Informe de Laboratorio 22

Tema: Interfaz Gráfica de Usuario

Nota

Estudiante	Escuela	Asignatura
Hernan Andy Choquehuanca Zapana hchoquehuancaz@unsa.edu.pe	Escuela Profesional de Ingenieria de Sistemas	Fundamentos de la Programacion II Semestre: II Código: 1701213

Laboratorio	Tema	Duración
22	Interfaz Gráfica de Usuario	16 horas

Semestre académico	Fecha de inicio	Fecha de entrega
2023 - B	Del 10 Enero 2024	Al 15 Enero 2024

1. Tarea

- Cree una versión del videojuego de estrategia usando componentes básicos GUI: Etiquetas, botones, cuadros de texto, JOptionPane, Color.
- Además, utilizar componentes avanzados GUI: Layouts, JPanel, áreas de texto, checkbox, botones de radio y combobox.
- Considerar nivel estratégico y táctico.
- Considerar hasta las unidades especiales de los reinos.
- Hacerlo iterativo.

2. Equipos, materiales y temas utilizados

- Sistema Operativo Windows 11 Pro 22H2 64 bits.
- Visual Studio Code.
- Git 2.42.0.
- Cuenta en GitHub con el correo institucional.
- Editor LaTeX en línea Overleaf.
- Herencia.
- Polimorfismo.
- Miembros de clase.
- Clases de Usuario.
- Javax.swing.
- Java.awt.
- Lambda.

3. URL de Repositorio Github

- URL del Repositorio GitHub para clonar o recuperar.
- `https://github.com/hernanchoquehuanca/fp2-23b.git`
- URL para el laboratorio 22 en el Repositorio GitHub.
- `https://github.com/hernanchoquehuanca/fp2-23b/tree/main/fase03/lab22`

4. Trabajo del Laboratorio 22

4.1. Clase VideoJuego.java

Listing 1: Commit 9575acc: Concluyendo la clase principal, la cual contiene el main junto con la interfaz gráfica

```
$ git add .
$ git commit -m "Concluyendo la clase principal en la que se desarrollara el juego"
$ git push -u origin main
```

4.1.1. Imports

- `import java.awt.*;`
- `import java.util.ArrayList;`
- `import javax.swing.*;`
- `import javax.swing.border.EmptyBorder;`

```
1  import java.awt.*;
2  import java.util.ArrayList;
3
4  import javax.swing.*;
5  import javax.swing.border.EmptyBorder;
```

4.1.2. Atributos

- `static boolean turnA = true;`: Para tener un acceso global a ese booleano que sirve como referencia para saber a quien pertenece el turno.
- `public static JButton buttonSel = null;`: Este botón permite tener un guardado del botón presionado, que en nuestro caso sería el soldado seleccionado.

```
7  public class VideoJuego {
8    static boolean turnA = true;
9    public static JButton buttonSel = null;
```

4.1.3. Métodos

- `main(String[] args)`:
- Nuestro método principal, en el cual se hace llamado al método startGame() que es quien inicia el juego.

```
10   public static void main(String[] args) {
11     startGame();
12   }
```

- `startGame()`:
- Este método contiene dentro de sí la ventana inicial, la cual nos da las opciones de Juego 1vs1, Juego personalizado y Salir.
- Dentro de este se usa JFrame, JLabel, JButton, JImageIcon y JPanel.
- Cada uno de los estos objetos creados cumple para que el diseño de la pagina principal sea la que se muestra a continuación:

Figura 1

```
15  public static void startGame() {
16
17    // Window menu settings
18    JFrame window1 = new JFrame("Medieval War Game");
19    window1.setSize(600, 720);
20    window1.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
21    window1.setLocationRelativeTo(null);
```

```
22      window1.setResizable(false);
23
24      // Window menu content
25
26        // Top
27      JLabel w1Label01 = new JLabel("Bienvenido al Juego");
28      w1Label01.setFont(new Font ("Courier New", Font.BOLD, 24));
29      w1Label01.setHorizontalAlignment(SwingConstants.CENTER);
30      w1Label01.setBorder(new EmptyBorder(30, 10, 10, 10));
31      window1.add(w1Label01, BorderLayout.NORTH);
32
33        //Image
34      ImageIcon w1Icon01 = new ImageIcon("fase03\\lab22\\src\\01game.png");
35      JLabel w1JLabel02 = new JLabel(w1Icon01);
36      w1JLabel02.setSize(400,400);
37      window1.add(w1JLabel02, BorderLayout.CENTER);
38
39        //Buttons
40      JPanel w1JPanel01 = new JPanel();
41      w1JPanel01.setLayout(new BoxLayout(w1JPanel01, BoxLayout.Y_AXIS));
42      w1JPanel01.setBorder(new EmptyBorder(20, 10, 20, 10));
43      JButton w1Button01 = new JButton(" Juego 1 vs 1 ");
44      JButton w1Button02 = new JButton("Juego Personalizado");
45      JButton w1Button03 = new JButton("   Salir      ");
46      Font w1Font01 = new Font("Courier New", Font.BOLD, 14);
47      w1Button01.setFont(w1Font01);
48      w1Button02.setFont(w1Font01);
49      w1Button03.setFont(w1Font01);
50      Dimension w1Dimension01 = new Dimension(200, 50);
51      w1Button01.setPreferredSize(w1Dimension01);
52      w1Button02.setPreferredSize(w1Dimension01);
53      w1Button03.setPreferredSize(w1Dimension01);
54      w1Button01.setHorizontalAlignment(SwingConstants.CENTER);
55      w1Button02.setHorizontalAlignment(SwingConstants.CENTER);
56      w1Button03.setHorizontalAlignment(SwingConstants.CENTER);
57      w1Button01.addActionListener(e -> game1vs1(window1));
58      w1Button02.addActionListener(e -> gameCustom(window1));
59      w1Button03.addActionListener(e -> System.exit(0));
60      w1JPanel01.add(Box.createRigidArea(new Dimension(0, 10)));
61      w1JPanel01.add(w1Button01);
62      w1JPanel01.add(Box.createRigidArea(new Dimension(0, 10)));
63      w1JPanel01.add(w1Button02);
64      w1JPanel01.add(Box.createRigidArea(new Dimension(0, 10)));
65      w1JPanel01.add(w1Button03);
66      w1JPanel01.add(Box.createRigidArea(new Dimension(0, 10)));
67      JPanel w1JPanel02 = new JPanel();
68      w1JPanel02.setLayout(new BoxLayout(w1JPanel02, BoxLayout.X_AXIS));
69      w1JPanel02.add(Box.createHorizontalGlue());
70      w1JPanel02.add(w1JPanel01);
71      w1JPanel02.add(Box.createHorizontalGlue());
72      window1.add(w1JPanel02, BorderLayout.SOUTH);
73
74      // Visible
75      window1.setVisible(true);
76  }
```

- `game1vs1(JFrame w1)`:
- El método contiene una segunda ventana en la cual se muestra la interfaz del juego de soldados.
- Comienza utilizando dispose() para dejar de mostrar la ventana anterior.
- Dentro configura la pantalla, agrega los elementos necesarios para el juego y se crean los objetos que permiten la ejecución del juego como: Mapa, Ejercito y Soldado.
- Resaltar que se hace uso de funciones lambda, las cuales permiten darle acciones a los botones, además de ayudar al dinamismo del juego.

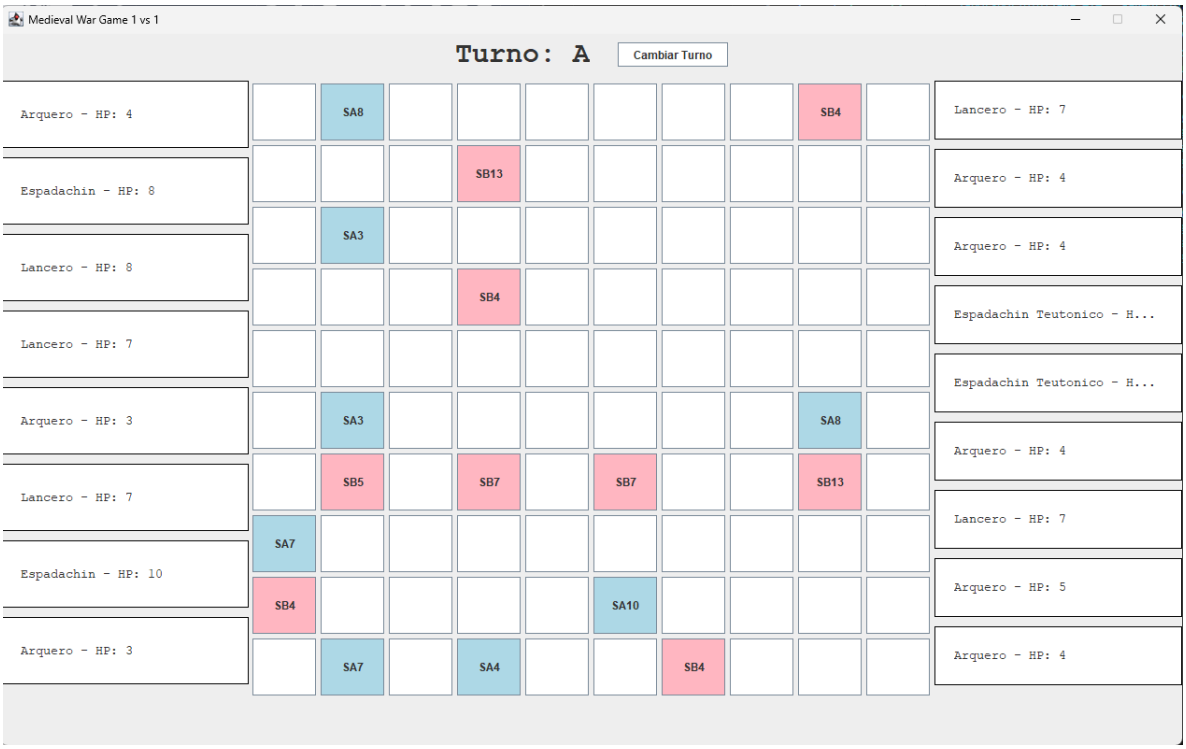

Figura 2

```
79  public static void game1vs1(JFrame w1) {
80    // Window 1 dispose
81    w1.dispose();
82
83    // Window 1vs1 settings
84    JFrame window2 = new JFrame("Medieval War Game 1 vs 1");
85    window2.setSize(1280, 800);
86    window2.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
87    window2.setLocationRelativeTo(null);
88    window2.setResizable(false);
89
90    //Create objects
91    Mapa map = Mapa.getInstance();
92    map.setRandomMapa();
93    Ejercito ejA = new Ejercito(map, 'A');
```

```
94      Ejercito ejB= new Ejercito(map, 'B');
95
96      //Window North
97      JPanel w2JPanel01 = new JPanel(new FlowLayout());
98      w2JPanel01.setPreferredSize(new Dimension(1280,50));
99      JLabel w2JLabel01 = new JLabel("Turno: A");
100     w2JLabel01.setFont(new Font("Courier New", Font.BOLD, 30));
101     JButton w2Button01 = new JButton("Cambiar Turno");
102     w2Button01.setBackground(Color.WHITE);
103     w2Button01.addActionListener(e -> {
104       turnA = changeTurn(turnA);
105       // Add Threads
106       w2JLabel01.setText(turnA ? "Turno: A" : "Turno: B");
107     });
108     w2JPanel01.add(w2JLabel01);
109     w2JPanel01.add(Box.createHorizontalStrut(20));
110     w2JPanel01.add(w2Button01);
111     window2.add(w2JPanel01, BorderLayout.NORTH);
112
113     // Window Center (board)
114     JPanel w2JPanel02 = createJPanelC(map.getBoard(), ejA, ejB, map);
115     window2.add(w2JPanel02, BorderLayout.CENTER);
116
117     // Window West
118     JPanel w2JPanel03 = creatPanelWE(ejA.getArmy());
119     w2JPanel03.setPreferredSize(new Dimension(265,800));
120     window2.add(w2JPanel03, BorderLayout.WEST);
121
122     // Window East
123     JPanel w2JPanel04 = creatPanelWE(ejB.getArmy());
124     w2JPanel04.setPreferredSize(new Dimension(265,800));
125     window2.add(w2JPanel04, BorderLayout.EAST);
126
127     // Window South
128     JPanel w2JPanel05 = new JPanel();
129     w2JPanel05.setPreferredSize(new Dimension(1280,50));
130     window2.add(w2JPanel05, BorderLayout.SOUTH);
131
132     // Show Window2
133     window2.setVisible(true);
134   }
```

- `createJPanelC(Soldado[][] ejS, Ejercito a, Ejercito b, Mapa map)`:
- Este método, el cual es usado anteriormente, crea un JPanel el cual va a contener el tablero con los soldados ya creados.
- Verificando con bucles crea botones según sea el caso, blancos para vacíos, azul para el equipo A y azules para el equipo B.
- Haciendo uso de funciones lambda también se le asigna eventos a los botones, estos nos sirven al momento de que el jugador selecciona algún soldado.

```
136 public static JPanel createJPanelC(Soldado[][] ejS, Ejercito a, Ejercito b, Mapa map){
137   JPanel xxJPanel01 = new JPanel(new GridLayout(10, 10, 5, 5));
138   for (int r = 0; r < 10; r++) {
139     for (int c = 0; c < 10; c++) {
140       Soldado sP = ejS[r][c];
141       JButton xxButton01 = new JButton();
142       xxButton01.setPreferredSize(new Dimension(5, 5));
143       if (sP != null) {
144         xxButton01.putClientProperty("s", sP.getActualLife());
145         xxButton01.putClientProperty("p", sP.getPosition());
146         xxButton01.setText("S" + sP.getName().substring(7,8) + sP.getActualLife());
147         xxButton01.addActionListener(e -> soldierClick(xxButton01, xxJPanel01,a , b, map));
148         if (sP.getTeam() == 'A'){
149           xxButton01.setBackground(new Color(173, 216, 230));
150         }
151         else if (sP.getTeam() == 'B'){
152           xxButton01.setBackground(new Color(255, 182, 193));
153         }
154       } else
155         xxButton01.setBackground(Color.WHITE);
156       xxJPanel01.add(xxButton01);
157     }
158   }
159   return xxJPanel01;
160 }
```

- `createPanelWE(ArrayList<Soldado>ej)`:
- Con este método se crean los paneles del este y oeste del JFrame.
- Estos paneles contienen los nombres y la vida de los soldados del ejército recibido.
- De esta manera los jugadores pueden ver con más claridad sus soldados creados.

```
162 public static JPanel creatPanelWE(ArrayList<Soldado> ej) {
163   JPanel panelWE = new JPanel();
164   panelWE.setLayout(new BoxLayout(panelWE, BoxLayout.Y_AXIS));
165
166   for (Soldado s : ej) {
167     JButton soldierButton = new JButton();
168     soldierButton.setLayout(new BorderLayout());
169     soldierButton.setBorder(BorderFactory.createCompoundBorder(
170       BorderFactory.createLineBorder(Color.BLACK),
171       BorderFactory.createEmptyBorder(10, 20, 10, 20)
```

```
172       ));
173       soldierButton.setBackground(Color.WHITE);
174       JLabel soldierLabel = new JLabel(s.getType() + " - HP: " + s.getActualLife());
175       soldierLabel.setFont(new Font("Courier New", Font.PLAIN, 14));
176       soldierButton.add(soldierLabel, BorderLayout.CENTER);
177       panelWE.add(soldierButton);
178       panelWE.add(Box.createRigidArea(new Dimension(0, 10)));
179     }
180     return panelWE;
181   }
```

- `soldierClick(JButton buttonClicked, JPanel bo, Ejercito a, Ejercito b, Mapa map)`:
- Este método tiene la finalidad de realizar el combate entre los soldados seleccionados.
- Se usa un botón atacante y uno atacado, luego con el método fight() se decide el ganador.

```
183   public static void soldierClick(JButton buttonClicked, JPanel bo, Ejercito a, Ejercito b,
          Mapa map){
184     if(buttonSel == null)
185       buttonSel = buttonClicked;
186     else {
187       JButton attacker = buttonSel;
188       JButton attacked = buttonClicked;
189
190       fight(bo, attacker, attacked, a, b, map);
191       buttonSel = null;
192     }
193   }
```

- `changeTurn(boolean turnA)`:
- Al ser llamado su función es retornar el booleano opuesto al ya obtenido, para de esta manera tener controlado los turnos de jugadores.

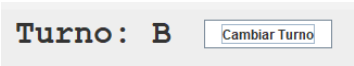

Figura 3

```
195   public static boolean changeTurn(boolean turnA){
196     return !turnA;
197   }
```

- `fight(JPanel game, JButton attacker, JButton attacked, Ejercito a, Ejercito b, Mapa m)`:
- Haciendo uso del nivel de vida de los soldados de la pelea, se usan condicionales para determinar el ganador y perdedor.
- el perdedor es borrado del tablero, mientras que el ganador en caso se pueda, va a la posición del derrotado o se queda en la misma si fue el atacado.

```
199 public static void fight(JPanel game, JButton attacker, JButton attacked, Ejercito a,
        Ejercito b, Mapa m){
200   int kerH = (int)attacker.getClientProperty("s");
201   int kedH = (int)attacked.getClientProperty("s");
202
203   //metrica especial
204
205   if (kerH < kedH){
206     //attacker.setVisible(false);
207     attacker.setBackground(Color.WHITE);
208     attacker.setText(null);
209     int coords = (int)attacker.getClientProperty("p");
210     attacker.putClientProperty("s", null);
211     attacker.putClientProperty("p", null);
212     m.editDeleteBoard(coords / 10, coords % 10);
213   }
214   else if (kedH < kerH){
215     //attacker.setVisible(false);
216     attacked.setBackground(Color.WHITE);
217     attacked.setText(null);
218     int coords = (int)attacked.getClientProperty("p");
219     attacked.putClientProperty("s", null);
220     attacked.putClientProperty("p", null);
221     m.editDeleteBoard(coords / 10, coords % 10);
222   }
223
224   if (endWar(m.getBoard())){
225     showWinner(m.getBoard());
226   }
227 }
```

- `endWar(Soldado[][] bo)`:
- Su única función es verificar que alguno de los dos ejércitos se quede sin soldados, en caso pase devuelve true.

```
233 public static boolean endWar(Soldado[][] bo){
234   int a = 0;
235   int b = 0;
236   for (Soldado so[] : bo){
237     for (Soldado s: so) {
238       if (s != null){
239         a += s.getTeam() == ('A') ? 1 : 0;
240         b += s.getTeam() == ('B') ? 1 : 0;
241       }
242     }
243   }
244   return a == 0 || b == 0;
245 }
```

- `showWinner(Soldado[][] bo)`:
- El método tiene la misma lógica del anterior, pero en este caso muestra en pantalla al ejército ganador y finaliza el programa.

```
247 public static void showWinner(Soldado[][] bo){
248   int a = 0;
249   int b = 0;
250   for (Soldado so[] : bo){
251     for (Soldado s: so) {
252       if (s != null){
253         a += s.getTeam() == ('A') ? 1 : 0;
254         b += s.getTeam() == ('B') ? 1 : 0;
255       }
256     }
257   }
258   String ejWin = a > b ? "A" : "B";
259   JOptionPane.showMessageDialog(null, "Gano el Ejercito" + ejWin, "Ganador",
          JOptionPane.INFORMATION_MESSAGE);
260   System.exit(0);
261 }
```

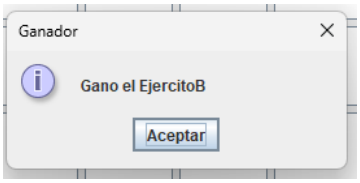

Figura 4

4.2. Clase Mapa.java

Listing 2: Commit 379f06f: Terminada la clase mapa

```
$ git add .
$ git commit -m "Terminada la clase mapa"
$ git push -u origin main
```

4.2.1. Imports

- `import java.util.Arrays;`:
- `import java.util.List;`:

```
1 import java.util.Arrays;
2 import java.util.List;
```

4.2.2. Atributos

- `static Mapa instanciaSingleton`: Crea el mapa de juego haciendo uso de Singleton.
- `String[] typesTerritory`: Contiene todos los posibles territorios que se pueden tener.
- `String territory`: Contiene el tipo de territorio del mapa actual.
- `Soldado[][] board`: Contiene el tablero de soldados de ambos ejércitos.

```
4  private static Mapa instanciaSingleton;
5  private String[] typesTerritory ={"Bosque","CampoAbierto", "Montaa", "Desierto", "Playa"};
6  private String territory;
7  private Soldado[][] board = new Soldado[10][10];
```

4.2.3. Métodos

- `getInstance()`:
- Usamos Singleton para no crear nuevos mapas y asi usar la misma instancia cuando se quiera jugar de nuevo.

```
9  private Mapa(){}
10
11 public static Mapa getInstance() {
12   if (instanciaSingleton == null)
13     instanciaSingleton = new Mapa();
14   return instanciaSingleton;
15 }
```

- `Getters y Setters`:
- Además del método para generar un territorio random.

```
9   public void setTerritory(String territory) {
10    List<String> typesTerritory2 = Arrays.asList(typesTerritory);
11    if (typesTerritory2.contains(territory))
12      this.territory = territory;
13    else{
14      //Mensaje error
15    }
16  }
17
18  public String getTerritory() {
19    return territory;
20  }
21
22  public void setBoard(Soldado[][] board){
23    this.board = board;
24  }
25
26  public Soldado[][] getBoard(){
27    return board;
28  }
29
30  public void setRandomMapa (){
31    this.territory = typesTerritory[(int) (Math.random() * typesTerritory.length)];
32  }
```

- `addSoldier(Soldado s, int r, int c)`:
- Sirve para agregar soldados al tablero del mapa.

```
42  public void addSoldier(Soldado s, int r, int c){
43    board[r][c] = s;
44  }
```

- `checkSoldier(int r, int c)`:
- Verifica si existe o no un soldado en la posición recibida.

```
46  public boolean checkSoldier(int r, int c){
47    return board[r][c] != null;
48  }
```

- `editDeleteBoard(int r, int c)`:
- Elimina soldados del tablero, útil para los enfrentamientos donde los soldados perdedores se eliminan.

```
50  public void editDeleteBoard(int r, int c){
51    board[r][c] = null;
52  }
```

4.3. Clase Ejercito.java

Listing 3: Commit 94bd50a: Redefiniendo la clase Ejercito, adaptando y reorganizando su contenido

```
$ git add .
$ git commit -m "Redefiniendo la clase Ejercito, adaptando y reorganizando su contenido"
$ git push -u origin main
```

4.3.1. Imports

- `import java.util.ArrayList;`

```
1  import java.util.ArrayList;
```

4.3.2. Atributos

- `String kingdom`: Contiene el tipo de reino que le tocó al ejército.
- `char name`: Contiene el char identificador ('A' o 'B').
- `ArrayList<Soldado>army`: Guarda a los soldados creados.

```
3  private String kingdom;
4  private char name;
5  private ArrayList<Soldado> army = new ArrayList<Soldado>();
```

4.3.3. Métodos

- `Ejercito(Mapa map, char name)`:
- El constructor de la clase Ejercito.
- Dentro se elije el reino aleatoriamente, además de crear los soldados y almacenarlos.

```
7  public Ejercito(Mapa map, char name){
8    String[] reinos = {"Inglaterra", "Francia","Castilla-Aragon",
9                       "Moros", "Sacro Imperio Romano-Germanico"};
10   kingdom = reinos[(int) (Math.random() * reinos.length)];
11
12   int n = (int) (Math.random() * 10) + 1;
13   for (int i = 0; i < n; i++) {
14     Soldado s = createSoldier(map, this, name, i);
15     army.add(s);
16     map.addSoldier(s, s.getRow(), s.getCol());
17   }
18   if (benefits(map)){
19     for (Soldado s : army){
20       s.setActualLife(s.getActualLife() + 1);
21       s.setLifeLevel(s.getLifeLevel() + 1);
22     }
23   }
24 }
```

- Getters y Setters:

```
30  public void setKingdom(String kingdom){
31    this.kingdom = kingdom;
32  }
33
34  public String getKingdom(){
35    return kingdom;
36  }
37
38  public void setName(char name){
39    this.name = name;
40  }
41
42  public char getName(){
43    return name;
44  }
45
46  public void setArmy(ArrayList<Soldado> army){
47    this.army = army;
48  }
49
50  public ArrayList<Soldado> getArmy(){
51    return army;
52  }
```

- createSoldier(Mapa map, Ejercito ej, char nameEj, int i):
- Este método es muy extenso porque verifica dentro que tipo de soldado se debe crear, va desde los especiales y verificando el reino para crearlos.
- En cada caso llama al constructor del soldado que se debe y lo retorna.

```
54  public Soldado createSoldier(Mapa map, Ejercito ej, char nameEj, int i ){
55    int row, col;
56    do {
57      row = (int) (Math.random() * 10);
58      col = (int) (Math.random() * 10);
59    } while (map.checkSoldier(row, col));
60
61    String[] typeSoldier = {"Espadachin", "Caballero", "Arquero", "Lancero"};
62    int typ = (int) (Math.random() * typeSoldier.length);
63    switch (typ) {
64      case 0 -> {
65        switch (ej.getKingdom()) {
66          case ("Inglaterra") ->{
67            return new EspadachinReal(map, ej, row, col, (char) (col + 'A'),
68            row * 10 + col, i, ej.getKingdom(), nameEj);
69          }
70          case ("Castilla-Aragon") ->{
71            return new EspadachinConquistador(map, ej, row, col, (char) (col + 'A'),
72            row * 10 + col, i, ej.getKingdom(), nameEj);
73          }
74          case ("Sacro Imperio Romano-Germanico") ->{
```

```
75            return new EspadachinTeutonico(map, ej, row, col, (char) (col + 'A'),
76            row * 10 + col, i, ej.getKingdom(), nameEj);
77          }
78          default -> {
79            return new Espadachin(map, ej, row, col, (char) (col + 'A'),
80            row * 10 + col, i, ej.getKingdom(), nameEj);
81          }
82        }
83      }
84      case 1 -> {
85        switch (ej.getKingdom()) {
86          case ("Francia") ->{
87            return new CaballeroFranco(map, ej, row, col, (char) (col + 'A'),
88            row * 10 + col, i, ej.getKingdom(), nameEj);
89          }
90          case ("Moros") ->{
91            return new CaballeroMoro(map, ej, row, col, (char) (col + 'A'),
92            row * 10 + col, i, ej.getKingdom(), nameEj);
93          }
94          default -> {
95            return new Caballero(map, ej, row, col, (char) (col + 'A'),
96            row * 10 + col, i, ej.getKingdom(), nameEj);
97          }
98        }
99      }
100     case 2 -> {
101       return new Arquero(map, ej, row, col, (char) (col + 'A'),
102         row * 10 + col, i, ej.getKingdom(), nameEj);
103     }
104     case 3 -> {
105       return new Lancero(map, ej, row, col, (char) (col + 'A'),
106       row * 10 + col, i, ej.getKingdom(), nameEj);
107     }
108   }
109   return null;
110 }
```

- `benefits(Mapa m)`:
- El método se encarga de devolver un booleano, verificanso si los soldados del reino merecen o no una bonificación de vida.

```
112 public boolean benefits(Mapa m){
113   String t = m.getTerritory();
114   switch (kingdom) {
115     case ("Inglaterra") -> {
116       if (t.equals("Bosque"))
117         return true;
118     }
119     case ("Francia") -> {
120       if (t.equals("Campo Abierto"))
121         return true;
122     }
123     case ("Castilla-Aragon") -> {
124       if (t.equals("Montaa"))
125         return true;
126     }
127     case ("Moros") -> {
128       if (t.equals("Desierto"))
129         return true;
130     }
131     case ("Sacro Imperio Romano-Germanico") -> {
132       if (t.equals("Bosque") || t.equals("Playa") ||
133           t.equals("Campo Abierto"))
134         return true;
135     }
136   }
137   return false;
138 }
```

4.4. Clase Soldado.java

Listing 4: Commit 87c0666: Redefiniendo la clase Soldado, adaptando y reorganizando su contenido

```
$ git add .
$ git commit -m "RRedefiniendo la clase Soldado, adaptando y reorganizando su contenido"
$ git push -u origin main
```

4.4.1. Atributos

- Contiene los atributos que se requieren, agregando adicionales para un mejor manejo de datos en diferentes situaciones.

```
2   private String name; //this
3   private int row; //this
4   private int col; //this
5   private char columnC; //this
6   private char team; //this
7   private int position; //this
8
9   private int attackLevel;
10  private int defenseLevel;
11  private int lifeLevel;
12  private int actualLife;
13  //private int speed; //~quitar
14  //private String attitude;//~quitar
15  private boolean lives; //this
16  private String type;
17  private String kingdom; //this
```

4.4.2. Métodos

- `Soldado()`:
- Contiene este constructor el cual incluso podría eliminarse, ya que no se crean soldados desde está clase, sino desde las clases hijas.

```
19  public Soldado() {
20
21  }
```

4.5. Clase Espadachin.java

Listing 5: Commit ff50555: Concluyendo las clases de los Tipos de Soldado[H

```
$ git add .
$ git commit -m "Concluyendo las clases de los Tipos de Soldado"
$ git push -u origin main
```

4.5.1. Atributos

- `int longEspada`: Contiene el tamaño de la espadad del Espadachin.

```
2  private int longEspada;
```

4.5.2. Métodos

- `Espadachin(Mapa map, Ejercito ej, int row, int col, char columnC, int position, int i, String kingdom, char team)`:
- El método constructor el cual recibe los parámetros necesarios para la creación del objeto, además de que en el interior se crean otros atributos aleatorios como el nivel de vida.

```
10 public Espadachin(Mapa map, Ejercito ej, int row, int col, char columnC, int position,
11                 int i, String kingdom, char team) {
12   setRow(row);
13   setCol(col);
14   setColumnC(columnC);
15   setPosition(position);
16   setName("Soldado" + team + i);
17   setkingdom(kingdom);
18   setTeam(team);
19   setLives(true);
20
21   setAttackLevel(10);
22   setDefenseLevel(8);
23
24   int life = (int) (Math.random() * (10 - 8 + 1) + 8);
25   setLifeLevel(life);
26   setActualLife(life);
27   setType("Espadachin");
28 }
```

4.6. Clase EspadachinConquistador.java

4.6.1. Atributos

- `Hachas hachas`: Contiene las hachas que posee el Espadachin Conquistador.
- `byte nivelEvolucion`: Guarda el nivel de evolución.

```
2  private Hachas hachas;
3  private byte nivelEvolucion;
```

4.6.2. Métodos

- `EspadachinConquistador(Mapa map, Ejercito ej, int row, int col, char columnC, int position, int i, String kingdom, char team)`:
- El método constructor el cual recibe los parámetros necesarios para la creación del objeto, además de que en el interior se crean otros atributos como las hachas.

```
5  public EspadachinConquistador(Mapa map, Ejercito ej, int row, int col, char columnC, int
       position,
6                                int i, String kingdom, char team) {
7    setRow(row);
8    setCol(col);
9    setColumnC(columnC);
10   setPosition(position);
11   setName("Soldado" + team + i);
12   setkingdom(kingdom);
13   setTeam(team);
14   setLives(true);
15
16   hachas = new Hachas();
17   setActualLife(14);
18   setLifeLevel(14);
19   setType("Espadachin Conquistador");
20 }
```

- `evolucion()`:
- El método que controla los niveles de evolución.

```
22 public void evolucion() {
23   if (nivelEvolucion < 4){
24     if (hachas.getCantidad() < 4)
25       hachas.setCantidad(hachas.getCantidad() + 1);
26
27     if (hachas.getTamao() < 5)
28       hachas.setTamao(hachas.getTamao() + 1);
29
30     if (nivelEvolucion < 3)
31       nivelEvolucion++;
32   }
33 }
```

4.7. Clase EspadachinReal.java

4.7.1. Atributos

- `Cuchillos cuchillos`: Contiene los cuchillos que posee el Espadachin Real.
- `byte nivelEvolucion`: Guarda el nivel de evolución.

```
2  private Cuchillos cuchillos;
3  private byte nivelEvolucion;
```

4.7.2. Métodos

- `EspadachinReal(Mapa map, Ejercito ej, int row, int col, char columnC, int position, int i, String kingdom, char team)`:
- El método constructor el cual recibe los parámetros necesarios para la creación del objeto, además de que en el interior se crean otros atributos como los cuchillos.

```
5  public EspadachinReal(Mapa map, Ejercito ej, int row, int col, char columnC, int position,
6                        int i, String kingdom, char team) {
7    setRow(row);
8    setCol(col);
9    setColumnC(columnC);
10   setPosition(position);
11   setName("Soldado" + team + i);
12   setkingdom(kingdom);
13   setTeam(team);
14   setLives(true);
15
16   cuchillos = new Cuchillos();
17   setActualLife(12);
18   setLifeLevel(12);
19   setType("Espadachin Real");
20 }
```

- `evolucion()`:
- El método que controla los niveles de evolución.

```
22 public void evolucion() {
23   if (nivelEvolucion < 4){
24     if (cuchillos.getCantidad() < 4)
25       cuchillos.setCantidad(cuchillos.getCantidad() + 1);
26
27     if (cuchillos.getTamao() < 5)
28       cuchillos.setTamao(cuchillos.getTamao() + 1);
29
30     if (nivelEvolucion < 3)
31       nivelEvolucion++;
32   }
33 }
```

4.8. Clase EspadachinTeutonico.java

4.8.1. Atributos

- `Jabalinas jabalinas`:
- `byte nivelEvolucion`:

```
2  private Jabalinas jabalinas;
3  private byte nivelEvolucion;
```

4.8.2. Métodos

- `EspadachinTeutonico(Mapa map, Ejercito ej, int row, int col, char columnC, int position, int i, String kingdom, char team)`:
- El método constructor el cual recibe los parámetros necesarios para la creación del objeto, además de que en el interior se crean otros atributos como los cuchillos.

```
5  public EspadachinTeutonico(Mapa map, Ejercito ej, int row, int col, char columnC, int
       position,
6                             int i, String kingdom, char team) {
7    setRow(row);
8    setCol(col);
9    setColumnC(columnC);
10   setPosition(position);
11   setName("Soldado" + team + i);
12   setkingdom(kingdom);
13   setTeam(team);
14   setLives(true);
15
16   jabalinas = new Jabalinas();
17   setActualLife(13);
18   setLifeLevel(13);
19   setType("Espadachin Teutonico");
20 }
```

- `evolucion()`:
- El método que controla los niveles de evolución.

```
22 public void evolucion() {
23   if (nivelEvolucion < 4){
24     if (jabalinas.getCantidad() < 4)
25       jabalinas.setCantidad(jabalinas.getCantidad() + 1);
26
27     if (jabalinas.getTamao() < 5)
28       jabalinas.setTamao(jabalinas.getTamao() + 1);
29
30     if (nivelEvolucion < 3)
31       nivelEvolucion++;
32   }
33 }
```

4.9. Clase Caballero.java

Listing 6: Commit ff50555: Concluyendo las clases de los Tipos de Soldado[H

```
$ git add .
$ git commit -m "Concluyendo las clases de los Tipos de Soldado"
$ git push -u origin main
```

4.9.1. Atributos

- `String arma`: Contiene el arma del Caballero.
- `boolean montado`: Guarda el estado, si está montado o no.

```
2  private String arma;
3  private boolean montado;
```

4.9.2. Métodos

- `public Caballero(Mapa map, Ejercito ej, int row, int col, char columnC, int position, int i, String kingdom, char team)`:
- El método constructor el cual recibe los parámetros necesarios para la creación del objeto, además de que en el interior se crean otros atributos aleatorios como el nivel de vida.

```
9  public Caballero(Mapa map, Ejercito ej, int row, int col, char columnC, int position,
10                 int i, String kingdom, char team) {
11   setRow(row);
12   setCol(col);
13   setColumnC(columnC);
14   setPosition(position);
15   setName("Soldado" + team + i);
16   setkingdom(kingdom);
17   setTeam(team);
18   setLives(true);
19
20   setAttackLevel(13);
21   setDefenseLevel(7);
22   int life = (int) (Math.random() * (12 - 10 + 1) + 10);
23   setLifeLevel(life);
24   setActualLife(life);
25   setType("Caballero");
26 }
```

- Métodos de las acciones especiales del Caballero:

```
27  public void alternarArma(){
28    if (arma.equals("espada"))
29      arma = "lanza";
30    else
31      arma = "espada";
32  }
33
34  public void desmontar(){
35    if (montado){
36      montado = false;
37      arma = "espada";
38    }
39  }
40
41  public void montar(){
42    if (!montado){
43      montado = true;
44      arma = "lanza";
45    }
46  }
```

4.10. Clase CaballeroFranco.java

4.10.1. Atributos

- `Lanzas lanzas`: Contiene las lanzas del Caballero Franco.
- `byte nivelEvolucion`: Guarda el nivel de evolución.

```
2  private Lanzas lanzas;
3  private byte nivelEvolucion;
```

4.10.2. Métodos

- `public CaballeroFranco(Mapa map, Ejercito ej, int row, int col, char columnC, int position, int i, String kingdom, char team)`:
- El método constructor el cual recibe los parámetros necesarios para la creación del objeto, además de que en el interior se crean otros atributos especiales como las Lanzas.

```
5  public CaballeroFranco(Mapa map, Ejercito ej, int row, int col, char columnC, int position,
6                         int i, String kingdom, char team) {
7    setRow(row);
8    setCol(col);
9    setColumnC(columnC);
10   setPosition(position);
11   setName("Soldado" + team + i);
12   setkingdom(kingdom);
13   setTeam(team);
14   setLives(true);
15
16   lanzas = new Lanzas();
17   setActualLife(15);
18   setLifeLevel(15);
19   setType("Caballero Franco");
20 }
```

- `evolucion()`:
- El método que controla los niveles de evolución.

```
22 public void evolucion() {
23   if (nivelEvolucion < 4){
24     if (lanzas.getCantidad() < 4)
25       lanzas.setCantidad((byte) (lanzas.getCantidad() + 1));
26
27     if (lanzas.getTamao() < 5)
28     lanzas.setTamao((byte) (lanzas.getTamao() + 1));
29
30     if (nivelEvolucion < 3)
31       nivelEvolucion++;
32   }
33 }
```

4.11. Clase CaballeroMoro.java

4.11.1. Atributos

- `Flechas flechas`: Contiene las flechas que posee el Caballero Moro.

```
2  private Flechas flechas;
3  private byte nivelEvolucion;
```

4.11.2. Métodos

- `public CaballerMoro(Mapa map, Ejercito ej, int row, int col, char columnC, int position, int i, String kingdom, char team)`:
- El método constructor el cual recibe los parámetros necesarios para la creación del objeto, además de que en el interior se crean otros atributos especiales como las Flechas.

```
5  public CaballeroMoro(Mapa map, Ejercito ej, int row, int col, char columnC, int position,
6                       int i, String kingdom, char team) {
7    setRow(row);
8    setCol(col);
9    setColumnC(columnC);
10   setPosition(position);
11   setName("Soldado" + team + i);
12   setkingdom(kingdom);
13   setTeam(team);
14   setLives(true);
15
16   flechas = new Flechas();
17   setActualLife(12);
18   setLifeLevel(12);
19   setType("Caballero Moro");
20 }
```

- `evolucion()`:
- El método que controla los niveles de evolución.

```
22 public void evolucion() {
23   if (nivelEvolucion < 4){
24     if (flechas.getCantidad() < 4)
25       flechas.setCantidad(flechas.getCantidad() + 1);
26
27     if (flechas.getTamao() < 5)
28       flechas.setTamao(flechas.getTamao() + 1);
29
30     if (nivelEvolucion < 3)
31       nivelEvolucion++;
32   }
33 }
```

4.12. Clase Arquero.java

Listing 7: Commit ff50555: Concluyendo las clases de los Tipos de Soldado[H

```
$ git add .
$ git commit -m "Concluyendo las clases de los Tipos de Soldado"
$ git push -u origin main
```

4.12.1. Atributos

- `Flechas flechas`: Guarda las flechas que contiene el Arquero.

```
2  private Flechas flechas;
```

4.12.2. Métodos

- `public Arquero(Mapa map, Ejercito ej, int row, int col, char columnC, int position, int i, String kingdom, char team)`:
- El método constructor el cual recibe los parámetros necesarios para la creación del objeto, además de que en el interior se crean otros atributos especiales como las Flechas.

```
9  public Arquero(Mapa map, Ejercito ej, int row, int col, char columnC, int position,
10                 int i, String kingdom, char team) {
11   setRow(row);
12   setCol(col);
13   setColumnC(columnC);
14   setPosition(position);
15   setName("Soldado" + team + i);
16   setkingdom(kingdom);
17   setTeam(team);
18   setLives(true);
19
20   setAttackLevel(7);
21   setDefenseLevel(3);
22   int life = (int) (Math.random() * (5 - 3 + 1) + 3);
23   setLifeLevel(life);
24   setActualLife(life);
25   setType("Arquero");
26 }
```

4.13. Clase Lancero.java

Listing 8: Commit ff50555: Concluyendo las clases de los Tipos de Soldado[H

```
$ git add .
$ git commit -m "Concluyendo las clases de los Tipos de Soldado"
$ git push -u origin main
```

4.13.1. Atributos

- `int longLanza`:

```
2  private int longLanza;
```

4.13.2. Métodos

- `public Lancero(Mapa map, Ejercito ej, int row, int col, char columnC, int position, int i, String kingdom, char team)`:
- El método constructor el cual recibe los parámetros necesarios para la creación del objeto, además de que en el interior se crean otros atributos especiales como las Flechas.

```
9  public Lancero(Mapa map, Ejercito ej, int row, int col, char columnC, int position,
10                 int i, String kingdom, char team) {
11   setRow(row);
12   setCol(col);
13   setColumnC(columnC);
14   setPosition(position);
15   setName("Soldado" + team + i);
16   setkingdom(kingdom);
17   setTeam(team);
18   setLives(true);
19
20   setAttackLevel(5);
21   setDefenseLevel(10);
22   int life = (int) (Math.random() * (8 - 5 + 1) + 5);
23   setLifeLevel(life);
24   setActualLife(life);
25   setType("Lancero");
26 }
```

4.14. Clase Armas.java

Listing 9: Commit 204e9a3: Agregando nuevas clases para un mejor manejo de las armas de cada unidad especial[H

```
$ git add .
$ git commit -m "Agregando nuevas clases para un mejor manejo de las armas de cada unidad
    especial"
$ git push -u origin main
```

4.14.1. Atributos

- `int tamaño`:
- `int cantidad`:

```
2  private int tamao;
3  private int cantidad;
```

4.14.2. Métodos

- `Getters y Setters`:

```
6  public void setTamao(int tamao) {
7    this.tamao = tamao;
8  }
9
10 public int getTamao() {
11   return tamao;
12 }
13
14 public void setCantidad(int cantidad) {
15   this.cantidad = cantidad;
16 }
17
18 public int getCantidad() {
19   return cantidad;
20 }
```

4.15. Clase Cuchillos.java

Listing 10: Commit 204e9a3: Agregando nuevas clases para un mejor manejo de las armas de cada unidad especial[H

```
$ git add .
$ git commit -m "Agregando nuevas clases para un mejor manejo de las armas de cada unidad
    especial"
$ git push -u origin main
```

4.15.1. Métodos

- Constructor:

```
2  public Cuchillos() {
3     setTamao(1);
4    setCantidad(1);
5  }
```

4.16. Clase Flechas.java

Listing 11: Commit 204e9a3: Agregando nuevas clases para un mejor manejo de las armas de cada unidad especial[H

```
$ git add .
$ git commit -m "Agregando nuevas clases para un mejor manejo de las armas de cada unidad
    especial"
$ git push -u origin main
```

4.16.1. Métodos

- Constructor:

```
2  public Flechas() {
3     setTamao(1);
4    setCantidad(1);
5  }
```

4.17. Clase Hachas.java

Listing 12: Commit 204e9a3: Agregando nuevas clases para un mejor manejo de las armas de cada unidad especial[H

```
$ git add .
$ git commit -m "Agregando nuevas clases para un mejor manejo de las armas de cada unidad
    especial"
$ git push -u origin main
```

4.17.1. Métodos

- Constructor:

```
2  public Hachas() {
3    setTamao(1);
4   setCantidad(1);
5  }
```

4.18. Clase Jabalinas.java

Listing 13: Commit 204e9a3: Agregando nuevas clases para un mejor manejo de las armas de cada unidad especial[H

```
$ git add .
$ git commit -m "Agregando nuevas clases para un mejor manejo de las armas de cada unidad
    especial"
$ git push -u origin main
```

4.18.1. Métodos

- Constructor:

```
2  public Jabalinas() {
3    setTamao(1);
4   setCantidad(1);
5  }
```

4.19. Clase Lanzas.java

Listing 14: Commit 204e9a3: Agregando nuevas clases para un mejor manejo de las armas de cada unidad especial[H

```
$ git add .
$ git commit -m "Agregando nuevas clases para un mejor manejo de las armas de cada unidad
    especial"
$ git push -u origin main
```

4.19.1. Atributos

- `byte tamaño`: Contiene el tamaño de las lanzas.
- `byte cantidad`: Contiene el número de lanzas que quedan disponibles.

```
2  private byte tamao;
3  private byte cantidad;
```

4.19.2. Métodos

- Constructor:

```
5  public Lanzas() {
6     tamao  = 1;
7    cantidad = 1;
8  }
```

- Getters y Setters:

```
10 public void setTamao(byte tamao) {
11   this.tamao  =  tamao ;
12 }
13
14 public byte getTamao() {
15   return  tamao ;
16 }
17
18 public void setCantidad(byte cantidad) {
19   this.cantidad = cantidad;
20 }
21
22 public byte getCantidad() {
23   return cantidad;
24 }
```

4.20. Diagrama de clase UML

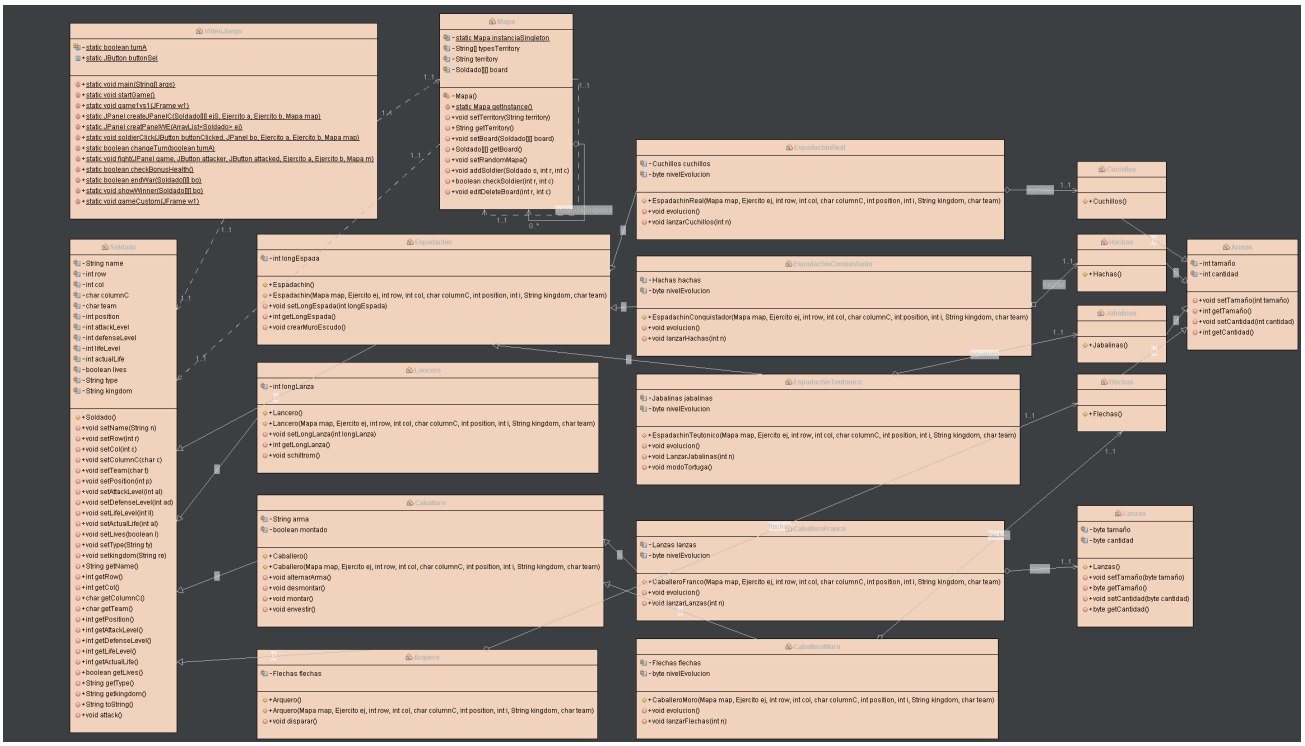

Figura 5

Diagrama 22 UML para acceder al diagrama UML del repositorio y se observe con más claridad.

4.21. Estructura de laboratorio 22

- El contenido que se entrega en este laboratorio es el siguiente:

```
lab20
|   Soldado.java
|   VideoJuego.java
|
|------latex
    |   Informe_Lab22.pdf
    |   Informe_Lab22.tex
    |
    |------img
    |       22uml.png
    |       logo_episunsa.png
    |       logo_abet.png
    |       logo_unsa.jpg
    |       vj1.png
    |       vj2.png
    |       vj3.png
    |       vj4.png
    |       vj5.png
    |
    |------src
        |   Armas.java
        |   Arquero.java
        |   Caballero.java
        |   CaballeroFranco.java
        |   CaballeroMoro.java
        |   Cuchillos.java
        |   Ejercito.java
        |   Espadachin.java
        |   EspadachinConquistador.java
        |   EspadachinReal.java
        |   EspadachinTeutonico.java
        |   Flechas.java
        |   Hachas.java
        |   Jabalinas.java
        |   Lancero.java
        |   Lanzas.java
        |   Mapa.java
        |   Soldado.java
        |   VideoJuego.java
        |------src
               01game.png
```

5. Rúbricas

5.1. Entregable Informe

Tabla 1: Tipo de Informe

Informe	
Latex	El informe está en formato PDF desde Latex, con un formato limpio (buena presentación) y fácil de leer.

5.2. Rúbrica para el contenido del Informe y demostración

- El alumno debe marcar o dejar en blanco en celdas de la columna **Checklist** si cumplió con el ítem correspondiente.
- Si un alumno supera la fecha de entrega, su calificación será sobre la nota mínima aprobada, siempre y cuando cumpla con todos lo ítem.
- El alumno debe auto calificarse en la columna **Estudiante** de acuerdo a la siguiente tabla:

Tabla 2: Niveles de desempeño

	Nivel			
Puntos	Insatisfactorio 25 %	En Proceso 50 %	Satisfactorio 75 %	Sobresaliente 100 %
2.0	0.5	1.0	1.5	2.0
4.0	1.0	2.0	3.0	4.0

Tabla 3: Rúbrica para contenido del Informe y demostración

Contenido y demostración		Puntos	Checklist	Estudiante	Profesor
1. GitHub	Hay enlace URL activo del directorio para el laboratorio hacia su repositorio GitHub con código fuente terminado y fácil de revisar.	2	X	2	
2. Commits	Hay capturas de pantalla de los commits más importantes con sus explicaciones detalladas. (El profesor puede preguntar para refrendar calificación).	4	X	4	
3. Código fuente	Hay porciones de código fuente importantes con numeración y explicaciones detalladas de sus funciones.	2	X	2	
4. Ejecución	Se incluyen ejecuciones/pruebas del código fuente explicadas gradualmente.	2	X	2	
5. Pregunta	Se responde con completitud a la pregunta formulada en la tarea. (El profesor puede preguntar para refrendar calificación).	2	X	2	
6. Fechas	Las fechas de modificación del código fuente están dentro de los plazos de fecha de entrega establecidos.	2	X	2	
7. Ortografía	El documento no muestra errores ortográficos.	2	X	2	
8. Madurez	El Informe muestra de manera general una evolución de la madurez del código fuente, explicaciones puntuales pero precisas y un acabado impecable. (El profesor puede preguntar para refrendar calificación).	4	X	3	
Total		20		19	

6. Referencias

- https://docs.oracle.com/javase%2F7%2Fdocs%2Fapi%2F%2F/javax/swing/package-summary.html
- https://docs.oracle.com/javase/tutorial/java/IandI/subclasses.html
- https://docs.oracle.com/javase/tutorial/java/IandI/polymorphism.html
- https://docs.oracle.com/javase/7/docs/api/java/awt/package-summary.html
- https://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html