre.getText(),txfApellido.getText(),txfCedula.getText(),txfUsuario.getText());
278     cont++;
279     jugadores.add(jugador);
280     controller.escribirObjeto("jugadores.dat", jugadores);
281     jugadores = controller.extraerObjeto("jugadores.dat");
282     limpiar();
283     llenarCombo();
284     Object jugadoresG[]={jugador.getId(),jugador.getNombre(),jugador.getApellido(),
285         jugador.getCedula(), jugador.getUsuario(),jugador.getVictorias()};
286     DefaultTableModel model = (DefaultTableModel) jTableJugadores.getModel();
287     model.addRow(jugadoresG);
288 }
289
290 private void btnmodificarActionPerformed(java.awt.event.ActionEvent evt) {
291     ClsJugador jugadorSeleccionado = (ClsJugador) cbbJugador.getSelectedItem();
292     jugadorSeleccionado.setNombre(txfNombre.getText());
293     jugadorSeleccionado.setApellido(txfApellido.getText());
294     jugadorSeleccionado.setCedula(txfCedula.getText());
295     jugadorSeleccionado.setUsuario(txfUsuario.getText());
296     Object jugadoresG[] = {jugadorSeleccionado.getId(), jugadorSeleccionado.getNombre(),
297         jugadorSeleccionado.getApellido(),jugadorSeleccionado.getCedula(), jugadorSeleccionado.getUsuario()};
298     DefaultTableModel model = (DefaultTableModel) jTableJugadores.getModel();
299     model.removeRow(cbbJugador.getSelectedIndex());
300
301     jugadores.remove(cbbJugador.getSelectedIndex());
302     model.insertRow(cbbJugador.getSelectedIndex(), jugadoresG);
303     jugadores.add(cbbJugador.getSelectedIndex(), jugadorSeleccionado);
304     controller.escribirObjeto("jugadores.dat", jugadores);
305     llenarCombo();
306     limpiar();
307     btnmodificar.setEnabled(false);

```

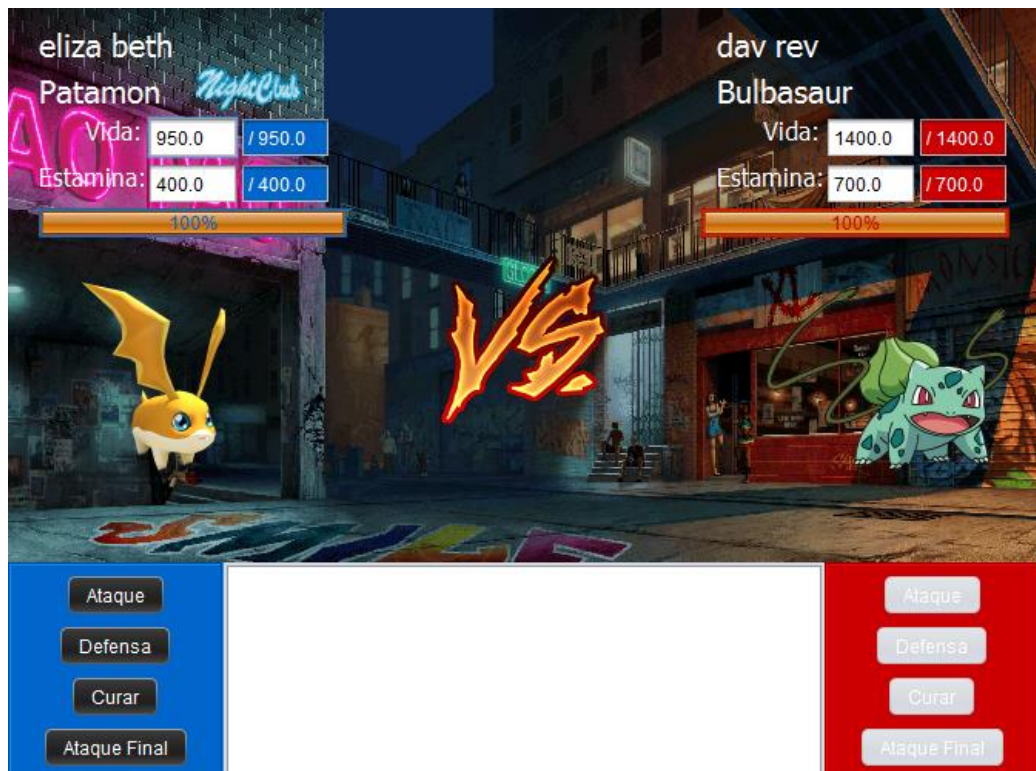
El botón de seleccionar personajes conduce a la ventana de selección de personajes haciendo visible esta.

```

private void btSeleccPersonajesActionPerformed(java.awt.event.ActionEvent evt) {
    FrmSeleccion seleccion= new FrmSeleccion();
    this.setVisible(false);
    seleccion.setVisible(true);
}

```

En la ventana de batalla, se utiliza varias etiquetas, cuadros de texto para la vida y la estamina, una barra de progreso para visualizar la vida del jugador en porcentaje, los respectivos botones de ataque, defensa, curar y ataque final de cada jugador y dos labels para mostrar el escudo y una alerta si el jugador esta con la vida baja.



Se creó enteros tipo static para almacenar los ids de la anterior pantalla, y si declaran variables para extraer los jugadores, los personajes, para hacer casting si utiliza el player1 y player2, lo mismo para los personajes, además de la clase animación y las variables como la vida y la estamina.

```

FrmBatalla.java x FrmEstadisticas.java x
Source Design History
1 package Juego;
2
3 import java.awt.Image;
4 import java.util.ArrayList;
5 import javax.swing.ImageIcon;
6 import javax.swing.JOptionPane;
7
8 public class FrmBatalla extends javax.swing.JFrame {
9
10     public static int j1,jp1,j2,jp2,f;
11     ClsControlador control = new ClsControlador();
12     ArrayList <Object> jugadores= new ArrayList<>();
13     ArrayList <Object> personajes= new ArrayList<>();
14     ClsJugador player1,player2;
15     ClsPersonaje person1,person2;
16     String text="";
17     ClsAnimacion animarP1, animarP2;
18     double VidaP1,VidaP2,EstaminaP1,EstaminaP2;

```

En la función inicial se toman los ids de la pantalla de selección, se coloca el fondo según el tamaño del label, se extraen los estudiantes y los personajes de los archivos.dat, se hace el casting de los jugadores y los personajes según los ids respectivos, además se utiliza el método mostrarDatosJugador que pasa los datos de los jugadores a los labels y según el tipo de personaje se hace el casting en el método tipoPersonajes.

Se pasa los valores constantes como la vida y la estamina a los cuadros de texto respectivos y se coloca los valores máximos y mínimos en la barra de progreso según la vida.

```

20 public FrmBatalla(int p1, int c1,int p2, int c2,int e) {
21     initComponents();
22     this.setLocationRelativeTo(null);
23     j1=p1;
24     jp1=c1;
25     j2=p2;
26     jp2=c2;
27     f=e;
28     //aqui se toma una imagen de fondo, se la adecua a las medidas de la
29     //etiqueta "Animación" y luego se pasa la imagen a la etiqueta
30     ImageIcon fondo = new ImageIcon(getClass().getResource("Imagenes/"+e+".gif"));
31     ImageIcon background= new ImageIcon(fondo.getImage().getScaledInstance(
32         lblAnimacion.getWidth(),lblAnimacion.getHeight(), Image.SCALE_DEFAULT));
33     lblAnimacion.setIcon(background);
34
35     jugadores = control.extraerObjeto("jugadores.dat");
36     personajes = control.extraerObjeto("personajes.dat");
37     player1=(ClsJugador) jugadores.get(p1);
38     player2=(ClsJugador) jugadores.get(p2);
39     player1.setPersonaje((ClsPersonaje)personajes.get(c1));
40     player2.setPersonaje((ClsPersonaje)personajes.get(c2));
41     mostrarDatosJugador();
42     tipoPersonajes();
43
44     VidaP1 = person1.getVida();
45     LifeP1.setText("/ "+VidaP1);
46     BarP1.setMaximum((int)VidaP1);
47     BarP1.setMinimum(0);
48     VidaP2 = person2.getVida();
49     LifeP2.setText("/ "+VidaP2); BarP2.setValue((int)VidaP2);
50     BarP2.setMaximum((int)VidaP2);

```

Luego se deshabilita los botones del jugador dos, se llena los campos de texto con la vida y la estamina, se pasa la imagen del personaje al label respectivo y se instancian dos objetos de la clase animación.


```

51     BarP2.setMinimum(0);
52     EstaminaP1 = person1.getEstamina();
53     StaminaP1.setText("/ " + EstaminaP1);
54     EstaminaP2 = person2.getEstamina();
55     StaminaP2.setText("/ " + EstaminaP2);
56     turnoP2(false);
57     llenarCampos();
58
59     lblPlayer1.setIcon(person1.getImagen());
60     lblPlayer2.setIcon(person2.getImagen());
61
62     animarP1 = new ClsAnimacion(lblPlayer1.getSize());
63     animarP1.setVelocidad(99);
64     lblPlayer1.add(animarP1);
65     lblPlayer1.repaint();
66     animarP2 = new ClsAnimacion(lblPlayer2.getSize());
67     animarP2.setVelocidad(99);
68     lblPlayer2.add(animarP2);
69     lblPlayer2.repaint();
70 }

```

El método ganador, verifica si la vida del personaje llegó a cero para determinar si el otro personaje ganó, al ganar se aumenta las victorias se muestra un mensaje en la pantalla, se quita el jugador del array list se añade el nuevo que tiene modificado las victorias, luego se sobrescribe en el archivo.dat y se cambia a la pantalla de estadísticas.

```

72 public void ganador() {
73     if(person1.getVida() <= 0) {
74         text = text + "Jugador 2 Ganó \t Jugador 1 Derrotado";
75         txaBatalla.setText(text);
76         player2.setVictorias(player2.getVictorias() + 1);
77         jugadores.remove(player2.getId());
78         jugadores.add(player2.getId(), player2);
79         control.escribirObjeto("jugadores.dat", jugadores);
80         JOptionPane.showMessageDialog(null, " Juego Terminado\nJugador 2 Ganador");
81         FrmEstadisticas puntajes = new FrmEstadisticas();
82         this.setVisible(false);
83         puntajes.setVisible(true);
84     }
85     if(person2.getVida() <= 0) {
86         text = text + "Jugador 1 Ganó \t Jugador 2 Derrotado";
87         txaBatalla.setText(text);
88         player1.setVictorias(player1.getVictorias() + 1);
89         jugadores.remove(player1.getId());
90         jugadores.add(player1.getId(), player1);
91         control.escribirObjeto("jugadores.dat", jugadores);
92         JOptionPane.showMessageDialog(null, " Juego Terminado\nJugador 1 Ganador");
93         FrmEstadisticas puntajes = new FrmEstadisticas();
94         this.setVisible(false);
95         puntajes.setVisible(true);
96     }
97 }
98 //mostrarDatosJugador pasa los nombres de los jugadores y de sus personajes
99 //a las respectivas etiquetas
100 public void mostrarDatosJugador() {
101     lblNombreP1.setText(player1.toString());
102     lblNombreP2.setText(player2.toString());
103     lblPersonajeP1.setText(player1.getPersonaje().getNombre());
104     lblPersonajeP2.setText(player2.getPersonaje().getNombre());
105 }

```

Algunos de los siguientes métodos tienen su funcionalidad comentada por lo que no se escribe nuevamente.

```

107 //tipoPersonajes toma el id de los personajes que se encuentran dentro del jugador
108 //dependiendo del id, se ejecuta un switch que realiza un casting para determinar
109 //los personajes de los jugadores
110 public void tipoPersonajes() {
111     int p1=player1.getPersonaje().getId();
112     int p2=player2.getPersonaje().getId();
113     switch(p1) {
114         case 0: person1=(ClsAgua)player1.getPersonaje();
115             break;
116         case 1: person1=(ClsTierra)player1.getPersonaje();
117             break;
118         case 2: person1=(ClsAire)player1.getPersonaje();
119             break;
120         case 3: person1=(ClsFuego)player1.getPersonaje();
121             break;
122     }
123     switch(p2) {
124         case 0: person2=(ClsAgua)player2.getPersonaje();
125             break;
126         case 1: person2=(ClsTierra)player2.getPersonaje();
127             break;
128         case 2: person2=(ClsAire)player2.getPersonaje();
129             break;
130         case 3: person2=(ClsFuego)player2.getPersonaje();
131             break;
132     }
133 }

134 //defense verifica si el escudo se encuentra esta activado, si esto ocurre se vuelve a
135 //ejecutar el método defensa que incrementa un contador interno de la respectiva clase,
136 //además que se coloca una imagen de un escudo para avisar al jugador que tiene defensa
137 public void defenseP1() {
138     if(person1.isEscudo()==true) {
139         person1.defensa();
140         lblEscudo1.setIcon(new ImageIcon(getClass().getResource("Imagenes/shield.png")));
141     }
142 }
143 public void defenseP2() {
144     if(person2.isEscudo()==true) {
145         person2.defensa();
146         lblEscudo2.setIcon(new ImageIcon(getClass().getResource("Imagenes/shield.png")));
147     }
148 }
149 //turno desactiva o activa los botones de juego dependiendo del jugador y del turno
150 public void turnoP1(boolean p1) {
151     btAtaqueP1.setEnabled(p1);
152     btDefensaP1.setEnabled(p1);
153     btCurarP1.setEnabled(p1);
154     btAFinalP1.setEnabled(p1);
155 }
156 public void turnoP2(boolean p2) {
157     btAtaqueP2.setEnabled(p2);
158     btDefensaP2.setEnabled(p2);
159     btCurarP2.setEnabled(p2);
160     btAFinalP2.setEnabled(p2);
161 }

```



```

162 //llenarCampos llena la vida y estamina de los dos jugadores en los cuadros de texto
163 public void llenarCampos(){
164     txtfVidaP1.setText(String.valueOf(person1.getVida()));
165     BarP1.setValue((int)person1.getVida());
166     txtfEstaminaP1.setText(String.valueOf(person1.getEstamina()));
167     txtfVidaP2.setText(String.valueOf(person2.getVida()));
168     BarP2.setValue((int)person2.getVida());
169     txtfEstaminaP2.setText(String.valueOf(person2.getEstamina()));
170 }
171 //comprobarEstamina verifica que exista la suficiente estamina para realizar
172 //cierto movimiento, si el botón del método no tiene la suficiente se desactiva
173 public void comprobarEstaminaP1(){
174     if(person1.getEstamina() < EstaminaP1 * 0.05){
175         btAtaqueP1.setEnabled(false);
176     }
177     if(person1.getEstamina() < EstaminaP1 * 0.25){
178         btDefensaP1.setEnabled(false);
179     }
180     if(person1.getEstamina() < EstaminaP1 * 0.20){
181         btCurarP1.setEnabled(false);
182     }
183     if(person1.getEstamina() < EstaminaP1 * 0.50){
184         btAFinalP1.setEnabled(false);
185     }
186     ganador();
187 }

```

En el método *comprobarEstamina* se implementa un método para cada jugador, además se utiliza el método *ganador* que se irá ejecutando en cada turno para verificar si un jugador venció al otro.

El método alerta es general para los dos jugadores y coloca una imagen vacía por si se tiene el gif de alerta cargado en el label, lo que hace el método es colocar un gif de alerta si la vida del jugador es menor o igual a 230, se tomó el 230 porque es el mayor daño que puede generar un personaje en un turno.

```

188 public void comprobarEstaminaP2(){
189     if(person2.getEstamina() < EstaminaP2 * 0.05){
190         btAtaqueP2.setEnabled(false);
191     }
192     if(person2.getEstamina() < EstaminaP2 * 0.25){
193         btDefensaP2.setEnabled(false);
194     }
195     if(person2.getEstamina() < EstaminaP2 * 0.20){
196         btCurarP2.setEnabled(false);
197     }
198
199     if(person2.getEstamina() < EstaminaP2 * 0.50){
200         btAFinalP2.setEnabled(false);
201     }
202     ganador();
203 }
204 public void alerta(){
205     ImageIcon vacio = new ImageIcon(getClass().getResource("Imagenes/vac.png"));
206     lblPlayer1.setIcon(vacio);
207     lblPlayer2.setIcon(vacio);
208     if(person1.getVida() <= 230){
209         ImageIcon alert1 = new ImageIcon(getClass().getResource("Imagenes/alerta.gif"));
210         lblAlertaP1.setIcon(alert1);
211     }else{
212         lblAlertaP1.setIcon(vacio);
213     }
214     if(person2.getVida() <= 230){
215         ImageIcon alert2 = new ImageIcon(getClass().getResource("Imagenes/alerta.gif"));
216         lblAlertaP2.setIcon(alert2);
217     }else{
218         lblAlertaP2.setIcon(vacio);
219     }
220 }

```

```

221 //animarPersonaje utiliza la clase animar la cual recibe el nombre de la carpeta y
222 //el personaje, inicia su animación y termina la animación del otro jugador
223 public void animarPersonajeP1(String nombreCarpeta){
224     alerta();
225     animarP1.setUrl("Imagenes/"+person1.getNombre()+"/"+nombreCarpeta+"/");
226     animarP1.startAnimation();
227     //Esta condición se usa ya que en el primer turno la animación del jugador 2
228     //no ha sido iniciada por lo que daría un error si se pone stopAnimation
229     if(animarP2.run==true){
230         animarP2.stopAnimation();
231     }
232 }
233 public void animarPersonajeP2(String nombreCarpeta){
234     alerta();
235     animarP2.setUrl("Imagenes/"+person2.getNombre()+"/"+nombreCarpeta+"/");
236     animarP2.startAnimation();
237     animarP1.stopAnimation();
238 }
239 //areaBattle muestra en el área de texto el nombre del jugador junto con el movimiento
240 //que realizó, además de la cantidad si se trata de un ataque
241 public void areaBattle(String txt, ClsJugador p, String habilidad){
242     text=text+p.toString()+" : "+txt+" ( "+habilidad+" ) "+"\\n";
243     txtBatalla.setText(text);
244 }
245 //escudoDesactivado sirve para quitar la imagen del escudo colocando una
246 //imagen png vacía, se ejecuta cuando el personaje no tiene escudo
247 public void escudoDesactivadoP1(){
248     if(person1.isEscudo()==false){
249         lblEscudo1.setIcon(new ImageIcon(getClass().getResource("Imagenes/vac.png")));
250     }
251 }
252 public void escudoDesactivadoP2(){
253     if(person2.isEscudo()==false){
254         lblEscudo2.setIcon(new ImageIcon(getClass().getResource("Imagenes/vac.png")));
255     }
256 }

```

El botón de ataque almacena el daño en un string que luego se pasará al área de texto, se anima el personaje ingresando el nombre de la carpeta donde se encuentra, el nombre del personaje esta implementado en el método, luego se ejecuta "areaBattle" que muestra los movimientos de los jugador en el área de texto, se llena los campos de texto con las nuevas características de vida y estamina, se desactivan los botones del actual jugador y se activan los botones del jugador contrario, pero si no se tiene la suficiente estamina ciertos botones del jugador deben desactivarse, para lo cual se usa el método de comprobación de estamina, se verifica si el escudo esta desactivado, se utiliza el método "defense" del jugador contrario y se aumenta el estamina en un 10% del jugador contrario.

```

515 private void btAtaqueP1ActionPerformed(java.awt.event.ActionEvent evt) {
516     String daño="daño causado: "+person2.daño(person1.ataque());
517     animarPersonajeP1("Ataque");
518     areaBattle("Usó ataque", player1, daño);
519     llenarCampos();
520     turnoP1(false);
521     turnoP2(true);
522     comprobarEstaminaP2();
523     escudoDesactivadoP1();
524     defenseP2();
525     person2.aumentoEstamina10();
526 }
527
528 private void btAtaqueP2ActionPerformed(java.awt.event.ActionEvent evt) {
529     String daño="daño causado: "+person1.daño(person2.ataque());
530     animarPersonajeP2("Ataque");
531     llenarCampos();
532     areaBattle("Usó ataque", player2, daño);
533     turnoP2(false);
534     turnoP1(true);
535     comprobarEstaminaP1();
536     escudoDesactivadoP2();
537     defenseP1();
538     person1.aumentoEstamina10();
539 }

```

En el botón de defensa se coloca automáticamente la imagen del escudo en la etiqueta respectiva, además de colocar el número del turno en cero, si llegará a presionar el botón nuevamente estando por ejemplo en el turno dos con el escudo activado. Luego se ejecuta el método defensa que activa el escudo y comienza a aumentar el contador de turno, se utiliza el método animarPersonaje que ya se explicó, pero en este caso accede a otro nombre de carpeta, esto seguirá cambiando dependiendo del botón que se use, se llena los campos de texto, luego el área de batalla, igualmente que en el ataque y en todos los botones se utiliza los métodos "turno", comprobarEstamina y aumento de estamina.

```

541 private void btDefensaP1ActionPerformed(java.awt.event.ActionEvent evt) {
542     lblEscudo1.setIcon(new ImageIcon(getClass().getResource("Imagenes/shield.png")));
543     person1.setTurno(0);
544     person1.defensa();
545     animarPersonajeP1("Defensa");
546     llenarCampos();
547     areaBattle("Usó defensa", player1, "Escudo activado");
548     turnoP1(false);
549     turnoP2(true);
550     comprobarEstaminaP2();
551     person2.aumentoEstamina10();
552 }
553
554 private void btDefensaP2ActionPerformed(java.awt.event.ActionEvent evt) {
555     lblEscudo2.setIcon(new ImageIcon(getClass().getResource("Imagenes/shield.png")));
556     person2.setTurno(0);
557     person2.defensa();
558     animarPersonajeP2("Defensa");
559     llenarCampos();
560     areaBattle("Usó defensa", player2, "Escudo activado");
561     turnoP2(false);
562     turnoP1(true);
563     comprobarEstaminaP1();
564     person1.aumentoEstamina10();
565 }

```

En el botón curar se utiliza el método curar, que incrementa la vida, y al igual que en los otros botones se ejecutan los mismos métodos comunes en el mismo orden.

```

567 private void btCurarP1ActionPerformed(java.awt.event.ActionEvent evt) {
568     person1.curar();
569     animarPersonajeP1("Curar");
570     llenarCampos();
571     areaBattle("Usó curación", player1, "aumentó vida: "+VidaP1*0.20);
572     turnoP1(false);
573     turnoP2(true);
574     comprobarEstaminaP2();
575     escudoDesactivadoP1();
576     defenseP2();
577     person2.aumentoEstamina10();
578 }
579
580 private void btCurarP2ActionPerformed(java.awt.event.ActionEvent evt) {
581     person2.curar();
582     animarPersonajeP2("Curar");
583     llenarCampos();
584     areaBattle("Usó curación", player2, "aumentó vida: "+VidaP2*0.20);
585     turnoP2(false);
586     turnoP1(true);
587     comprobarEstaminaP1();
588     escudoDesactivadoP2();
589     defenseP1();
590     person1.aumentoEstamina10();
591 }

```

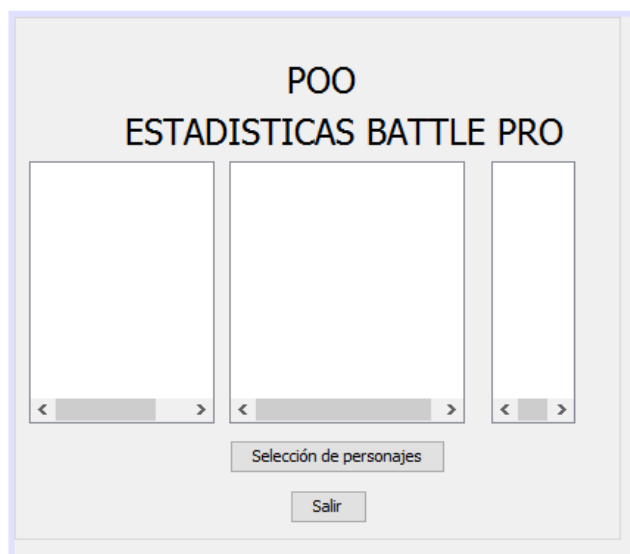
El botón ataque final, ejecuta los mismos métodos y en el mismo orden que el ataque, lo único que cambia es el método de la clase personaje el nombre de la carpeta de animación y el string que recibe en "areaBattle".

```

593 private void btAFinalP1ActionPerformed(java.awt.event.ActionEvent evt) {
594     String daño="daño causado: "+person2.daño(person1.ataqueFinal());
595     animarPersonajeP1("AtaqueFinal");
596     llenarCampos();
597     areaBattle("Usó ataque final",player1,daño);
598     turnoP1(false);
599     turnoP2(true);
600     comprobarEstaminaP2();
601     escudoDesactivadoP1();
602     defenseP2();
603     person2.aumentoEstamina10();
604 }
605
606 private void btAFinalP2ActionPerformed(java.awt.event.ActionEvent evt) {
607     String daño="daño causado: "+person1.daño(person2.ataqueFinal());
608     animarPersonajeP2("AtaqueFinal");
609     llenarCampos();
610     areaBattle("Usó ataque final",player2,daño);
611     turnoP2(false);
612     turnoP1(true);
613     comprobarEstaminaP1();
614     escudoDesactivadoP2();
615     defenseP1();
616     person1.aumentoEstamina10();
617 }

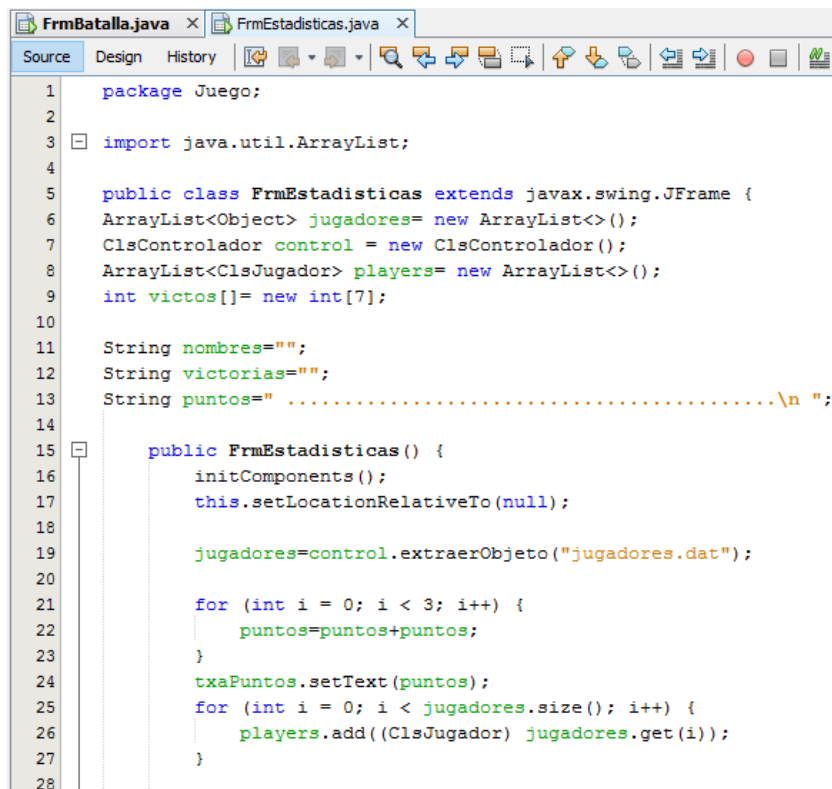
```

Finalmente, en la ventana de estadísticas se utilizar tres áreas de texto, un área para los jugadores, otra para los puntos seguidos y otra para el número de victorias además de dos etiquetas y dos botones.



Al inicio se crea un array list para los jugadores, un controlador para extraer los mismos y array list de players para hacer el casting del anterior array list y un arreglo "victos" que almacenará las victorias.

Primero se utiliza un for y los puntos para llenar el área de texto con los puntos. Luego otro for para realizar el casting de los jugadores.



```

1  package Juego;
2
3  import java.util.ArrayList;
4
5  public class FrmEstadisticas extends javax.swing.JFrame {
6      ArrayList<Object> jugadores= new ArrayList<>();
7      ClsControlador control = new ClsControlador();
8      ArrayList<ClsJugador> players= new ArrayList<>();
9      int victos[]= new int[7];
10
11      String nombres="";
12      String victorias="";
13      String puntos=" .....\\n ";
14
15      public FrmEstadisticas() {
16          initComponents();
17          this.setLocationRelativeTo(null);
18
19          jugadores=control.extraerObjeto("jugadores.dat");
20
21          for (int i = 0; i < 3; i++) {
22              puntos=puntos+puntos;
23          }
24          txaPuntos.setText(puntos);
25          for (int i = 0; i < jugadores.size(); i++) {
26              players.add((ClsJugador) jugadores.get(i));
27          }
28

```

Aquí se pasa las victorias de todos los jugadores al arreglo “victos”, luego se ordena este arreglo mediante el método sort, para después almacenarlo en el string de victorias y pasarlo al área de texto.

Después se compara el arreglo de “players” con el de “victos” para determinar cuál tiene puntaje mayor y según esto ir colocándole en el área de texto.



```

29      for (int i = 0; i < 7; i++) {
30          if(players.size()>i){
31              victos[i]=players.get(i).getVictorias();
32          }else{
33              victos[i]=0;
34          }
35      }
36      sort(victos);
37      for (int i = 6; i >= 0; i--) {
38          victorias=victorias+victos[i]+"\\n";
39      }
40      int cont=0;
41      for (int k = 6; k >=0; k--) {
42
43          for (int j = 0; j < players.size(); j++) {
44              if(victos[k]==players.get(j).getVictorias()){
45                  nombres=nombres+players.get(j).toString()+"\\n";
46                  players.remove(j);
47                  j=players.size();
48                  cont++;
49              }
50          }
51      }
52      for(int i=0; i<7-cont;i++){
53          nombres = nombres+ "Anonimo\\n";
54      }
55
56      txaJugadores.setText(nombres);
57      txaVictorias.setText(victorias);

```

El método sort es el único método y lo que hace es ordenar el arreglo de menor a mayor, el tipo de ordenamiento que utiliza es Shell sort.

```
58 |  
59 | }  
60 |  
61 | public void sort(int arr[]) {  
62 |     int inner, outer;  
63 |     int temp;  
64 |  
65 |     int h = 1;  
66 |     while (h <= arr.length / 3) {  
67 |         h = h * 3 + 1;  
68 |     }  
69 |     while (h > 0) {  
70 |         for (outer = h; outer < arr.length; outer++) {  
71 |             temp = arr[outer];  
72 |             inner = outer;  
73 |  
74 |             while (inner > h - 1 && arr[inner - h] >= temp) {  
75 |                 arr[inner] = arr[inner - h];  
76 |                 inner -= h;  
77 |             }  
78 |             arr[inner] = temp;  
79 |         }  
80 |         h = (h - 1) / 3;  
81 |     }  
82 |  
83 | }
```

El botón seleccionar personajes cambia a la pantalla de selección de personajes una vez que se haya agregado un usuario y el botón salir lo que hace es terminar la ejecución del programa y salir.

```
205 | private void btSeleccionarPersonajesActionPerformed(java.awt.event.ActionEvent evt) {  
206 |     FrmSeleccion seleccion= new FrmSeleccion();  
207 |     this.setVisible(false);  
208 |     seleccion.setVisible(true);  
209 | }  
210 |  
211 | private void btSalirActionPerformed(java.awt.event.ActionEvent evt) {  
212 |     System.exit(0);  
213 | }
```