

## Testing elaborat: Masmorra\_TQS

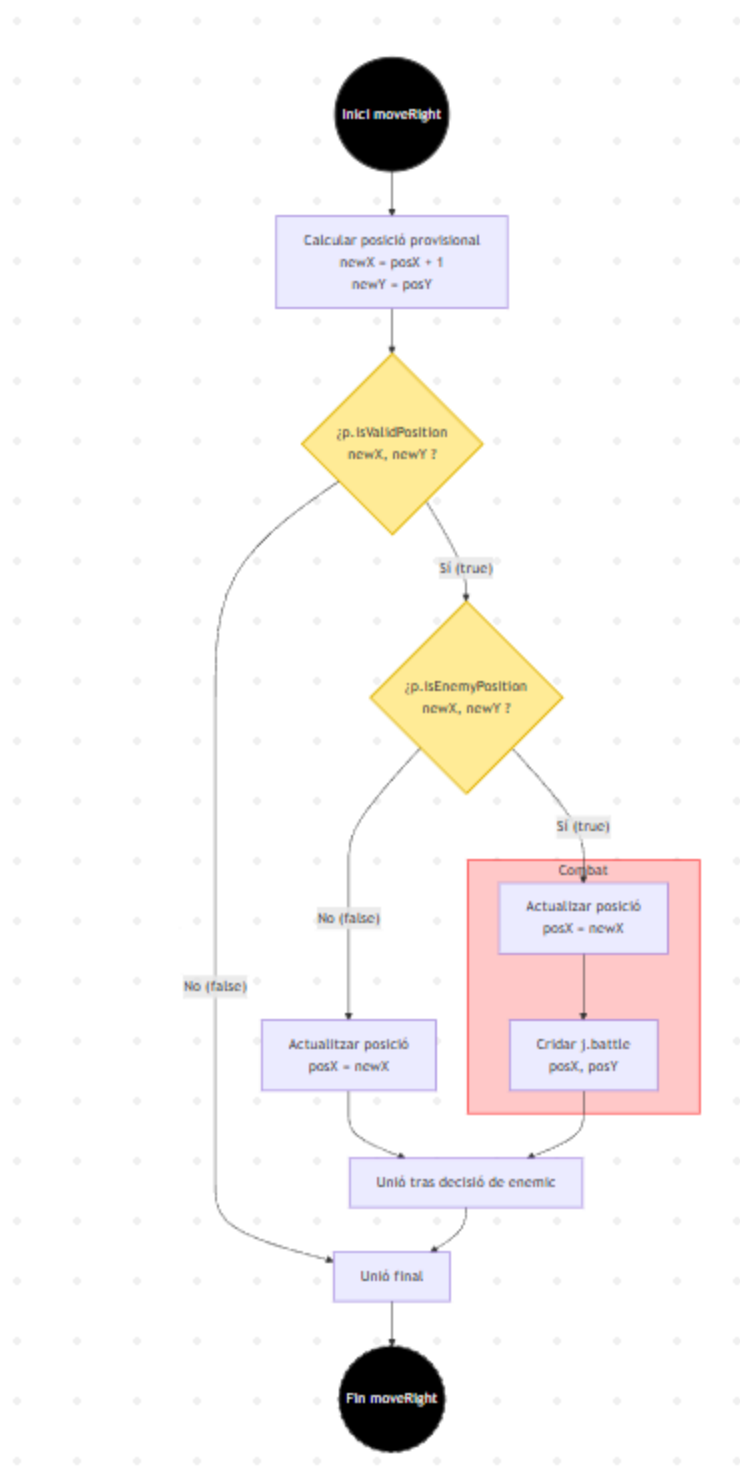
### Classe Jugador

Classe que controla tota la lògica relacionada al jugador; moviment, estadístiques posicions etc. A aquesta classe he testejat els mètodes moveRight, moveLeft, moveUp i moveDown, a més de setHp, setEXP, openDoor, isPlayerAtDoor i lvUp.

**moveRight, moveLeft, moveUp, moveDown:** fent ús de MockObjects per les classes planta i Joc he testejat els diferents camins possibles(decision coverage, path coverage, statement coverage) alhora de moure's en el joc.

```
public void moveRight(Planta p, Joc j) {
    int newX = posX + 1; // movimiento provisional
    int newY = posY;

    if (p.isValidPosition(newX, newY)) {
        if (p.isEnemyPosition(newX, newY)) {
            posX = newX;
            j.battle(posX, posY);
        } else {
            posX = newX;
        } // solo movemos si es válido
    }
}
```



**setHp, setEXP:** testeig de valors límits i de frontera per comprovar que no poden tomar valors negatius, o sobrepassar els valors màxims de salut i experiència.

**openDoor:** testeig simple de que es pot obrir la porta

**isPlayerAtDoor:** comprovació condition i decision coverage de que el jugador es troba a la mateixa casella de la porta.

```
public boolean isPlayerAtDoor(Planta p) {  
    if (p.getDoorposX() == this.posX && p.getDoorposY() ==  
this.posY) {  
        return true;  
    } else {  
        return false;  
    }  
}
```

**lvUp:** comprovació de que els valors incrementen correctament al pujar de nivell i proves de caixa blanca per al redondeig correcte.

## Classe Planta

**getEnemy, isEnemyPosition i isDoorPosition:** comprovació de casos límits i normals i condicions i decisions; quan l'enemic/porta es troba correctament i quan no.

**isValidPosition:** testeig de valors límit i fora de límit i de les diferents condicions.

```
public boolean isValidPosition(int x, int y) {  
    int limX = 79;  
    int limY = 19;  
    return x >= 0 && x <= limX  
        && y >= 0  
        && y <= limY  
        && this.floorLayout[y].charAt(x) != '#';  
}
```

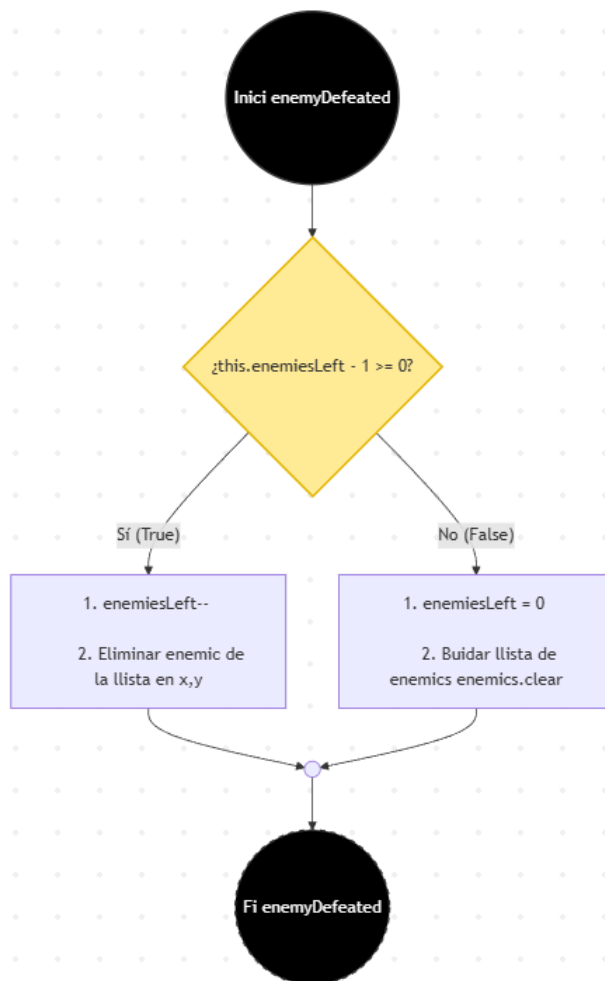
**enemyDefeated:** testeig de casos i path coverage.

```
public void enemyDefeated(int x, int y) {  
    if (this.enemiesLeft - 1 >= 0) {  
        this.enemiesLeft--;  
        for (int i = 0; i < this.enemics.size(); i++) {
```

```

    if (this.enemics.get(i).getPos_x() == x &&
this.enemics.get(i).getPos_y() == y) {
        this.enemics.remove(i);
    }
}
} else {
    this.enemiesLeft = 0;
    this.enemics.clear();
}
}
}

```



## Classe Poció

**usarItem:** testeig de casos habituals i límits. Comprovació d'augment de salut i de múltiples crides.

## Classe Joc

Classe encarregada de inicialitzar el joc i de gestionar tota la lògica entre diferents elements d'aquest.

**startGame, missatgeTemporal:** comprovació de valors d'inicialització adequats i de múltiples trucades.

**battle:** path coverage de les diferents situacions possibles al combatir, a més de testeig de diferents valors límits a les batalles i de l'obtenció d'items després d'aquestes.

```
public void battle(int x, int y) {
    Enemic e = this.plantaActual.getEnemy(x, y);

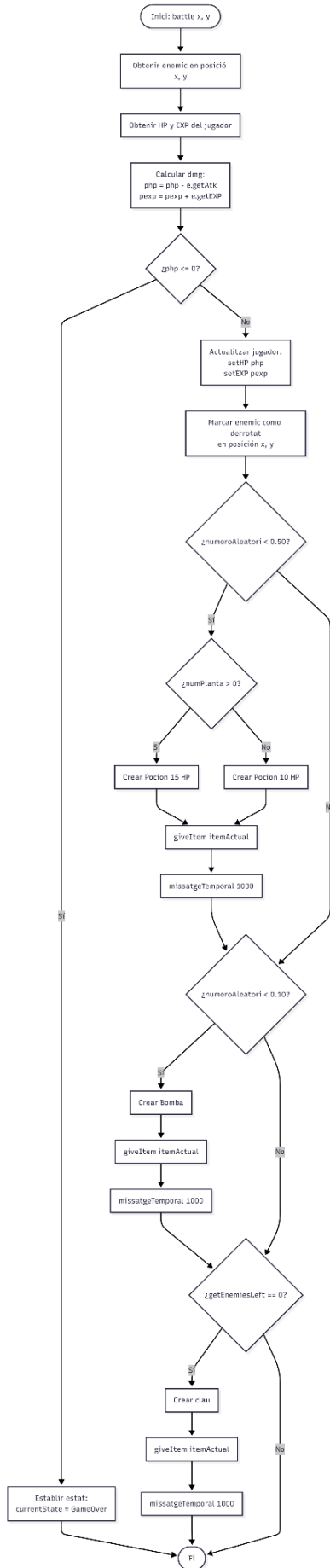
    int php = this.player.getHP();
    int pexp = this.player.getEXP();

    php = php - e.getAtk();
    pexp = pexp + e.getEXP();

    if (php <= 0) {
        currentState = GameState.gameOver;
    } else {
        this.player.setHP(php);
        this.player.setEXP(pexp);
        this.plantaActual.enemyDefeated(x, y);

        //Drop de pocio
        if (numeroAleatori() < 0.50) {
            if(this.numPlanta > 0){
                this.itemActual = new Poció(this.player,15);
            }else{
                this.itemActual = new Poció(this.player,10);
            }
        }
    }
}
```

```
    }  
    giveItem(itemActual);  
    missatgeTemporal(1000);  
}  
  
//Drop de bomba  
if(numeroAleatori() < 0.10 ){  
    this.itemActual = new Bomba(this.player);  
    giveItem(itemActual);  
    missatgeTemporal(1000);  
}  
  
if (this.plantaActual.getEnemiesLeft() == 0) {  
    this.itemActual = new Llave(this.player);  
    giveItem(this.itemActual);  
    missatgeTemporal(1000);  
}  
}  
}
```



**boom:** comprovacions de valors límit i estandars alhora d'eliminar múltiples enemics.

**passarDePlanta:** testeig de condicions i modificació correcta de variables.

## Classe LoopJoc

Classe que gestiona el bucle principal de joc i que fa de controlador entre la vista (Interface) i el model (Joc). Per dur a terme el testeig necessari del seu únic mètode *startGame* he hagut de crear les funcions *crearPulsacions* i *crearCarac* per simular les pulsacions del teclat

**startGame:** testeig de les diferents accions possibles depenent de l'estat de joc i de les diferents transicions.

