

*Fight Knights*

# Memoria: Desarrollo de un videojuego en Java

**17 de Mayo de 2018**

*Alejandro Fernández Maceira*

*Oscar Fernández Sánchez*

*Adrián Palma Lima*

*Daniel López Moreno*

*Pedro Elías Cabada*

## Índice.

1. Introducción e historia. ....	3
1.2. Metodología/Herramientas .....	3
2. Diseño del videojuego. ....	3
2.1. Requisitos mínimos. ....	3
2.2. Objetivo del juego. ....	3
2.3. Lógica del juego. ....	4
2.4. Controles. ....	4
2.5. Interfaz. ....	4
2.6. Patrones de diseño. ....	5
3. Implementación. ....	5
4. Conclusión. ....	7



La pantalla principal.

## **1. Introducción e historia**

Fight Knights es un videojuego de cartas en 2D por turnos con perspectiva top-down.

En la época medieval, dos caballeros luchan por ver quién es el elegido para salvar el reino de la amenaza que se avecina. Para ello, lucharán en combate singular valiéndose de sus cartas y de su estrategia.

Solo uno será el vencedor. ¿Quién se alzará con la victoria?

### **1.2. Metodología/Herramientas**

Para la edición de las cartas, el tablero y demás elementos gráficos se han usado programas como Photoshop, Paint, Tiled etc. Para la creación del propio juego, la librería Slick2D del lenguaje de programación Java ha sido empleada.

## **2. Diseño del videojuego**

El tipo de videojuego elegido es cartas basado en fantasía medieval con perspectiva top-down. Es un videojuego con un concepto simple, pensado para el gran público, cuyas normas son claras y no requiere una gran curva de aprendizaje.

### **2.1. Requisitos mínimos**

Los requisitos mínimos de hardware para jugar al juego no son muy elevados, cualquier ordenador actual puede ejecutarlo sin problemas de estabilidad ni cuelgues en el equipo, ya que, al no tener inteligencia artificial, el uso de CPU es escaso.

En cuanto a los requisitos software, un entorno de desarrollo compatible con Java es necesario para la ejecución, como por ejemplo NetBeans. No se necesita otro software específico, pues la librería Slick2D ya viene incluida en el propio proyecto.

### **2.2. Objetivo del juego**

El objetivo del videojuego es ser el vencedor en el combate entre caballeros. Para ello, las cartas que se invocan en la mesa atacarán al jugador enemigo y le restarán puntos de vida en función de su ataque. Gana el jugador que consiga llevar a 0 los puntos de vida del contrario.

## 2.3. Lógica del juego

Los elementos del juego son:

- a) Jugadores. Representados por su salud en un orbe azul, es el avatar del usuario. Solo dispone de la vida actual, y para jugar deberá usar las cartas de su mazo.
- b) Unidades. Representadas por cartas, son los objetos con los que los jugadores combatirán. Cada jugador solo puede tener 5 cartas en su mano y 5 cartas en el tablero como máximo en todo momento. Cada unidad tiene características específicas, como curar vida al jugador o eliminar cartas del mazo del oponente. Sin embargo, para usar una carta, el jugador necesita energía.
- c) Energía. Elemento necesario para usar cartas. Los jugadores empiezan su turno con 1 de energía, y cada turno se incrementará en uno. En cada carta está indicado el coste en energía, al usar la carta, la energía se restará del contador.
- d) Tablero. Donde se colocarán las unidades de cada carta. Está representado con un estilo medieval, y en él se muestra la información de los jugadores y las cartas seleccionadas, así como botones para rendirse, salir, reiniciar o pasar turno.

El juego comienza en un turno aleatorio, puede tocarle al jugador 1 o al 2. Se mostrarán 3 cartas en el mazo, y el jugador decide lo que hacer.

La partida termina cuando un jugador quiera rendirse, salir o uno de los dos llegue a 0 puntos de vida.

## 2.4. Controles

El método de control es con el ratón. Se seleccionarán los cuadros de texto haciendo clic con el ratón, al igual que las cartas y el ataque. Para pausar la partida, basta con pulsar la tecla de escape y el juego se detendrá.

## 2.5. Interfaz

Al ejecutar el programa, aparece una pantalla de carga. Después aparecerá un menú con la opción de pulsar start. Se clic en el cuadro de texto y empieza la partida. En el tablero hay varias opciones:



Captura de un momento del juego.

- Reiniciar. Al pulsarse, reiniciará la partida y generará nuevas cartas para los jugadores.
- Rendirse. Declara la victoria al jugador contrario.
- Salir. Sale del juego.
- Pasar turno. Termina el turno del jugador actual y empieza el del siguiente. Aumenta en 1 la energía del nuevo jugador.

Para dar una ambientación medieval característica al juego, se reproduce un hilo musical con inspiración medieval. Si se quiere desactivar el sonido, en el menú inicial hay una opción para ello.

El juego se reproduce a 1280x720 de resolución, en modo ventana, para que sea compatible con la mayoría de pantallas actuales. En todo momento se muestran los FPS, para comprobar que todo funciona correctamente.

## 2.6. Patrones de diseño

Para la realización del videojuego se han empleado los siguientes patrones de diseño de videojuegos:

-Singleton. Para la creación de una sola instancia de partida. Este patrón de diseño permite que solo haya activa una instancia de la clase Match. Con esto se evita posibles pérdidas de información o escritura en instancias equivocadas. Para

implementar el patrón, se ha creado un constructor privado, al que se accede mediante un método público y que no permite ser llamado más de una vez, garantizando la unidad de la instancia.

-Factory. Para la creación de las cartas. Este patrón de diseño permite crear instancias de distintos tipos de cartas fácilmente. Se ha implementado creando un switch en el constructor de la clase Unit. Este constructor recibe un String como parámetro, el nombre de la carta, y busca en el switch el nombre que coincide. Una vez encontrado, inicializa los atributos de la clase con el valor establecido dentro del código case del switch.

```

26 public Unit(String card_name) throws SlickException {
27     switch (card_name) {
28         case "1":
29             this.name = "Contramaestre";
30             this.cost = 4;
31             this.initial_damage = 3;
32             this.initial_health = 3;
33             this.damage = this.initial_damage;
34             this.health = this.initial_health;
35             this.type = "Minion";
36             this.attackExecuted = true;
37             this.rutaImagenTablero = new Image("res/1.png");
38             this.rutaImagenMesa = new Image("res/r1.png");
39             break;
40         case "2":
41             this.name = "Caballero Sagrado";
42             this.cost = 6;
43             this.initial_damage = 5;
44             this.initial_health = 6;
45             this.damage = this.initial_damage;
46             this.health = this.initial_health;
47             this.type = "Minion";
48             this.attackExecuted = true;
49             this.guard = true;
50             this.rutaImagenTablero = new Image("res/2.png");
51             this.rutaImagenMesa = new Image("res/r2.png");
52             break;

```

Fragmento del patrón Factory.

### 3. Implementación

El juego está basado en estados, cada ventana es un estado diferente, y se ejecuta desde la clase MainEstados. Los métodos básicos render, update e init se han implementado. En init se inicializan las variables necesarias en cada clase, en update se actualiza la posición del ratón y si se ha hecho clic o no, y en render se generan las imágenes de las cartas, del tablero y demás elementos visuales.

La clase MainEstados contiene la definición de cada pantalla/estado y el método main desde el que se ejecutará la aplicación. También se define el tamaño de pantalla y la resolución.

La clase PantallaCarga representa la primera pantalla que verá el jugador. Contiene el logo del equipo y un indicador para comprobar el estado de la carga del juego. No hay ninguna interacción con el jugador en esta pantalla. De esta clase se pasa a la siguiente, Menú.

La clase Menú es el menú principal del juego, donde el jugador elegirá empezar la partida. Contiene el logo del juego. Para capturar el ratón se emplea el método `isMousePressed()` de la clase `Input` de `Slick2D`. Si el ratón está pulsado y se encuentra dentro de las coordenadas del cuadro de texto dibujado, entonces empieza la partida. No empezará hasta que se pulse.

La clase `Unit` representa una unidad. El constructor es un switch en el que se inicializan las cartas según su número. También contiene métodos que emplean las cartas como `initSkill`, la habilidad que tienen al ponerse en la mesa; `attackSkill`, la habilidad al atacar; o `deathSkill`, la habilidad al morir. Según la carta, estas habilidades realizan una acción u otra.

```

577 // AL CREAR LA UNIDAD SE EJECUTA INITSKILL Y ALIVESKILL PARA COMPROBAR
578 // SI ESA CARTA TIENE EFECTOS DE ESOS.
579 public void initSkill() throws SlickException // Efecto que se ejecuta al invocar la carta.
580 {
581     switch (name) {
582         case "Bola de Fuego": {
583             Match match = Match.getMatchInstance();
584             int danio_incrementado;
585
586             if (match.getP_turn() == 1) {
587                 danio_incrementado = match.getP1_spellIncrement();
588             } else {
589                 danio_incrementado = match.getP2_spellIncrement();
590             }
591
592             if (match.getP_turn() == 1)
593             {
594                 for (Unit unit : match.getP2_table())
595                 {
596                     unit.setHealth(unit.getHealth() - (1 + danio_incrementado));
597                 }
598             }
599             else
600             {
601                 for (Unit unit : match.getP1_table())
602                 {
603                     unit.setHealth(unit.getHealth() - (1 + danio_incrementado));
604                 }
605             }
606
607             //target.setHealth(target.getHealth() - (this.damage + danio_incrementado));
608             break;
609         }
610         case "Clerigo": {
611             Match match = Match.getMatchInstance();
612             if (match.getP_turn() == 1)
613             {
614                 match.setP1_health(match.getP1_health() + 5);
615             }
616             else
617             {
618                 match.setP2_health(match.getP2_health() + 5);
619             }
620             //target.setHealth(target.getHealth() + 5);
621             break;
622         }
623     }

```

Fragmento del método `initSkill`.

En el fragmento superior puede verse la estructura seguida para la habilidad inicial de las cartas. Por ejemplo, la bola de fuego resta uno de vida a todas las unidades enemigas teniendo en cuenta el nivel de hechizos, como puede se observa en las llamadas a `setHealth` con

La clase `Match` emplea el patrón de diseño singleton, creando una única instancia de la partida, con todos los valores necesarios para su funcionamiento: los mazos de cada jugador, representados con un `arrayList`, el turno, la vida de los jugadores, su energía, las unidades en el tablero, etc. También se incluyen métodos `get` y `set` de cada atributo establecido.

La clase `GameMethods` contiene los métodos necesarios para el correcto funcionamiento de las cartas y sus acciones. Un método es cambiar turno, en el que se

aumenta la energía del jugador siguiente, se suma una carta a su mano y cambia el turno al siguiente. Otro método es `executeAttack`, donde se comprueban los valores necesarios para que una carta ataque a otra, teniendo en cuenta si existe un efecto de provocación activo en ese momento. Para ello, se resta la vida de la carta objetivo en tantos puntos como ataque tenga la carta atacante.

También está en esta clase el método para atacar al jugador, en el que se pulsa la carta atacante y se selecciona el orbe azul del contrario. Después se resta tanta vida como ataque tenga la carta. El último método es `invokeCard`, donde se comprueba que el jugador tiene energía, y después se elimina la carta del mazo y se coloca en el tablero.

La clase `PantallaJuego` es donde se desarrolla la acción del videojuego. Aquí se crean y se usan todos los métodos y atributos de las clases anteriores, para el correcto desarrollo de la partida. Lo más importante es la captura de coordenadas del ratón. Según la posición en la que se encuentre, se realiza una acción u otra. También se realiza la lógica de los botones y su acción, como reiniciar, pausar o salir. Para usar una carta, primero se comprueba si el ratón está sobre esa carta, luego se obtienen datos de la carta y se arrastra al tablero, automáticamente se colocará en una posición disponible, o en ninguna si no hay espacio. Si se desplaza el cursor sobre las cartas del mazo, se ampliará la imagen para ver sus características. Dentro del tablero, cada unidad tiene a sus lados unos orbes azul y rojo que representan la vida y el ataque, respectivamente. Luego se captura la zona de actuación del ratón, y se actúa según la entrada del jugador. Si pulsa en la carta y luego en el orbe de vida del contrario, se atacará. Una carta solo puede atacar una vez por turno, y si hay alguna carta con efecto de provocación deberá ser atacada. Cuando la partida acaba, bien porque el jugador eligió rendirse, o porque la vida de alguno llegó a 0, se pasa a la siguiente clase.

La clase `VictoryScreen` se muestra cuando se cumple una de las circunstancias anteriores. Contiene la información sobre quién es el ganador de la partida y dos botones, uno para salir, y otro para volver a jugar.





La pantalla de victoria.

La clase PantallaPausa se muestra cuando, dentro del juego principal, un jugador pulsa la tecla escape del teclado. La partida se detendrá y se reanudará una vez que se haga clic en cualquier parte de la pantalla.

### 3.1. Estructura del proyecto

El proyecto se divide en varios paquetes. Por un lado, están los relativos a la librería de Slick2D, que no se han modificado. Por otro se encuentran los paquetes FightKnights, en los que se han incluido las clases creadas para el desarrollo del videojuego.

Para una mayor facilidad en la localización de los recursos, todas las imágenes y el audio se han incluido en el paquete res. El último paquete es fonts, que contiene las fuentes empleadas para la visualización de los diversos textos.

## 4. Conclusión

Fight Knights es un juego de cartas medieval desarrollado en Java con el motor Slick2D y que permite a dos jugadores en el mismo ordenador jugar por turnos. Tiene una interfaz amigable y una gran carga lógica y matemática.

## 5. Bibliografía y fuentes de apoyo

Para la parte de código se ha consultado el manual de Java de Oracle y los foros de Stack Overflow, así como el manual de Slick2D y los videos de YouTube del usuario Makigas.

En la parte de diseño artístico, las cartas han sido creadas por nosotros, cortesía de Adrián Palma Lima. El tablero también lo hemos creado nosotros, gracias a Pedro Elías Cabada. La música la hemos cogido del artista Kevin Mc Leod, sin copyright. El estilo de letra es Hellgrazer de la página [www.dafont.com](http://www.dafont.com).

## **6. División del trabajo**