

# Civilizations



*Projecte AMS i AWS curs 23-24*

# Índice

<b>Civilizations.....</b>
Recursos.....
Edificios.....
Tecnologías.....
Unidades.....
Unidades ofensivas, de mejor a peor.....
Unidades defensivas de mejor a peor.....
Unidades especiales.....
<b>Clase Civilization.....</b>
Clase ResourceException.....
Clase BuildingException.....
Clase AttackUnity.....
Clase DefenseUnit.....
Clase SpecialUnit.....
Interface MilitaryUnit.....
<b>Clases Swordsman, Spearman, Crossbow, Cannon.....</b>
Clases ArrowTower, Catapult, RocketLauncherTower.....
Clases Magician, Priest.....
<b>Interfiecie Variables.....</b>
<b>Clase Battle.....</b>
Mecánica de la batalla:.....
<b>Clase Main.....</b>
Ayudas.....
<b>M02-Base de datos.....</b>
<b>M05-Entornos de desarrollo.....</b>
Diagrama de clases.....
Control de versiones.....

# Civilizations

En nuestra aplicación, vamos a estar al mando de una pequeña civilización. Dispondremos de recursos, tecnologías, edificios, defensas y unidades tanto de ataque como defensa.

Desgraciadamente, no vamos a poder vivir tranquilos, dado que hay otras civilizaciones que quieren conquistarnos i que nos van a atacar constantemente. Ésto nos va a obligar a cuidar las defensas de nuestra civilización para defendernos de los ataques continuos.

Elementos de nuestra civilización:

## Recursos

Comida, Madera, Hierro, Maná.

Nuestra civilización generará una cierta cantidad de recursos cada cierto tiempo, excepto maná, que solamente se generará en función de las torres mágicas que tengamos.

Habrá ciertas construcciones que incrementarán la generación de recursos de nuestra civilización. Aparte, en cada batalla se generará una cantidad de escombros (madera y Hierro) resultante de las unidades derrotadas.

## Edificios

Podremos construir edificios que nos permitirán generar un extra de recursos. Este extra de recursos puede ser modificado si se cree conveniente con el fin de crear un juego más equilibrado.

**Granjas:** Incrementan en un 10% la generación de comida de nuestra civilización.

**Carpintería:** Incrementan en un 10% la generación de madera de nuestra civilización.

**Herrería:** Incrementan en un 10% la generación de hierro de nuestra civilización.

**Torre Mágica:** Por defecto, sin torres mágicas, nuestra civilización no generará maná, será necesario crear al menos 1 torre mágica para ello.

Éstas también nos permitirán crear magos, Iglesias y sacerdotes.

**Iglesia:** Cada iglesia nos permitirá generar 1 sacerdote. Es decir, si queremos tener n sacerdotes, necesitaremos n iglesias.

Costes de cada edificio y generación de recursos:

	Cost				Plus generations resources			
	Food	Wood	Iron	Mana	Food	Wood	Iron	Mana
Smithy	5000	10000	12000					+5%
Carpentry	5000	10000	12000					+5%
Farm	5000	10000	12000					+5%
Magic_Tower	10000	20000	24000					3000
Church	10000	20000	24000	10000				

## Tecnologías

Las tecnologías nos servirán para crear mejores unidades, y así tener una ventaja sobre el ejército enemigo. Podemos avanzar en las tecnologías de **defensa**, que nos permitirá crear unidades con una defensa mejor, y la tecnología de **ataque**, que nos permitirá crear unidades con más poder de ataque.

Costes de cada tecnología:

	Cost		
	Food	Wood	Iron
Defense	100	200	2000
Attack	100	200	2000
Increase cost Defense	+10%	+15%	+20%
Increase cost Attack	+10%	+15%	+20%

## Unidades

### Unidades ofensivas, de mejor a peor

Espadachin: Se necesita comida y hierro para crearlos

Lancero: Se necesita comida y hierro para crearlos

Ballesta: Se necesita madera y hierro para crearlas

Cañon: Se necesita madera y hierro para crearlos

### Unidades defensivas de mejor a peor

Torre lanza: Se necesita madera para crearlas

Catapulta: Se necesita madera y hierro para crearlas

Torre lanzacohetes: Se necesita madera y hierro para crearlas

## Unidades especiales

**Mago:** Se necesita comida y maná para crearlos. Tienen un gran poder de ataque y una alta probabilidad de repetición, pero 0 de defensa.

**Sacerdote:** No tienen poder de ataque y tampoco defensa. Mientras haya un sacerdote en nuestro ejército, y maná suficiente, nuestras unidades actuales ( las que se crean después, no estarán santificadas) pueden ser santificadas, lo que mejorará los stats de nuestras unidades. Si mueren todos los sacerdotes de nuestro ejército, todas las unidades dejarán de estar santificadas. ( una posible mejora sería que cada sacerdote sólo pudiese santificar un número limitado de unidades)

La siguiente tabla recoge los costes y los stats de las unidades militares que podemos crear:

	Cost				Stats				Chance of Generate Wastings
	Food	Wood	Iron	Mana	Armor	Attack Power	Chance of Attack Again		
Swordsman	8000	3000	50	0	400	80	3	55	
Spearman	5000	6500	50	0	1000	150	7	65	
Crossbow	0	45000	7000	0	6000	1000	45	80	
Cannon	0	30000	15000	0	8000	700	70	90	
Arrow Tower	0	2000	0	0	200	80	5	55	
Catapult	0	4000	500	0	1200	250	12	65	
Rocket Launcher	0	50000	5000	0	7000	2000	30	75	
Magician	12000	2000	0	5000	0	3000	75	0	
Priest	15000	0	0	15000	0	0	0	0	

Tendremos que elegir convenientemente qué unidades formarán nuestro ejército, dado que cada unidad tendrá unas características diferentes que nos ayudará a que nuestras batallas sean más efectivas.

Vamos a describir cómo podemos implementar una aplicación que describa lo anterior.

# Clase Civilization

Necesitaremos una clase **Civilization**, que representará nuestra civilización. Que tendrá las siguientes características:

```
int technologyDefense ;
int technologyAttack ;

int wood;
int iron;
int food;
int mana;

int magicTower;
int church;
int farm;
int smithy;
int carpentry;

int battles;
ArrayList<MilitaryUnit> army = new ArrayList[9];
```

Las características de nuestro planeta son bastante intuitivas.

La característica más compleja es army, que representa nuestro ejército. Es un array de arrays, donde cada posición del array representa el conjunto de unidades del mismo tipo.

Army[0] → arrayList de Swordsman  
 Army[1] → arrayList de Spearman  
 Army[2] → arrayList de Crossbow  
 Army[3] → arrayList de Cannon  
 Army[4] → arrayList de Arrow Tower  
 Army[5] → arrayList de Catapult  
 Army[6] → arrayList de Rocket Launcher  
 Army[7] → arrayList de Magician  
 Army[8] → arrayList de Priest

Los métodos necesarios en la clase Civilization, además de los setters, getters y constructores, serán:

- **void newChurch,**
- **void newMagicTower,**
- **void newFarm,**
- **void newCarpentry,**
- **void new Smithy:**

Con estos métodos podemos crear una nueva iglesia o una nueva torre mágica. En caso de no tener recursos suficientes, lanzaremos una excepción del tipo **ResourceException**, que comentaremos más adelante.

- **void upgradeTechnologyDefense,**
- **void upgradeTechnologyAttack:**

Con estos métodos, pretendemos actualizar nuestras tecnologías de ataque/defensa, pero antes tendremos que comprobar que tenemos recursos suficientes para actualizar dicha tecnología. En caso de no tener recursos suficientes, lanzaremos una excepción del tipo **ResourceException**, que comentaremos más adelante.

Tenemos que tener en cuenta, que cada vez que subimos un nivel de tecnología, la siguiente actualización será un porcentaje establecido más caro.

Por ejemplo, si pasar del nivel 1 de defensa al nivel 2 de defensa costase 100 de Hierro, y el porcentaje establecido de incremento de precio fuese un 10%, pasar del nivel 2 al nivel 3 nos costaría 110 de hierro. Estos valores estaría previamente indicados en las características upgradeDefenseTechnologyIronCost, upgradeAttackTechnologyIronCost, upgradeDefenseTechnologyWoodCost, upgradeAttackTechnologyWoodCost = 0;

- **void newSwordsman(int n),**
- **void newSpearman(int n),**
- **void newCrossbow(int n),**
- **void newCannon(int n),**
- **void newArrowTower(int n),**
- **void newCatapult(int n),**
- **void newRocketLauncher(int n),**
- **void newMagician(int n),**
- **void newPriest(int n).**

Estos métodos servirán para añadir nuevas unidades militares a nuestro ejército army mencionado anteriormente.

Estos métodos reciben un entero n que indica el número de unidades que queremos añadir, si no tenemos suficientes recursos para añadir las unidades que queremos, lanzará una excepción del tipo ResourceException indicando el mensaje informativo. Pero se añadirán todas las unidades posibles que permitan nuestros recursos.

Es decir, si queremos añadir 10 swordsman, y sólo tenemos recursos para añadir 5, se lanzará una excepción del tipo ResourceException, pero se añadirán los 5 swordsman que podemos generar y se nos mostrará también un mensaje informativo indicando el número de swordsman que se han añadido.

En el caso de crear magos, si no tenemos al menos una torre mágica, lanzaremos una excepción del tipo BuildingException.

En el caso de crear sacerdotes, si no tenemos al menos una iglesia, lanzaremos una excepción del tipo BuildingException.

- **Void printStats().** Este método nos servirá para mostrar una visión del estado de nuestro planeta por consola, una posible salida cuando llamamos a este método podría ser:

```
*****CIVILIZATION STATS*****
```

```
-TECHNOLOGY-
```

Attack	Defense
0	0

```
-BUILDINGS-
```

Farm	Smithy	Carpentry	Magic Tower	Church
0	0	0	0	0

```
-DEFENSES-
```

Arrow Tower 10	Catapult 10	Rocket Launcher 9
-----ATTACK UNITS-----		
Swordsman 10	Spearman 10	Crossbow 10
-----ESPECIAL UNITS-----		
	Mague 0	Priest 0
-----RESOURCES-----		
Food 4870000	Wood 45000	Iron 429000
-----GENERATION RESOURCES-----		
Food 8000	Wood 5000	Iron 1500
		Mana 0

## Clase ResourceException

Para gestionar las excepciones cuando queramos crear algo y no dispongamos de los recursos necesarios, vamos a crear nuestra propia excepción, la clase **ResourceException**, que extenderá de Exception, y nos mostrará un mensaje descriptivo del motivo de la excepción.

## Clase BuildingException

Para gestionar las excepciones lanzadas cuando queramos construir alguna unidad especial que requiera de edificios, vamos a crear nuestra propia excepción, la clase **BuildingException**, que extenderá de Exception, y nos mostrará un mensaje descriptivo del motivo de la excepción.

## Clase AttackUnity

Para crear nuestra flota, es decir, unidades militares no defensivas, vamos a crear la clase abstracta AttackUnit.

Esta clase tendrá como propiedades:

```
int armor;
int initialArmor;
int baseDamage;
int experience
boolean sanctified
```

armor será la armadura restante de nuestra unidad durante las batallas.

InitialArmor será la armadura inicial de nuestra unidad, es decir, la armadura al momento de crearla.

BaseDamage el poder de ataque inicial, le llamaremos baseDamage.

Experience será la experiencia ganada por dicha unidad. Cada batalla, las unidades que queden en pie aumentarán en 1 su experiencia, lo que le derá una mejora en los stats.

Sanctified nos indicará si hemos santificado los tropas, en cuyo caso mejoraremos los stats de todas las unidades el porcentaje de los stats base en función de las características especificadas en la interfaz variables:

```
int PLUS_ARMOR_UNIT_SANCTIFIED ;
int PLUS_ATTACK_UNIT_SANCTIFIED;
```

Esta clase implementará las interfaces MilitaryUnit i Variables.

## Clase DefenseUnit

Unidades militares defensivas.

Esta clase tendrá como propiedades:

```
int armor;
int initialArmor;
int baseDamage;
int experience;
boolean sanctified;
```

armor será la armadura restante de nuestra unidad durante las batallas.

InitialArmor será la armadura inicial de nuestra unidad, es decir, la armadura al momento de crearla.

BaseDamage el poder de ataque inicial, le llamaremos baseDamage.

Experience será la experiencia ganada por dicha unidad. Cada batalla, las unidades que queden en pie aumentarán en 1 su experiencia, lo que le derá una mejora en los stats.

Sanctified nos indicará si hemos santificado los tropas, en cuyo caso mejoraremos los stats de todas las unidades el porcentaje de los stats base en función de las características especificadas en la interfaz variables:

```
int PLUS_ARMOR_UNIT_SANCTIFIED ;
int PLUS_ATTACK_UNIT_SANCTIFIED;
```

Esta clase implementará las interfaces MilitaryUnit i Variables.

## Clase SpecialUnit

Unidades especiales (Mago, Sacerdote).

Esta clase tendrá como propiedades:

```
int armor;
int initialArmor;
int baseDamage;
int experience;
```

armor será siempre cero para estas unidades.

InitialArmor será siempre cero.

En estas unidades se podría implementar algún tipo de mejora relacionado con la armadura, pero el propósito, es que sean unidades que puedan ser eliminadas fácilmente para dar equilibrio a la batalla.

BaseDamage el poder de ataque inicial, le llamaremos baseDamage.

Experience será la experiencia ganada por dicha unidad. Cada batalla, las unidades que queden en pie aumentarán en 1 su experiencia, lo que le derá una mejora en los stats.

Esta clase implementará las interfaces MilitaryUnit i Variables.

## Interface MilitaryUnit

MilitaryUnit será una interfaz que implementarán todas nuestras unidades (defensa y ataque), y que nos servirá para dos cosas, para gestionar todas las unidades militares por igual, y para obligarnos a implementar los métodos que serán comunes a todas las unidades militares.

La interfaz MilitaryUnit que tan sólo declarará los métodos:

- abstract int attack();  
Nos devolverá el poder de ataque que tenga la unidad.
- abstract void takeDamage(int receivedDamage);  
Restará a nuestro blindaje el daño recibido por parámetro.
- abstract int getActualArmor();  
Nos devolverá el blindaje que tengamos actualmente, después de haber recibido un ataque.
- abstract int getFoodCost();  
Nos devolverá el coste de Comida que tiene crear una nueva unidad.
- abstract int getWoodCost();  
Nos devolverá el coste de Madera que tiene crear una nueva unidad.
- abstract int getIronCost();  
Nos devolverá el coste de Hierro que tiene crear una nueva unidad.
- abstract int getManaCost();  
Nos devolverá el coste de Mana que tiene crear una nueva unidad.
- abstract int getChanceGeneratinWaste();  
Nos la probabilidad de generar residuos al ser totalmente eliminada (blindaje 0 o inferior).
- abstract int getChanceAttackAgain();  
Nos la probabilidad de volver a atacar.
- abstract void resetArmor();  
Nos restablecerá nuestro blindaje a su valor original.
- abstract void setExperience(int n);  
Establecerá la experiencia a un nuevo valor.
- abstract void getExperience();  
Nos devolverá la experiencia actual de la unidad.

Estos métodos se tendrán que implementar en todas las clases que hereden de AttackUnit.

# Clases Swordsman, Spearman, Crossbow, Cannon

Para cada tipo de unidad de ataque nuestro ejército, crearemos una clase que heredará de la clase AttackUnit.

Todas ellas dispondrán de dos constructores.

## Constructor 1:

Al que le pasaremos dos parámetros; int armor, int baseDamage.

Para establecer la armadura de nuestra unidad, armor, tendremos que tener en cuenta dos constantes, por ejemplo, en el caso de un Swordsman.

*ARMOR\_SWORDSMAN* i *PLUS\_ARMOR\_SWORDSMAN\_BY TECHNOLOGY*.

La primera es la armadura que tendrá nuestro swordsman en el caso que nuestra tecnología de defensa sea cero, y la segunda es el plus de armadura que tendrá en función del nivel de tecnología de defensa que tengamos.

Si *ARMOR\_SWORDSMAN*=1000, *PLUS\_ARMOR\_SWORDSMAN\_BY TECHNOLOGY*=5 y nuestro nivel de tecnología =2

La armadura de un nuevo swordsman será de  $1000 + (2*5)1000/100 = 1100$ .

Es decir, la armadura con un nivel de tecnología de defensa igual a cero, más el  $2*5 = 10\%$ .

En caso de tener un nivel de tecnología en defensa igual a 3, la armadura de un nuevo swordsman sería  $1000 + (3*5)%1000 = 1150$ .

Para establecer al poder de ataque baseDamage, tendremos que tener en cuenta las constantes, *BASE\_DAMAGE\_SWORDSMAN* i *PLUS\_ATTACK\_SWORDSMAN\_BY TECHNOLOGY*. El funcionamiento será el mismo que hemos descrito en el establecimiento de la armadura, pero ahora teniendo en cuenta el nivel de tecnología de ataque.

Por último, estableceremos la propiedad initialArmor al mismo valor que armor, que nos servirá para restablecer la armadura a su valor original después de una batalla.

## Constructor 2:

Al que no le pasaremos ningún parámetro.

Este constructor creará una nueva unidad y establecerá la armadura y el poder de ataque a sus valores básicos, sin tener en cuenta las tecnologías.

Este constructor nos servirá para crear unidades para un supuesto ejército enemigo desde nuestra clase principal Main.

Cada una de estas clases tendrá que implementar los métodos definidos en la interfaz MilitaryUnit.

# Clases ArrowTower, Catapult, RocketLauncherTower

Para cada tipo de unidad de nuestra defensa, crearemos una clase que heredará de la clase DefenseUnit.

Todas ellas dispondrán de un único constructor al que le pasaremos como argumentos int armor, int baseDamage.

El funcionamiento del constructor de nuestras defensas, será igual que el de nuestro ejército de ataque.

Se establecerán el blindaje y la armadura dependiendo del nivel de nuestras tecnologías.

Cada una de estas clases tendrá que implementar los métodos definidos en la interfaz MilitaryUnit.

## Clases Magician, Priest

Para cada tipo de unidad especial, crearemos una clase que heredará de la clase SpecialUnit.

Todas ellas dispondrán de un único constructor al que le pasaremos como argumentos int armor , int baseDamage.

El funcionamiento del constructor de nuestras unidades especiales, será igual que el de nuestro ejército de ataque.

Cada una de estas clases tendrá que implementar los métodos definidos en la interfaz MilitaryUnit.

# Interfieie Variables

Esta será una interfaz que nos permitirá parametrizar el juego, de forma que después de testearlo varias veces, encontremos unos valores equilibrados, en principio la podemos definir como sigue:

```
public interface Variables {
    // resources available to create the first enemy fleet

    public final int IRON_BASE_ENEMY_ARMY = 26000;
    public final int WOOD_BASE_ENEMY_ARMY = 180000;
    public final int FOOD_BASE_ENEMY_ARMY = 70000;

    // percentage increase of resources available to create enemy fleet
    public final int ENEMY_FLEET_INCREASE = 6;

    // resources increment every minute

    public final int CIVILIZATION_IRON_GENERATED = 1500;
    public final int CIVILIZATION_WOOD_GENERATED = 5000;
    public final int CIVILIZATION_FOOD_GENERATED = 8000;

    public final int CIVILIZATION_IRON_GENERATED_PER_SMITHY = (int) (0.5*CIVILIZATION_IRON_GENERATED);
    public final int CIVILIZATION_WOOD_GENERATED_PER_CARPENTRY = (int) (0.5*CIVILIZATION_WOOD_GENERATED);
    public final int CIVILIZATION_FOOD_GENERATED_PER_FARM = (int) (0.5*CIVILIZATION_FOOD_GENERATED);
    public final int CIVILIZATION_MANA_GENERATED_PER_MAGIC_TOWER = 10;

    // TECHNOLOGY COST

    public final int UPGRADE_BASE_DEFENSE_TECHNOLOGY_IRON_COST = 2000;
    public final int UPGRADE_BASE_ATTACK_TECHNOLOGY_IRON_COST = 2000;
    public final int UPGRADE_PLUS_DEFENSE_TECHNOLOGY_IRON_COST = 60;
    public final int UPGRADE_PLUS_ATTACK_TECHNOLOGY_IRON_COST = 60;

    public final int UPGRADE_BASE_DEFENSE_TECHNOLOGY_WOOD_COST = 0;
    public final int UPGRADE_BASE_ATTACK_TECHNOLOGY_WOOD_COST = 0;
    public final int UPGRADE_PLUS_DEFENSE_TECHNOLOGY_WOOD_COST = 0;
    public final int UPGRADE_PLUS_ATTACK_TECHNOLOGY_WOOD_COST = 0;

    // COST ATTACK UNITS

    public final int FOOD_COST_SWORDSMAN = 8000;
    public final int FOOD_COST_SPEARMAN = 5000;
    public final int FOOD_COST_CROSSBOW = 0;
    public final int FOOD_COST_CANNON = 0;
    public final int WOOD_COST_SWORDSMAN = 3000;
    public final int WOOD_COST_SPEARMAN = 6500;
    public final int WOOD_COST_CROSSBOW = 45000;
    public final int WOOD_COST_CANNON = 30000;
    public final int IRON_COST_SWORDSMAN = 50;
    public final int IRON_COST_SPEARMAN = 50;
    public final int IRON_COST_CROSSBOW = 7000;
    public final int IRON_COST_CANNON = 15000;
    public final int MANA_COST_SWORDSMAN = 0;
    public final int MANA_COST_SPEARMAN = 0;
    public final int MANA_COST_CROSSBOW = 0;
    public final int MANA_COST_CANNON = 0;

    // COST DEFENSES ARROWTOWER, CATAPULT, ROCKETLAUNCHERTOWER

    public final int IRON_COST_ARROWTOWER = 0;
    public final int IRON_COST_CATAPULT = 500;
    public final int IRON_COST_ROCKETLAUNCHERTOWER = 5000;
    public final int WOOD_COST_ARROWTOWER = 2000;
    public final int WOOD_COST_CATAPULT = 4000;
    public final int WOOD_COST_ROCKETLAUNCHERTOWER = 50000;
    public final int FOOD_COST_ARROWTOWER = 0;
```

```

public final int FOOD_COST_CATAPULT = 0;
public final int FOOD_COST_ROCKETLAUNCHERTOWER = 0;
public final int MANA_COST_ARROWTOWER = 0;
public final int MANA_COST_CATAPULT = 0;
public final int MANA_COST_ROCKETLAUNCHERTOWER = 0;

//Cost Especial units

public final int FOOD_COST_MAGICIAN = 12000;
public final int FOOD_COST_PRIEST = 15000;
public final int WOOD_COST_MAGICIAN = 2000;
public final int WOOD_COST_PRIEST = 0;
public final int IRON_COST_MAGICIAN = 500;
public final int IRON_COST_PRIEST = 0;
public final int MANA_COST_MAGICIAN = 5000;
public final int MANA_COST_PRIEST = 15000;

// array units costs SWORDSMAN, SPEARMAN, CROSSBOW, CANNON, ARROWTOWER, CATAPULT, ROCKETLAUNCHERTOWER

public final int WOOD_COST_UNITS =
{WOOD_COST_SWORDSMAN, WOOD_COST_SPEARMAN, WOOD_COST_CROSSBOW, WOOD_COST_CANNON, WOOD_COST_ARROWTOWER, WOOD_COST_CATAPULT, WOOD_COST_
_ROCKETLAUNCHERTOWER, WOOD_COST_MAGICIAN, WOOD_COST_PRIEST};
public final int IRON_COST_UNITS =
{IRON_COST_SWORDSMAN, IRON_COST_SPEARMAN, IRON_COST_CROSSBOW, IRON_COST_CANNON, IRON_COST_ARROWTOWER, IRON_COST_CATAPULT, IRON_COST_
_ROCKETLAUNCHERTOWER, IRON_COST_MAGICIAN, IRON_COST_PRIEST};
public final int FOOD_COST_UNITS =
{FOOD_COST_SWORDSMAN, FOOD_COST_SPEARMAN, FOOD_COST_CROSSBOW, FOOD_COST_CANNON, FOOD_COST_ARROWTOWER, FOOD_COST_CATAPULT, FOOD_COST_
_ROCKETLAUNCHERTOWER, FOOD_COST_MAGICIAN, FOOD_COST_PRIEST};

//Cost Buildings

public final int FOOD_COST_FARM = 5000;
public final int WOOD_COST_FARM = 10000;
public final int IRON_COST_FARM = 12000;

public final int FOOD_COST_CARPENTRY = 5000;
public final int WOOD_COST_CARPENTRY = 10000;
public final int IRON_COST_CARPENTRY = 12000;

public final int FOOD_COST_SMITHY = 5000;
public final int WOOD_COST_SMITHY = 10000;
public final int IRON_COST_SMITHY = 12000;

public final int FOOD_COST_CHURCH = 5000;
public final int WOOD_COST_CHURCH = 10000;
public final int IRON_COST_CHURCH = 12000;

public final int FOOD_COST_MAGICTOWER = 5000;
public final int WOOD_COST_MAGICTOWER = 10000;
public final int IRON_COST_MAGICTOWER = 12000;

// BASE DAMAGE ATTACK UNITS
public final int BASE_DAMAGE_SWORDSMAN = 80;
public final int BASE_DAMAGE_SPEARMAN = 150;
public final int BASE_DAMAGE_CROSSBOW = 1000;
public final int BASE_DAMAGE_CANNON = 700;

// BASE DAMAGE DEFENSES

public final int BASE_DAMAGE_ARROWTOWER = 80;
public final int BASE_DAMAGE_CATAPULT = 250;
public final int BASE_DAMAGE_ROCKETLAUNCHERTOWER = 2000;

public final int BASE_DAMAGE_MAGICIAN = 3000;

// ARMOR ATTACK UNITS
public final int ARMOR_SWORDSMAN = 400;
public final int ARMOR_SPEARMAN = 1000;
public final int ARMOR_CROSSBOW = 6000;
public final int ARMOR_CANNON = 8000;

// ARMOR DEFENSES

```

```

public final int ARMOR_ARROWTOWER = 200;
public final int ARMOR_CATAPULT = 1200;
public final int ARMOR_ROCKETLAUNCHERTOWER = 7000;

//Attack Units armor increase percentage per tech level

public final int PLUS_ARMOR_SWORDSMAN_BY TECHNOLOGY = 5;
public final int PLUS_ARMOR_SPEARMAN_BY TECHNOLOGY = 5;
public final int PLUS_ARMOR_CROSSBOW_BY TECHNOLOGY = 5;
public final int PLUS_ARMOR_CANNON_BY TECHNOLOGY = 5;

// defense armor increase percentage per tech level

public final int PLUS_ARMOR_ARROWTOWER_BY TECHNOLOGY = 5;
public final int PLUS_ARMOR_CATAPULT_BY TECHNOLOGY = 5;
public final int PLUS_ARMOR_ROCKETLAUNCHERTOWER_BY TECHNOLOGY = 5;

// attack units power increase percentage per tech level

public final int PLUS_ATTACK_SWORDSMAN_BY TECHNOLOGY = 5;
public final int PLUS_ATTACK_SPEARMAN_BY TECHNOLOGY = 5;
public final int PLUS_ATTACK_CROSSBOW_BY TECHNOLOGY = 5;
public final int PLUS_ATTACK_CANNON_BY TECHNOLOGY = 5;

// Defense attack power increase percentage per tech level

public final int PLUS_ATTACK_ARROWTOWER_BY TECHNOLOGY = 5;
public final int PLUS_ATTACK_CATAPULT_BY TECHNOLOGY = 5;
public final int PLUS_ATTACK_ROCKETLAUNCHERTOWER_BY TECHNOLOGY = 5;

public final int PLUS_ATTACK_MAGICIAN_BY TECHNOLOGY = 6;

public final int PLUS_ARMOR_UNIT_PER_EXPERIENCE_POINT = 4;
public final int PLUS_ATTACK_UNIT_PER_EXPERIENCE_POINT = 4;

//Units plus armor/attack increase percentage when sanctified
public final int PLUS_ARMOR_UNIT_SANCTIFIED = 7;
public final int PLUS_ATTACK_UNIT_SANCTIFIED = 7;

// Chance of resurrection by magician
public final int CHANCE_MAGICIAN_RESSURECT = 2;

// fleet probability of generating waste

public final int CHANCE_GENERATING_WASTE_SWORDSMAN = 55;
public final int CHANCE_GENERATING_WASTE_SPEARMAN = 65;
public final int CHANCE_GENERATING_WASTE_CROSSBOW = 80;
public final int CHANCE_GENERATING_WASTE_CANNON = 90;

// Defense probability of generating waste ARROWTOWER, CATAPULT, ROCKETLAUNCHERTOWER

public final int CHANCE_GENERATING_WASTE_ARROWTOWER = 55;
public final int CHANCE_GENERATING_WASTE_CATAPULT = 65;
public final int CHANCE_GENERATING_WASTE_ROCKETLAUNCHERTOWER = 75;

// especial Units
public final int CHANCE_GENERATING_WASTE_PRIEST = 0;
public final int CHANCE_GENERATING_WASTE_MAGICIAN = 0;

// AttackUnit chance to attack again

public final int CHANCE_ATTACK AGAIN SWORDSMAN = 3;
public final int CHANCE_ATTACK AGAIN SPEARMAN = 7;
public final int CHANCE_ATTACK AGAIN CROSSBOW = 45;
public final int CHANCE_ATTACK AGAIN CANNON = 70;

```

```
//Defense chance to attack again

public final int CHANCE_ATTACK AGAIN ARROWTOWER = 5;
public final int CHANCE_ATTACK AGAIN CATAPULT = 12;
public final int CHANCE_ATTACK AGAIN ROCKETLAUNCHERTOWER = 30;

public final int CHANCE_ATTACK AGAIN MAGICIAN = 75;
public final int CHANCE_ATTACK AGAIN PRIEST = 0;

// CHANCE ATTACK EVERY UNIT

// SWORDSMAN, SPEARMAN, CROSSBOW, CANNON, ARROWTOWER, CATAPULT, ROCKETLAUNCHERTOWER, MAGICIAN, PRIEST
public final int CHANCE_ATTACK CIVILIZATION UNITS = {4,9,13,37,4,9,14,10,0};

// SWORDSMAN, SPEARMAN, CROSSBOW, CANNON
public final int CHANCE_ATTACK ENEMY UNITS = {10,20,30,40};

// percentage of waste that will be generated with respect to the cost of the units
public final int PERCENTATGE WASTE = 70;

}
```

Observamos que todo son constantes que caracterizan la dificultad del juego.

# Clase Battle

La clase Battle nos servirá para gestionar el desarrollo de las batallas. Ésta será posiblemente la clase más compleja de implementar del proyecto.

Los variables obligatorias con las que tendremos que trabajar:

- **ArrayList<MilitaryUnit> civilizationArmy** → para almacenar nuestro ejército
- **ArrayList<MilitaryUnit> enemyArmy** → para almacenar el ejército enemigo.
- **ArrayList armies** → que es un array de ArrayList de dos filas y nueve columnas, donde almacenaremos nuestro ejército en la primera fila, y el ejército enemigo en la segunda fila;
- **String battleDevelopment** → Donde guardamos todo el desarrollo de la batalla paso a paso
- **int initialCostFleet** → coste de comida, madera y hierro de los ejercitos iniciales  

$$\text{initialCostFleet} = [[\text{comida}][\text{madera}][\text{hierro}], [\text{comida}][\text{madera}][\text{hierro}]]$$
, donde initialCostFleet[0] costes unidades de nuestra civilización , initialCostFleet[1] costes unidades enemigas. Lo necesitamos para saber las pérdidas en materiales de cada ejército.
- **int initialNumberUnitsCivilization, initialNumberUnitsEnemy** → La batalla se acabará cuando uno de los dos ejércitos se quede con el 20% o menos de sus unidades iniciales, por tanto es necesario saber la cantidad de unidades iniciales de cada ejército.
- **int wasteWoodIron** → residuos generados en la batalla [madera, hierro].
- **int enemyDrops, int civilizationDrops**, necesarios para generar reporte de batalla, y para calcular las pérdidas materiales de cada ejército.
- **int resourcesLooses** → array de dos filas y cuatro columnas, resourcesLooses[0] = {pérdidas comida civilización, pérdidas madera civilización, pérdidas hierro civilización, pérdidas hierro civilización + 5\* pérdidas madera civilización + 10\*pérdidas comida civilización}, resourcesLooses[1] lo mismo pero para el ejercito enemigo.  
 Multiplicamos por 5 las pérdidas de madera y por 10 las pérdidas de comida, debido a que queremos cuantificar con un solo valor las pérdidas. Los recursos más valiosos son el hierro, luego la madera ( 1 de hierro = 5 de madera) y luego la comida ( 1 de hierro = 10 de comida)  
 Para decidir el ganador, será que que tenga el numero menor en la tercera columna. ResourcesLooses[0][3] y ResourcesLooses[1][3], que representan las pérdidas ponderadas.
- **int initialArmies** → Array de dos filas y 9 columnas. Servirá para cuantificar cada tipo de unidad de los ejercitos iniciales.

InitialArmies[0] serán las unidades iniciales de nuestra civilización y  
 InitialArmies[1] las unidades iniciales enemigas.  
 InitialArmies[0][0] swordsman de nuestra civilización antes de iniciar batalla.  
 InitialArmies[0][1] spearman de nuestra civilización antes de iniciar batalla..

Este array nos ayudará a calcular los costes de las flotas iniciales y por tanto, las pérdidas.

- **int actualNumberUnitsCivilization, int actualNumberUnitsEnemy** --> arrays que cuantifican las unidades actuales de cada grupo, tanto para nuestra civilización, como para el enemigo. El orden seria:  
 actualNumberUnitsPlanet[0] --> swordsman  
 actualNumberUnitsPlanet[1] --> spearman  
 actualNumberUnitsPlanet[2] --> crossbow  
 actualNumberUnitsPlanet[3] --> cannon  
 actualNumberUnitsPlanet[4] --> arrow tower  
 actualNumberUnitsPlanet[5] --> catapult  
 actualNumberUnitsPlanet[6] --> rocket launcher tower  
 actualNumberUnitsPlanet[7] --> magician  
 actualNumberUnitsPlanet[8] --> priest  
 Es necesario tener contabilizadas las unidades actuales de cada clase, debido a que en la mecánica de batalla, se escoge un defensor en función de la cantidad de unidades de cada clase.

## Mecánica de la batalla:

Partimos de dos ejércitos con diferentes unidades, nuestra civilización tiene defensas y el ejercito que nos ataca no tiene defensas.

A cada tipo de unidad diferente le llamamos grupo, para abbreviar.

Empieza a atacar un ejercito aleatoriamente.

De ese ejército, cada grupo tiene una probabilidad de atacar, definido en la interfaz Variables.

```
// SWORDSMAN, SPEARMAN, CROSSBOW, CANNON, ARROWTOWER, CATAPULT, ROCKETLAUNCHERTOWER, MAGICIAN, PRIEST
public final int CHANCE_ATTACK_CIVILIZATION_UNITS = {4,9,13,37,4,9,14,10,0};

// SWORDSMAN, SPEARMAN, CROSSBOW, CANNON
public final int CHANCE_ATTACK_ENEMY_UNITS = {10,20,30,40};
```

Es decir, que la probabilidad de ser los cañones los que ataquen es un 37% en nuestro caso y un 40% en el caso enemigo, la de un swordsman es un 4% en nuestro caso, y un 10% en el caso enemigo.

Más adelante explicaremos un algoritmo sencillo para hacer uso de los arrays de probabilidades.

Una vez se ha escogido el grupo atacante, se selecciona una unidad al azar del grupo que será la unidad atacante.

Seguidamente escogeremos un grupo defensor, y para ello, las probabilidades de escoger cada uno de los grupos, estará basada en la cantidad de unidades del grupo.

Si por ejemplo el ejército defensor está compuesta por:

90 swordsman  
 60 spearman  
 30 crossbow  
 20 cannon

La probabilidad de escoger un defensor del primer grupo ( swordsman ) será de un 45%, es decir:

$$100 * (\text{Cantidad de swordsman}) / (\text{total de unidades}) = 9000/200 = 45$$

La probabilidad de escoger un defensor del segundo grupo será del 30%.

$$100 * (\text{Cantidad spearman}) / (\text{total de unidades}) = 6000/200 = 30.$$

También explicaremos un algoritmo sencillo para escoger un grupo en función de las probabilidades definidas arriba.

Una vez escogido el grupo defensor, escogeremos uno de los defensores dentro del grupo de forma aleatoria.

Una vez escogidas las unidades atacante y defensora, la atacante infligirá el daño igual a su poder de ataque y reducirá la armadura de la defensora en una cantidad igual al poder de ataque de la unidad atacante.

Si la armadura de la unidad defensora se queda en cero o un número negativo, se eliminará. Antes de eliminarla, comprobaremos si genera residuos, la probabilidad de generar residuos está definida en la interfaz Variables, por ejemplo, int CHANCE\_GENERATNG\_WASTE\_SWORDSMAN = 55

Si se generan residuos, se generará un porcentaje de los recursos de madera y hierro que costó construir dicha unidad, este porcentaje estará definido en la interfaz Variables, final int PERCENTATGE\_WASTE = 70. Igual para todas las unidades.

Sólo se generarán residuos de madera y hierro.

Una vez realizada las acciones antes mencionadas, comprobaremos si la unidad atacante tiene opción de atacar de nuevo, la probabilidad de atacar de nuevo de cada unidad, está descrita en la interfaz Variables, por ejemplo, int CHANCE\_ATTACK AGAIN\_SWORDSMAN = 3, indica que los awordsman tienen un 3% de probabilidades de repetir ataque.

En caso de repetir ataque, se vuelve a escoger un grupo aleatorio defensor, y después un defensor del grupo escogido y volvemos a realizar los mismos pasos.

En caso de no repetir ataque, se cambia el turno, el ejercito atacante pasa a ser ejército defensor, el defensor pasa a ser atacante y aplicamos la misma mecánica descrita.

Este proceso se repetirá mientras uno de los dos ejércitos mantenga al menos el 20% de las unidades totales del que estaba compuesto antes de empezar el ataque.

Esta probabilidad del 20 por ciento, también podría definirse en la interfaz Variables, a modo de calibrar el juego.

Al final de la batalla, si ganamos la batalla, podremos quedarnos con los residuos generados.

Para calcular quien gana la batalla, calcularemos el siguiente número para cada flota:

(total de pérdidas de hierro) + (total de pérdidas de madera)/5 + (total de pérdidas de comida)/10

Que serán las perdidas ponderadas de forma que 5 unidades de madera tengan el mismo valor que 1 unidad de hierro.

El ejército que tenga menos pérdidas, será el ejército ganador.

Todos los eventos que ocurren durante la batalla, deberán ser guardados en algún String de forma que obtengamos el siguiente formato cuando queramos ver el **reporte paso a paso** de la batalla:

```
*****CHANGE ATTACKER*****
Attacks army enemy: Spearman attacks Arrow Tower
Spearman generates the damage = 150
Arrow Tower stays with armor = 80

*****CHANGE ATTACKER*****
Attacks Civilization: Crosswob attacks Swordsman
Crosswob generates the damage = 700
Swordsman stays with armor = -300
we eliminate Swordsman
Attacks Civilization: Crosswob attacks Cannon
Crosswob generates the damage = 700
Cannon stays with armor = 5300

*****CHANGE ATTACKER*****
Attacks army enemy: Crosswob attacks Rocket Launcher Tower
Crosswob generates the damage = 700
Rocket Launcher Tower stays with armor = -470
we eliminate Rocket Launcher Tower
```

También tendremos que devolver un reporte resumen de unidades y recursos perdidos por ejército:

```
4)View Battle Reports
Option >
4
Battle Reports
Select Report Read (0 go back): (1)
Option >
1
BATTLE NUMBER: 1
BATTLE STATISTICS

Army planet      Units    Drops   Initial Army Enemy      Units    Drops
Sordsman          11       8       Sordsman                  19       17
Spearman           3        1       Spearman                 7        5
Crossbow            1        0       Crossbow                  1        1
Cannon              1        0       Cannon                   1        0
Arrow Tower         11       9
Catapult             1        1
Rocket Launcher Tower  1        0
Magician             1        0
Priest               1        1

*****
Cost Army Civilization          Cost Army Enemy
Food:      203500          Food:      177500
Wood:      28200           Wood:      23300
Iron:      35000            Iron:      32000

*****
Losses Army Civilization          Losses Army Enemy
Food:      52000           Food:      128500
Wood:      950             Wood:      8100
Iron:      23000            Iron:      29000
```

```

Waste Generated:
Wood           22150
Iron            12000

Battle Wonned by Civilization, We Collect Rubble

#####
View Battle development?(S\n)

```

En caso de seleccionar ver desarrollo de la batalla, mostraremos el **reporte paso a paso**.

Estos dos últimos reportes serán devueltos cuando llamemos a los métodos:

**String getBattleReport(int battles)** → resumen, battles será el número de batallas que hayamos acumulado

**String getBattleDevelopment()** → paso a paso.

Los siguientes métodos pueden ser útiles para el desarrollo de la batalla:

**void initInitialArmies()** → Para inicializar el array initialArmies y poder calcular los reportes.

**void updateResourcesLooses()** - → Para generar el array de pérdidas.

**fleetResourceCost(ArrayList<MilitaryUnit> army)** → Para calcular costes de los ejércitos.

**initialFleetNumber(ArrayList<MilitaryUnit> army)** → Para calcular el número de unidades iniciales de cada ejército

**int remainderPercentageFleet(ArrayList<MilitaryUnit> army)** → Para calcular los porcentajes de unidades que quedan respecto los ejércitos iniciales.

**int getGroupDefender(ArrayList<MilitaryUnit> army)** → para que dado un ejército, nos devuelva el grupo defensor, 0-3 en el caso de la flota enemiga, 0-8 en el caso del ejército de nuestra civilización.

**int getCivilizationGroupAttacker(), int getEnemyGroupAttacker()** → Que nos servirán para escoger el grupo atacante tanto de nuestra civilización como del ejército enemigo.

**void resetArmyArmor()** → que restablecerá las armaduras de nuestro ejército.

# Clase Main

Desde la clase Main podremos acceder a diferentes opciones como crear unidades en nuestra civilización, ya sean defensivas como ofensivas, crear edificios ( Granja, Carpintería, Herrería, Torre Mágica, Iglesia) mejorar nuestras tecnologías, ver los reportes de batalla de al menos las últimas 5 batallas, o en caso de haber un ejército enemigo a punto de atacarnos, ver el ejército enemigo.

Tendremos la opción de ver nuestros stats: ejército, recursos, edificios, tecnologías ...

Se controlará desde esta clase el aumento de recursos de nuestra civilización. Esto se podrá realizar mediante una tarea “TimerTask” que se explicará más adelante.

Se creará un ejército enemigo cada 3 minutos y una batalla. Estas dos tareas también se podrán implementar mediante la clase TimerTask

Se dejará a disposición de los alumnos una versión por consola del funcionamiento del juego.

## **createEnemyArmy()**

Para crear el ejército enemigo, dispondremos de unos recursos iniciales, que conforme vayan sucediendo batallas, serán mayores .

Iremos creando unidades enemigas aleatoriamente pero con las siguientes probabilidades: Swordsman 35%, Spearman 25%, Crossbow 20%, Cannon 20%.

Mientras tengamos suficientes recursos para crear la unidad con menor coste, es decir, Swordsman iremos creando unidades aleatoriamente según las probabilidades anteriores.

## **Viewthreat()**

Este método nos mostrará que tipo de ejército nos viene a atacar:

NEW threat COMMING

Swordsman	16
Spearman	12
Crossbow	1
Cannon	1

La aplicación aquí descrita está basada toda en modo consola. Se deberá implementar una aplicación gráfica desde la cual podamos controlar lo mismo que desde la consola. El diseño de la aplicación gráfica será libre.

## Ayudas

### **Elección grupo defensor en función de las unidades del grupo y elección del atacante en función de unas probabilidades de atacar predefinidas.:**

Supongamos que tenemos un array de n posiciones, y en cada posición del array tenemos un número entero que representa la cantidad de unidades que tenemos de una categoría. Representando cada posición del array una categoria diferente a las demás. Por ejemplo:

Array = {10,35,27,70}

Y queremos escoger una categoría, pero con una probabilidad proporcional a número de unidades de dicha categoría. En nuestro ejemplo, como en la posición 3 tenemos 70 unidades y en la posición 1 tenemos la mitad, queremos que la categoria 3 tenga el doble de probabilidad de salir escogida respecto de la categoria 2.

Una posible forma de resolver este problema sería:

Calculamos la suma total de todas las categorías. SumaTotal =  $10 + 35 + 27 + 70 = 152$ . Escogemos un número aleatorio numAleatorio entre 1 y la suma total, 152 en nuestro caso. Y vamos recorriendo el array sumando las cantidades de cada posición, hasta que dicha suma sea mayor que el número aleatorio.

En nuestro caso, si numAleatorio = 70.

array[0] = 10 < numAleatorio. No escogemos la categoria 0.  
array[0] + array[1] = 45 < numAleatorio. No escogemos la categoria 1.  
array[0] + array[1] + array[2] = 72 > numAleatorio. Escogemos la categoria 2.

### **Repetición de un evento cada X milisegundos:**

Para ello crearemos un objeto del tipo TimerTask:

```
TimerTask task = new TimerTask() {
    public void run()
    {
    }
};
```

Dentro del método run implementaremos la tarea que querremos ejecutar cada cierto tiempo, en nuestro caso imprimir “Hola Mundo”

```
TimerTask task = new TimerTask() {
    public void run()
    {
        System.out.println("Hola mundo");
    }
};
```

Ahora sólo necesitamos programar dicha tarea cada 5 segundos, para ello crearemos un objeto de la clase Timer.

```
Timer timer = new Timer();
```

Este objeto dispone del método schedule

```
public void schedule(TimerTask task, long delay, long period)
```

- task es la tarea que queremos ejecutar.
- delay el tiempo que queremos que pase antes de que se ejecute la primera vez en milisegundos
- period es el tiempo que queremos que pase entre cada ejecución de la tarea, en milisegundos también.

Si queremos imprimir el mensaje “Hola Mundo” cada 5 segundos ( 5000 milisegundos) , pero que el primero aparezca a los 10 segundos (10000 milisegundos), podremos ejecutar.

```
timer.schedule(task, 10000, 5000);
```

Comprueba el resultado.

Luego crea una segunda tarea, por ejemplo:

```
TimerTask task2 = new TimerTask() {
```

```
    public void run()
    {
        System.out.println("Bienvenido");
    }
};
```

y ejecuta:

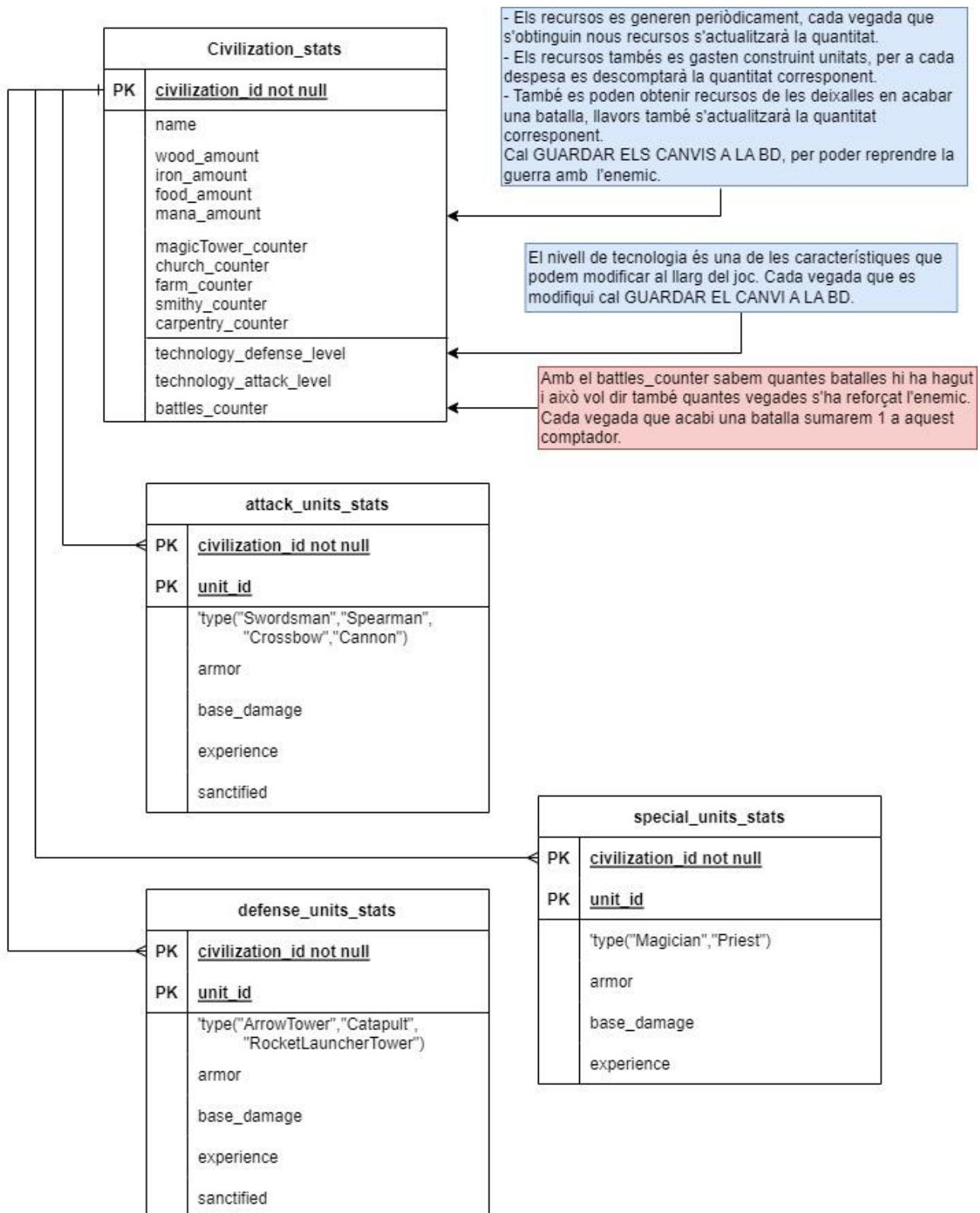
```
timer.schedule(task1, 10000, 5000);
timer.schedule(task2, 8000, 3000);
```

Es decir, la primera tarea cada 10 segundos con un delay de 5, y la segunda cada 2 segundos con un delay de 8 segundos.

De esta manera ya podemos planificar diversas tareas con tiempos diferentes con el mismo objeto de la clase Timer.

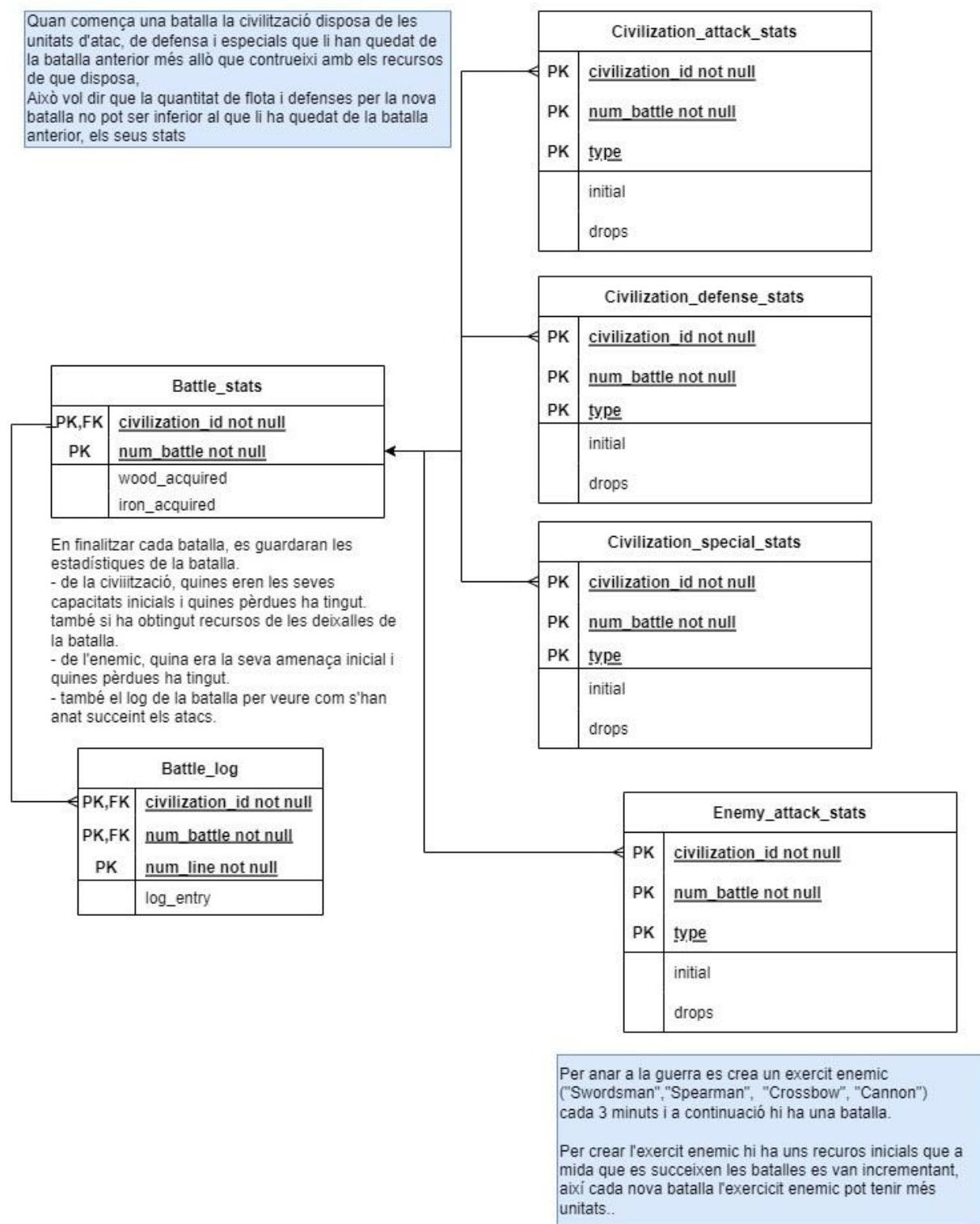
# M02-Base de datos

La aplicación tendrá que guardar y recuperar los datos en una base de datos Oracle. Un ejemplo de las tablas necesarias para guardar los datos del juego es el siguiente;



En el diagrama anterior se muestran los datos a guardar de una civilización, una vez acabado o pausado el juego.

En esta diagrama se muestran los datos a guardar de las batallas con el enemigo, para poder mostrar en cualquier momento el resultado de los enfrentamientos que nuestra civilización ha tenido con el ejército enemigo.



Os pasaremos un ejemplo de conexión a una base de datos.

## M04 - Lenguaje de marcas

Para la asignatura de M04, trabajaremos a través de GitHub en su carpeta particular, en la que tendrá que haber los siguientes puntos para cada clase padre del proyecto (recursos, edificios, tecnologías, unidades):

- El código Python/Java para obtener los datos de la BBDD en un archivo XML (este puede ser común para todos).
- El archivo XML resultante de ejecutar el código anterior, conteniendo los objetos base de cada clase padre.
- El archivo XSL para transformar el archivo anterior en un archivo HTML.
- El código python para transformar el archivo XML en HTML a través de XSL (este puede ser común para todos).
- El archivo HTML resultando de ejecutar el código anterior.

Los archivos Python se tienen que poder ejecutar en terminal.

La estructura del proyecto es:

- dataToXml.py
- xmlToHtml.py
- Carpeta “html” con los ficheros HTML de cada clase (y los archivos CSS convenientes).
- Carpeta “xml” con los archivos XML y XSL de cada clase.

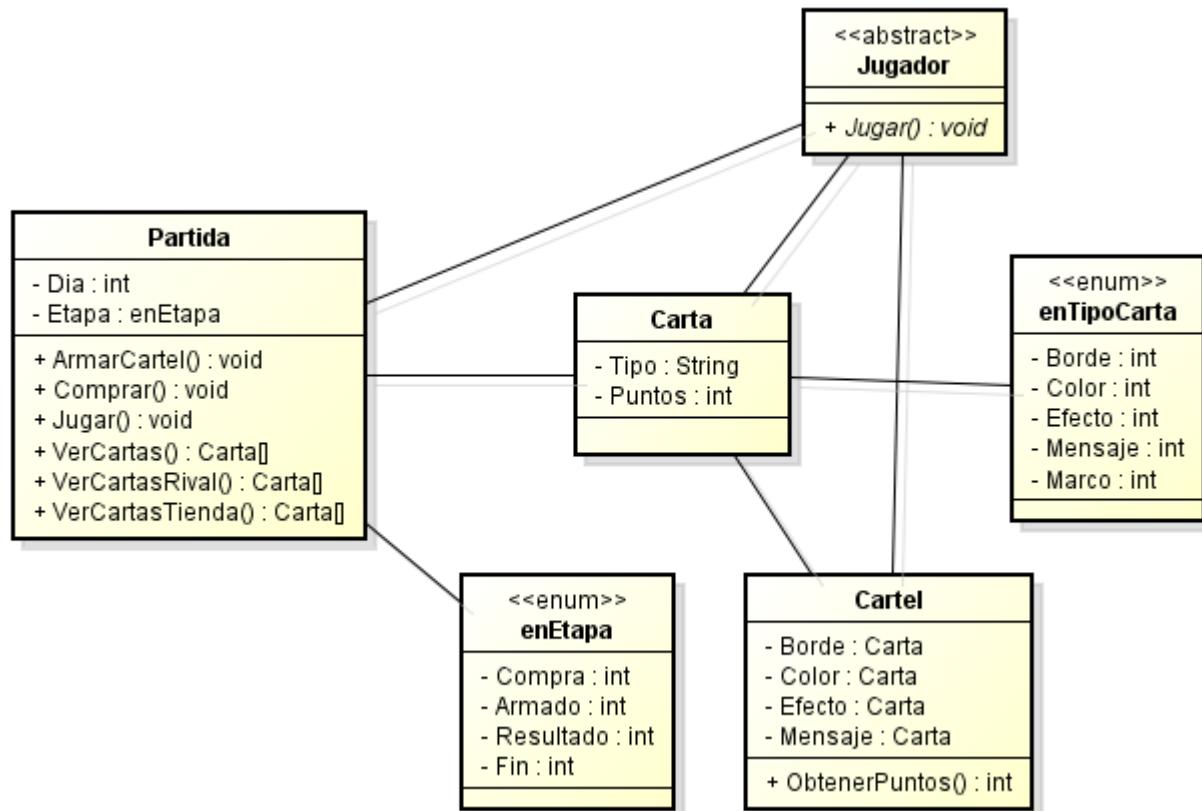
# M05 - Entornos de desarrollo

## Diagrama de clases

*Hay que realizar un diagrama de clases de las clases e interfaces de la aplicación, incluyendo atributos y métodos.*

*No hay que incluir todas las diferentes clases, las suficientes para comprender la estructura de objetos de la aplicación y sus relaciones.*

*Ejemplo:*



*Este ejemplo es sólo para ilustrar el tipo de representación que se pide, no se debe emplear como referencia exacta del que se debe realizar para vuestra aplicación.*

## Control de versiones

- *Hay que utilizar GitHub, cada commit debe ir acompañado del mensaje referente al cambio realizado.*
- *En el proyecto debe haber una carpeta por cada módulo M1, M2, M3, M4 i M5*
- *El proyecto debe tener un “README.md” con una breve explicación del proyecto e instrucciones para instalarlo y ejecutarlo.*
- *Cada alumno debe tener una “branch” de trabajo que debe consolidarse a la de preproducción.*
- *La “branch” principal se llamará “main”.*

### Especificaciones no funcionales:

- *En la rama de "reproducción" es donde se hace el 'merge' de los cambios que ha realizado cada miembro del equipo.*
- *La entrega final se hará en formato de 'release' a GitHub.*
- *La URL de la "release" se subirà a a tarea de Moodle de M5.*

# Planificación

*Una relación de las tareas que se deben llevar a cabo, es recomendable realizar una estimación del tiempo que se prevea para cada una de ellas y revisar posteriormente si se ha cumplido la previsión.*

Mòdul	Tasca	T. estimat	T. real
M05	<i>Crear el repositori i les branques a Github</i>		
M05	<i>Crear el diagrama de clases</i>		
M01	<b>Fer la màquina virtual i configurar-la</b>		
M01	<b>Instal·lar i configurar la DDBB Oracle</b>		
M02	<i>Crear los Script de la DDBB y ejecutarlos</i>		
M03	<i>Programar classe main</i>		
M03	<i>Programar classe battle</i>		
M03	<i>Programar classes interfície variables</i>		
M03	<i>Programar interficies</i>		
M03	<i>Programar classes "unit"</i>		
M03	<i>Programar elementos gráficos</i>		
M04	<i>Fer la base de la pàgina web (menú, peu, CSS)</i>		
M04	<i>Fer la pàgina de portada</i>		
M04	<i>Fer la pàgina de tutorial</i>		
M04	<i>Fer la pàgina 'About Us'</i>		
M05	<i>Crear el README.md a Github</i>		
M05	<i>Crear la release a Github</i>		
	<i>Fer i entrenar la presentació</i>		

Posibles Mejoras a Modo de Ejemplo:

Que resetear las armaduras de nuestro ejército tenga un coste.

Que los ataques vengan de otra civilización, que produce recursos en grandes cantidades y por tanto tiene opción de crear un ejército para enviárnoslo a atacar.

Que los ataques sean cada cierto tiempo aleatorio.

Limitar el nivel máximo de las tecnologías.

Que el mantenimiento de las unidades Swordsman y Spearman suponga cierto gasto de comida, y en caso de no haber suficiente, mueran las que no podamos alimentar.