

Juana de Arco en el Castillo de las Montañas de Reims



Juana de Arco (también conocida como la Doncella de Orléans, o en francés, la Pucelle), fue una heroína militar nacida en la región de la Lorena, Francia. Desde muy jovencita se vio metida en los tumultuosos hechos guerreros de la Francia de principios del siglo XV luchando a favor del reino de Carlos VII. Entre numerosas hazañas, Juana de Arco supo dar ánimos a sus compatriotas y batalló contra los ingleses, que ocupaban en aquel momento la ciudad de Reims y alrededores. (Wikipedia).

Deberás guiar a nuestra guerrera Juana de Arco por el castillo situado de las Montañas de Reims y descifrar el enigma que según cuenta la leyenda destruirá el maleficio que ha permitido a los ingleses apoderarse de él. El castillo sirve de morada para el general Brayan (súbdito de Enrique VI, rey de Inglaterra) y sus soldados ingleses, los cuales han preparado innumerables peligros y trampas para evitar ser invadidos. ¡Cuidado! durante el juego Juana de Arco deberá comer regularmente para no morir de inanición.

En tus manos te dejamos el inicio del juego que te permitirá controlar los movimientos de la guerrera protagonista y guiarla por los rincones del castillo mediante las flechas de dirección del cursor, 'm' para pausar el juego para pensar por donde seguir y 'ESC' para salir.

Después de probarlo deberás acabarlo. Aquí podrás aplicar lo que has aprendido durante estos meses, modificando el código fuente del juego que te proporcionamos para cumplir con tus objetivos.

El código base

El código base (ProjecteJuanaArco.zip en el campus virtual) es el conjunto de clases que te proporcionamos para poder tener el juego funcionando con un mínimo de utilidades, incluida la que contiene el main. Fíjate en el estilo y documentación del código proporcionado y documenta apropiadamente tus propias clases.

Estructuras de datos

El **castillo** está formado por varias plantas, cada una de ellas con un número cualquiera de salas (o habitaciones). Cada una de las habitaciones sobre las que se desarrolla el juego se divide en celdas de un tamaño fijo formando una cuadrícula de 25 x 17 celdas, sobre una celda podemos situar un muro, una puerta o nada (espacio libre de obstáculos).

La definición de cada **habitación** se realiza mediante un fichero de texto plano que contiene determinados caracteres: 'P' puerta, 'X' muro, '0' (cero) suelo. Cada carácter corresponde a una de

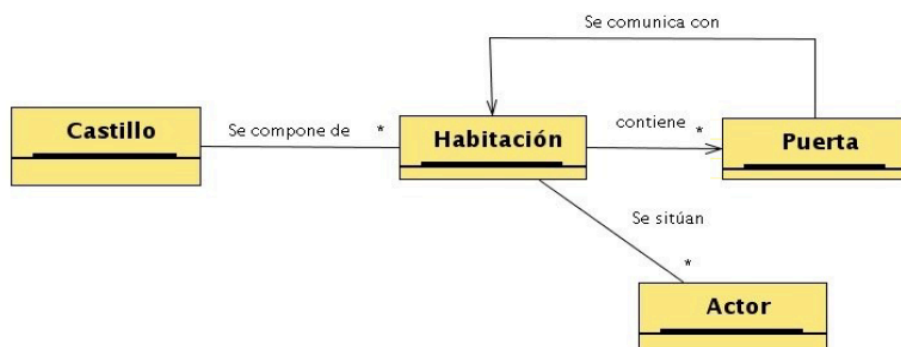
las celdas de la cuadrícula. Los ficheros deben situarse en el directorio "src/recursos".
 Aquí podemos ver el contenido del fichero "h0_0.txt" que corresponde con una de las habitaciones del programa demostración que se incluye:

```

X,X,X,X,X,X,X,X,X,X,P,X,X,X,X,X,X,X,X,X,X,X,X
X,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,X
X,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,X
X,0,0,0,0,0,0,0,0,0,0,0,0,0,X,X,X,X,X,X,X,X,0,0,X
X,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,X,0,0,X
X,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,X,0,0,X
X,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,X,0,0,X
X,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,X,0,0,X
X,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,X,0,0,X
X,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,X,0,0,X
X,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,X,0,0,X
X,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,X,0,0,X
X,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,X,0,0,X
X,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,X,0,0,X
X,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,X,0,0,P
X,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,X,0,0,X
X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X
    
```

Podemos ver que contiene un muro periférico y 2 **puertas**, una situada al norte y otra al sureste. Así como un pasillo en forma de L invertida.

En cada sala (o habitación) se situarán diversos objetos como alimentos, flechas, gemas, monedas, etcétera, a los que llamaremos **Actores**. El siguiente diagrama muestra como se relacionan todos los conceptos de los que hemos tratado hasta ahora:

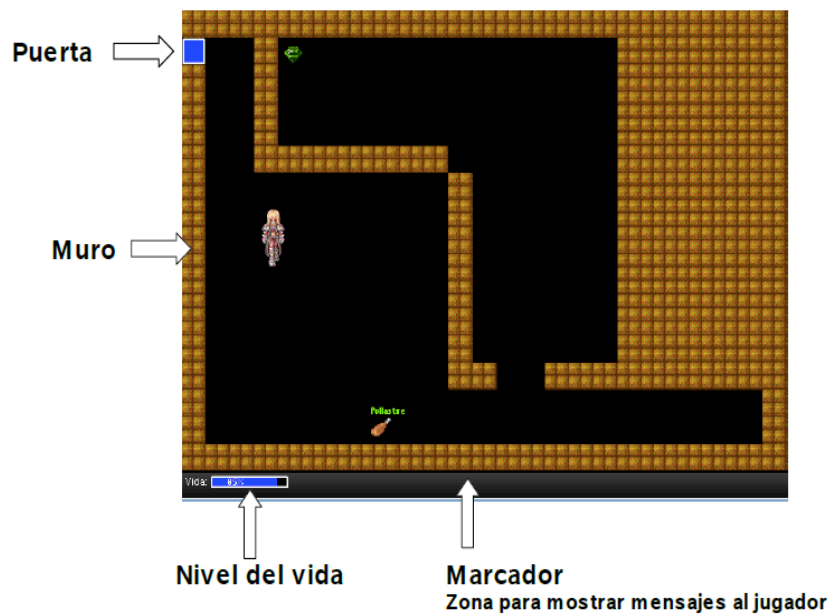


Fíjate que las puertas pueden comunicarse con cualquier habitación – no necesariamente con las habitaciones de la misma sala.




A continuación tenemos la pantalla de bienvenida del juego:



Debajo tenemos una captura en un momento del juego, correspondiente al mapa que hemos dibujado anteriormente (h0_0.txt):



Personajes

	Juana de Arco: Esta doncella es la guerrera protagonista de la historia. Está dispuesta a combatir con los soldados ingleses que se han apoderado del castillo de Reims, dónde teme que está secuestrado el hijo de Carlos VII.
	Duque de Anjou: Es el hijo predilecto del rey francés de la época. Ha sido secuestrado por guerrillas inglesas. El rey ha encargado a Juana de Arco rescatarlo sano y salvo cuanto antes.
	General Brayan: Temido general militar responsable del secuestro del duque. Ha raptado al hijo del rey con el fin de chantajearlo a cambio de más tierras y riquezas.



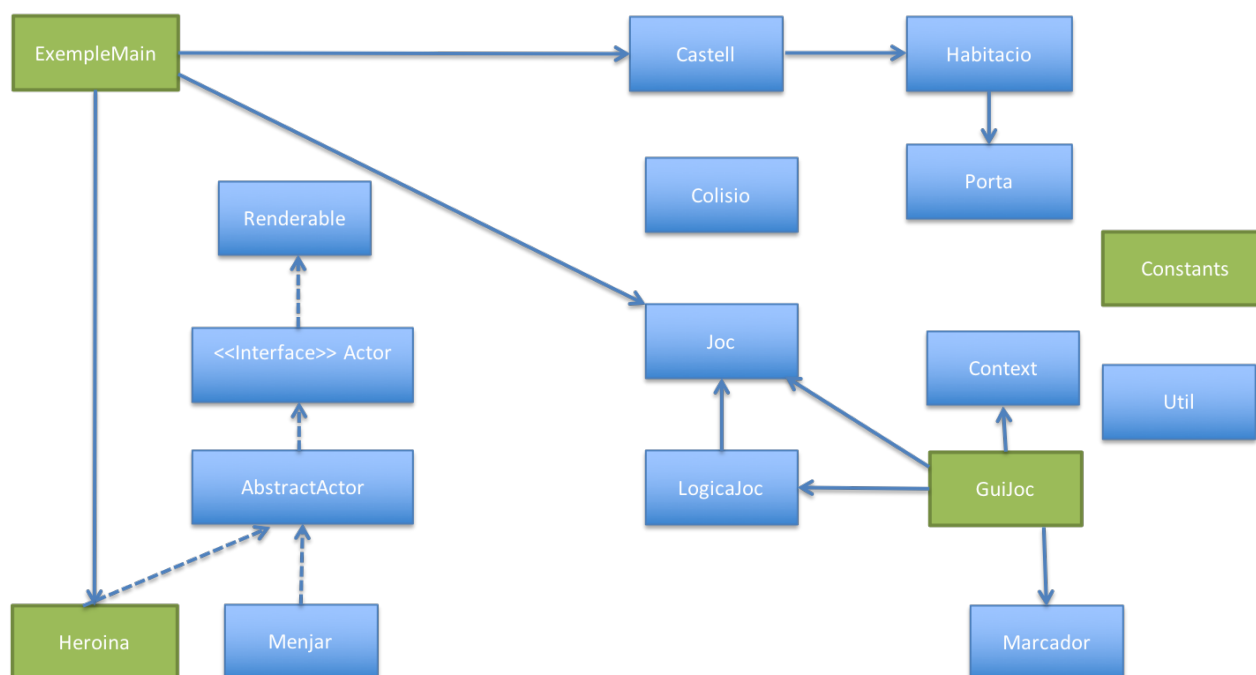
Soldados Ingleses: Estos soldados son los más formados del ejército inglés y han sido contratados para vigilar y proteger el castillo de Reims. Están escondidos en cualquier parte del castillo ¡y no dudarán en atacarte si te descubren!

Estructura del código

El juego consta de distintas partes funcionales:

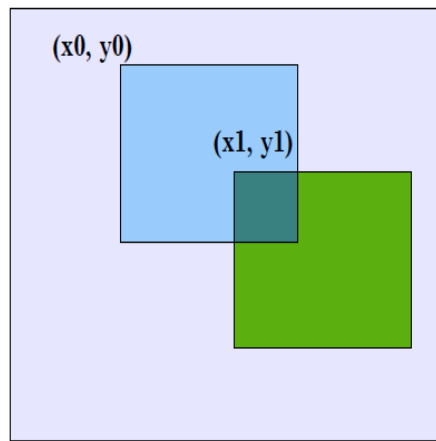
- **El motor del juego:** controla la animación y coordina las acciones que se han de realizar en cada momento. Este motor se encuentra en la clase `Joc.java`.
- **La lógica del juego:** que establece unos estados: Menú, Jugar, Fin de Juego (GameOver) y Salida (Exit). Esta lógica del juego se encuentra en la clase `LogicaJoc.java`.
- **Los tipos de datos:** son las clases que representan los datos sobre los que se desarrolla el juego: `Castell`, `Habitacio`, `Joc`, `Marcador`; y las clases que modelan cada uno de los objetos (Actores) que aparecen en el juego: `Heroína`, `Menjar`.
- **Los recursos:** las imágenes que se usan para crear los escenarios (podéis encontrar más en: <http://www.iconarchive.com>, <http://www.veryicon.com>, <http://www.iconseeker.com>, <http://images.google.com/images?q=sprite+sheet>, ...). Las imágenes se guardan en la carpeta `src/recursos`.

Los rectángulos en verde son las clases que tienes que modificar.



Descripción del funcionamiento interno del juego

La animación se ejecuta varias veces por segundo (la frecuencia es de unos 60 frames por segundo), para cada frame los actores actualizan su posición y estado, y pueden interactuar con otros actores, modificando alguno de sus atributos. Por ejemplo, el objeto **Menjar** interacciona con nuestra heroína aumentando su nivel de vida cada vez que "colisionan". Para detectar las colisiones, cada actor definirá su dimensión (ancho y alto en píxeles) como un rectángulo situado en una posición X, Y. Dos actores en las posiciones (x0, y0) y (x1, y1) colisionarán cuando sus rectángulos delimitadores interseccionen en algún punto:



Cuando se detecta alguna colisión de este tipo, el método `tractarColisio(Colisio colisio)` de cada actor implicado es ejecutado, de forma que el actor podrá modificar su estado, si es necesario. La definición completa de la interfaz `Actor` es la siguiente:

```
public void inicialitzar();
public void setPosicioInicial(int x, int y);
public int[] getPosicioInicial();
public void setPosicio(int x, int y);
public int[] getPosicio();
public void actualitzar(Context context);
public Rectangle getLimits();
public void tractarColisio(Colisio colisio);
public float getVida();
public void setVida(float nivell);
public int getEstat();
public void setEstat(int estat);
```

Consulta la documentación para conocer más sobre cada método.

Cada escena del juego transcurre en una única planta y sala. La clase `Castell` proporciona todos los métodos necesarios para modificar la sala y planta actuales, así como para añadir y acceder a las habitaciones.

```
public void addHabitacio(int planta, int numHabitacio, Habitacio habitacio)
public void setPlanta(int planta)
public int getPlanta()
public void setNumHabitacio(int habitacio)
public int getNumHabitacio()
public Habitacio getHabitacio()
public Habitacio[] getHabitacions(int numPlanta)
public int getNumPlantes()
public int getNumHabitacions(int planta)
```

Como podéis imaginar la clase `Castell` contiene un atributo del tipo `tabla` que guarda la estructura de las habitaciones que hay en cada piso:

```
// habitacions es una matriz de plantas y las habitaciones que hay en cada
// planta: el número de habitaciones es el mismo en todas las plantas
private Habitacio[ ][ ] habitacions;
...
// en el constructor de Castell se realiza la siguiente instrucción:
habitacions = new Habitacio[numPlantes][numHabitacionsPerPlanta];
```

Para comunicar las habitaciones entre sí utilizaremos la clase `Porta`, que contiene la habitación y puerta con la que se comunica.

Ejemplo

A título de ejemplo, se proporciona una clase principal `ExempleMain.java` que crea un juego con 3 habitaciones (salas) y algunos objetos de comida distribuidos por ellas. Hay tres tipos de comida: “pizza”, “pollastre”, “síndria”. Como se puede ver en el método `distribuirMenjar`, inicialmente se sitúa un número aleatorio de comida de los diferentes tipos en cada habitación. Además, en el código que tenéis a continuación se muestra uno de los métodos que construye la habitación inicial:

```
private ExempleMain() {
    castell = new Castell(1, 3); //atribut definit a la classe de tipus Castell
    castell.addHabitacio(0, 0, crearHabitacio0Planta0());
    . . .

    heroina = new Heroina(); //atribut definit a la classe de tipus Heroina
    Habitacio h = castell.getHabitacio(0, 0);
    int[] p = h.getPosicioCela(10, 10);
    heroina.setPosicioInicial(p[0], p[1]);

    distribuirMenjar();

    // inicialització del joc
    joc = new Joc(castell, heroina);
    GuiJoc gui = new GuiJoc(joc);
}

private Habitacio crearHabitacio0Planta0() {
    Habitacio h = Util.carregarHabitacio("h0_0.txt");

    Porta porta = h.getPorta(14, 24);
    porta.setNumPlantaDesti(0);
    porta.setNumHabitacioDesti(1);
    porta.setPosicioHabitacioDesti(h.getPosicioCela(1, 1));

    porta = h.getPorta(0, 10);
    porta.setNumPlantaDesti(0);
    porta.setNumHabitacioDesti(2);
    porta.setPosicioHabitacioDesti(h.getPosicioCela(14, 10));

    return h;
}

private void distribuirMenjar() {
    String[] menjars = { "Pizza", "Pollastre", "Síndria" } ;
    // { calories,x,y,width,height } de cada imatge
    int[][] dades = {
        { 25, 540, 14, 30, 22 },
        { 50, 439, 14, 27, 23 },
        { 50, 97, 100, 30, 20 }
    };

    for (int i = 0; i < castell.getNumPlantes(); i++) {
        for (int j = 0; j < castell.getNumHabitacions(i); j++) {
            Habitacio h = castell.getHabitacio(i, j);
            int numMenjars = (int)(Math.random() * (MAX_MENJAR_PER_HABITACIO + 1));
            for (int k = 0; k < numMenjars; k++) {
                int[] cela = obtenirCelaLliure(h);
                int imenjar = (int)(Math.random() * menjars.length);
```

```

        Menjar m = new Menjar(menjars[imenjar],
                                dades[imenjar][0], dades[imenjar][1], dades[imenjar][2],
                                dades[imenjar][3], dades[imenjar][4]);

        int[] posicio = h.getPosicioCela(cela[0], cela[1]);
        m.setPosicioInicial(posicio[0], posicio[1]);
        h.addActor(m);
    }
}
}

```

Compilación y Ejecución

Descarga el fichero JuanaDeArco_codi_base.zip que encontrarás en el campus virtual, importa el proyecto desde Netbeans como fichero zip y ejecútalo.

Implementación

Seguid los 6 pasos descritos a continuación para implementar el juego.

PAS 1 – CLASE INICIAL. Crea un fichero con el nombre **Practica.java** que contenga un método **main**. Esta clase será al estilo de la clase de ejemplo `ExempleMain.java` que se proporciona. Ten en cuenta que entonces deberás compilar el fichero `Practica.java` y no `ExempleMain.java`, del proyecto Netbeans, como hasta ahora.

PAS 2 – CONFIGURACIÓN DEL TERRITORIO. Crea un castillo con al menos 3 plantas y 3 habitaciones por planta. Para cada habitación deberás crear un fichero `.txt` con la definición de la habitación. Si lo deseas, puedes copiar el fichero `"h0_2.txt"` que define una sala vacía y modificarlo según desees. Es importante que las puertas se comuniquen con celdas que no son muro – de lo contrario, la heroína no podrá moverse. Recuerda que las habitaciones se sitúan en el directorio `src/recursos`.

PAS 3 – CONFIGURACIÓN DE LAS HABITACIONES. En la clase `Practica.java`, crea las habitaciones y las puertas que comunican las habitaciones entre sí según desees. Ten en cuenta que la información del archivo `.txt` ha de ser coherente con las instrucciones que realizas en la clase `Practica.java`.

PAS 4 – VIDA. Crea un nuevo actor con el nombre de clase `Vida`. Si la heroína colisiona establece su vida a 100. Distribuye 2 vidas por el castillo. Puedes inspirarte en la clase `Menjar.java`, pero ten en cuenta que en este caso cada vida de una habitación se crea y se posiciona en una celda concreta definida al crearse la habitación misma.

PAS 5 – SERPIENTE. Crea un nuevo actor con el nombre de clase `Serp`. Si la heroína colisiona con una serpiente disminuye su vida en 50 unidades. Distribuye el número de serpientes que quieras por el castillo (mínimo 4) acorde con la distribución de los muros y puertas, ten cuidado en no situar una serpiente cerca de una puerta. Puedes inspirarte en la clase `Menjar.java`, pero ten en cuenta que en este caso cada serpiente de una habitación se crea y se sitúa en su sitio al crearse la habitación misma. Es decir, las posiciones de las serpientes en todas las partidas son fijas, preestablecidas al crearse la habitación.

PAS 6 – FINAL POR LLAVE. Haz que el juego se acabe cuando la protagonista haya recogido las cinco llaves que permiten abrir las diferentes puertas de la prisión y la celda donde está el duque de Anjou. En caso de colisionar con una llave, la protagonista se la guardará para usarla

posteriormente. Ten en cuenta que a la protagonista se le va agotando la vida mientras busca las llaves. Si su vida llega a 0, la protagonista pierde en el juego.

- Crea un nuevo actor en el paquete `edu.ub.juanadearco.actors` con el nombre de clase `Clau`. (Puedes inspirarte en el actor `Menjar.java`). Cada llave tiene un color según el material del que está hecha. Los diferentes tipos de llaves son: “Ferro”, “Bronze”, “Llauto”, “Plata” y “Or”. La llave de “Or” es una llave maestra que permite abrir todas las puertas de la prisión.
- En la clase `Practica`, posiciona las 5 llaves aleatoriamente de tipo “Ferro”, “Bronze”, “Llauto” y “Plata” en las habitaciones del castillo. Además, posiciona de forma aleatoria una única llave de tipo “Or”. (Puedes fijarte en el método `distribuirMenjar` de la clase `ExempleMain`).
- Añade el método `haTrobatLesClaus()` de la clase `Heroína` de manera que retorne `TRUE` si se ha encontrado las 5 llaves o la llave maestra.
- Añade también en la clase `Heroína` un método `addClau(Clau clau)` que permita a la heroína guardarse la llave. El método `addClau(Clau clau)` debería llamarse cuando la heroína colisiona con una llave. En este método es obligatorio guardar las llaves en una tabla.
- En la clase `GuiJoc`, cuando se actualiza el juego, se debería tener en cuenta si la heroína ha encontrado las llaves. Si la ha encontrado todas las llaves, el juego acabará con el mensaje: "HAS POGUT TROBAR LES CLAUS A TEMS!!!". Si se le acaba la vida a la protagonista, aparecerá el mensaje de "`* G A M E O V E R *`".

Ayudas

Demostración

Junto con el código os proporcionamos el fichero "JuanaDeArco.jar" que contiene una demo ejecutable con algunos ejercicios solucionados, para ejecutarla:

```
> java -jar JuanaDeArco.jar
```

Tabla de posición de imágenes

Nombre	Fichero	Posición X	Posición Y	Ancho	Alto
Clau	objectes.png	675	50	20	30
Heroína 1	heroína.png	0	0	25	50
Heroína 2	heroína.png	25	0	25	50
Heroína 3	heroína.png	50	0	25	50
Heroína 4	heroína.png	75	0	25	50
Menjar 1	objectes.png	540	14	30	22
Menjar 2	objectes.png	439	14	27	23
Menjar 3	objectes.png	97	100	30	20
Serp 1	objectes.png	700	0	40	42
Serp 2	objectes.png	700	42	40	42
Serp 3	objectes.png	700	84	40	42
Vida	objectes.png	174	12	26	24

Obtención de números aleatorios:

Para obtener un número aleatorio puedes utilizar la función **Math.random** de java, que retorna un **número mayor o igual a 0 y menor de 11**. Ejemplos:

```
// número aleatorio entre 0 y 10
int r = (int)(Math.random() * 11);

0
int r = (int)(Math.round(Math.random() * 10));
```