



# **Projet de Programmation**

## **Composant**

**M1 STL 2018-2019**

Hongxing WANG

Yueyi WANG

## Sommaire

Introduction .....	3
Manuel d'utilisation succinct de l'implémentation .....	3
Spécification formelle complète du projet .....	5
Les exemples d'exécutions du jeu .....	9
La description formelle de tests MBT effectué .....	12
Les difficultés du projet .....	15

## Introduction

Dans ce projet, on va créer un petit jeu Lode Runner, on va implémenter la spécification du jeu par des contrats selon le motif Decorator vu en cours. On va tester aussi le jeu en utilisant JUnit selon la méthode MBT.

## Manuel d'utilisation succinct de l'implémentation

Pour jouer ce jeu, il y a deux grandes parties à préparer.

Dans un premier temps, on doit créer un environnement du jeu en utilisant l'opération de la service EditableScreen. Dans notre projet, on dessine la cadre du jeu dans les fichiers `level.lv1` dans le package `levels`, le cadre est composé par des chiffres.

Les spécifications des chiffres sont suivantes :

- 0- EMP
- 1- MTL
- 2- PLT
- 3- LAD
- 4- HDR

Après on doit initialiser les positions du joueur, des gardes et des trésors, dans une classe qui s'appelle `GameFacadeCorrect`. Dans cette classe on implémente le jeu en utilisant des contacts selon le motif Decorator. Dans la fonction `buildLevel(int level)`, on peut déclarer tous les positions initiales des caractères et des trésor. Dans notre cas, on définit la largeur du jeu est 28 et la hauteur est 16. On a seulement un joueur, deux gardes, deux trésors, un item Super, et un item Arms. Si vous voulez ajouter les caractères ou trésors, vous pourriez implémenter dans le constructeur de `GameFacadeCorrect`.

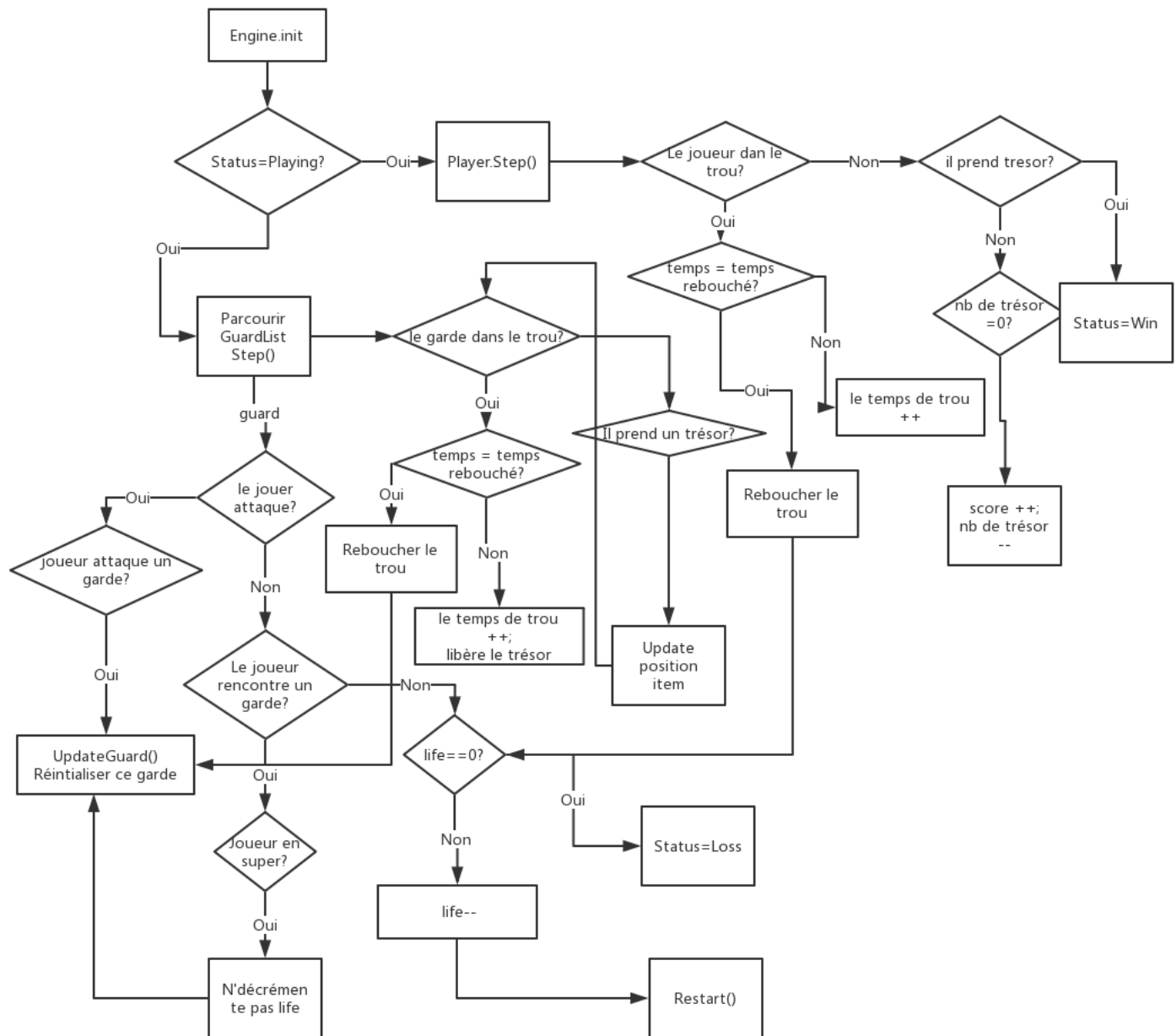
Par exemple, la cadre du jeu de niveau1 est la suivante :












## 1 Engine

On peut voir le diagramme moteur est le suivant :



## 1 Synchronisation l'engine avec tous les services

On crée des classes `requireService` comme on a vu en Tme4, chaque classe contient une fonction qui permet de créer les liaisons entre deux services et fournir les opérateurs de service.

- >  RequireEditableScreenService.java
- >  RequireEngineService.java
- >  RequireEnvironmentService.java
- >  RequireGuardService.java
- >  RequireHolesService.java
- >  RequireItemService.java
- >  RequirePlayerService.java

Par exemple, dans l'implémentation de la service Environnement . On a utilisé

```
public void bindEngineService(EngineImpl service)
```

Pour synchroniser toutes les positions de CellContent à chaque étape, la définition de la fonction `getCellContent` est suivante :

```
@Override
public HashSet<CellContent> getCellContent(int x, int y) {
    cellContentList.clear();
    for (Entry<CellContent, Pair<Integer, Integer>> entry
        : engine.getCellContent().entrySet()) {
        if(entry.getValue().getKey()==x && entry.getValue().getValue()==y) {
            cellContentList.add(entry.getKey());
        }
    }
    return cellContentList;
}
```

D'ici, on a parcouru tous le tableau Map qui stock tous les contenus de Cell, et leurs coordonnées correspondantes. Après on prend les CellConten à la position (x,y), et l'enregistre dans un HashSet<CellContent>

A chaque étape du joueur, si les positions des caractères ou items ont changé, le moteur engine met à jours le tableau CellMap.

## 2 Spécification de Guard

Pour saisir l'identification d'un garde, on a raffiné la fonction `init()` du service caractère comme suit:

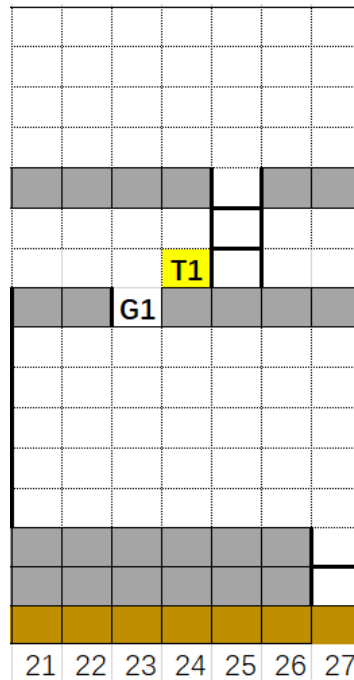
```
public void init(Screen screen, int x, int y, int id);
```

On a aussi ajouté une fonction

```
public ArrayList<Integer> getGuardIdList();
```

Pour stocker tous les id des gardes.

En plus, si le garde tombe dans le trou que le joueur a creusé, et que de plus si la case en dessous de lui est vide, il ne va pas continuer à tomber, mais rester dans le trou. Il peut etre rebouché ou grimper dans 5 seconds



Si le garde est à la fois sur une échelle et sur une case non vide, il va choisir le chemin qui a une distance au joueur est plus courte.

### 3 Spécification de Item

Pour saisir l'identification d'un item et sa propriété, on a raffiné la fonction `init()` de la service caractère à suivant :

```
public void init(Screen screen, int x, int y, int id,
ItemType nature);
```

On a aussi ajouté une fonction

```
public ArrayList<Integer> getItemIdList();
```

Pour stocker tous les id des items.

Dans la classe ItemType

On a ajouté deux types : **ARMS**, **SUPER** pour la partie d'extension du projet.

Super est le mode invincible de joueur, quand il est attrapé par le garde, il ne perd pas de vie. Mais il peut utiliser une seul fois.

Arms est une arme, si le joueur le prend, il peut utiliser Command à attaque les gardes à sa gauche ou à sa droite jusqu'à la fin d'une partie

## 4 Spécification de Joueur

On a ajouté deux commandes pour le joueur, **ATTACKRIGHT** et **ATTACKLEFT**

Il peut attaquer le garde seulement s'il prend item **ARMS**. t il peut seulement tuer le garde juste à côté de lui.



## Les exemples d'exécutions du jeu

### Exemple de Super :

On peut voir en dessus que le joueur se trouve à la case contenant un item Super, quand il est attaqué par le garde1 il ne perd pas de vie, et le garde revient à la position initiale. Mais à la suite du jeu, s'il est attaqué une deuxième fois par le garde1, il perd une vie. Le jeu recommence, maintenant la vie est 2.

```
-----Level1-----
Player Wdt :11 player Hgt :6
Guard 0 Wdt :1 Hgt :14
Guard 1 Wdt :15 Hgt :6
Item0 Wdt :24 Hgt :9
Item1 Wdt :8 Hgt :3
Item2 Wdt :0 Hgt :6
Item3 Wdt :12 Hgt :6
-----
Player marche vers droite
Player Wdt :12 player Hgt :6
Le garde 0 marche vers droite
Guard 0 Wdt :2 Hgt :14
Le garde 1 marche vers gauche
Guard 1 Wdt :14 Hgt :6
Le joueur prend item Super3
-----
Player marche vers droite
Player Wdt :13 player Hgt :6
Le garde 0 marche vers droite
Guard 0 Wdt :3 Hgt :14
Le garde 1 marche vers gauche
Player a ete attape par 1
Guard id :1 a ete attaque le joueur en mode super
Score :0 Life :3
Guard 1 Wdt :15 Hgt :6
-----
Player marche vers droite
Player Wdt :14 player Hgt :6
Le garde 0 marche vers droite
Guard 0 Wdt :4 Hgt :14
Le garde 1 marche vers gauche
Player a ete attape par 1
!!! Recommencer le jeu !!!
Status :Playing Score :0 Life :2
Player Wdt :11 player Hgt :6
Guard 0 Wdt :1 Hgt :14
Guard 1 Wdt :15 Hgt :6
Item0 Wdt :24 item Hgt :9
Item1 Wdt :8 item Hgt :3
Item2 Wdt :0 item Hgt :6
Item3 Wdt :12 item Hgt :6
Guard 1 Wdt :15 Hgt :6
```

### Exemple de arms :

```

-----Level1-----
Player Wdt :23 player Hgt :9
Guard 0 Wdt :0 Hgt :6
Guard 1 Wdt :27 Hgt :9
Item0 Wdt :20 item Hgt :9
Item1 Wdt :21 item Hgt :9
Item2 Wdt :22 item Hgt :9
Playing
-----
Player marche vers gauche
Player Wdt :22 player Hgt :9
Le garde 0 marche vers droite
Guard 0 Wdt :1 Hgt :6
Le garde 1 marche vers gauche
Guard 1 Wdt :26 Hgt :9
Item0 Wdt :20 item Hgt :9
Item1 Wdt :21 item Hgt :9
Le joueur a pris une arme 2
Item2 Wdt :22 item Hgt :9
Playing

```

On peut voir quand le joueur attaque le garde1 qui est à sa droite, le garde réinitialise à (27,9)

```

-----
Player attaque sa droite
Player Wdt :21 player Hgt :9
Le garde 0 monte
Guard 0 Wdt :2 Hgt :9
Le garde 1 marche vers gauche
Player attaque le garde 1
Guard :1 a ete attaque par le joueur
Guard 1 Wdt :27 Hgt :9
Item0 Wdt :20 item Hgt :9
Playing
-----

```

**Exemple le garde prends un trésor et, dépose un trésor quand il tombe.**

Le joueur creuse un trou à sa droite (22,8)

```

-----Level1-----
Player Wdt :21 player Hgt :9
Guard 0 Wdt :2 Hgt :14
Guard 1 Wdt :27 Hgt :9
Item0 Wdt :24 Hgt :9
Item1 Wdt :8 Hgt :3
Item2 Wdt :3 Hgt :15
Item3 Wdt :4 Hgt :4
-----
Hole Wdt :22 Hole Hgt :8
Player creuse a droite
Player Wdt :21 player Hgt :9
Le garde 0 marche vers droite
Guard 0 Wdt :3 Hgt :14
Le garde 1 marche vers gauche
Guard 1 Wdt :26 Hgt :9
-----
Player Wdt :21 player Hgt :9
Le garde 0 marche vers droite
Guard 0 Wdt :4 Hgt :14
Le garde 1 marche vers gauche
Guard 1 Wdt :25 Hgt :9
-----
Player Wdt :21 player Hgt :9
Le garde 0 marche vers droite
Guard 0 Wdt :5 Hgt :14
Le garde 1 marche vers gauche
Guard 1 Wdt :24 Hgt :9
Guard 1 prend le tresor 0 Wdt :24Hgt :9
Item0 Wdt :24 item Hgt :9

```

En 4 étapes, le joueur ne bouge pas. Le garde1 marche à gauche, parce que le joueur est à gauche de lui, il prend le trésor. Après on peut voir le garde tombe dans le trou (22,8), et il libère le trésor à la case au-dessus de lui (22,9)

```

-----
Player marche vers droite
Player Wdt :22 player Hgt :9
Le garde 0 marche vers droite
Guard 0 Wdt :8 Hgt :14
Le garde 1 tombe
temps +1, temps dans le trou est 1
Guard 1 Wdt :22 Hgt :8
Item0 Wdt :22 item Hgt :8
Garde1 libere le tresor
Item0 Wdt :22 item Hgt :9
Le joueur se trouve le tresor0 Wdt :22 Hgt :9
-----
Player marche vers droite
Player Wdt :23 player Hgt :9
Le garde 0 tombe
Guard 0 Wdt :8 Hgt :13
temps +1, temps dans le trou est 2
Guard 1 Wdt :22 Hgt :8
-----
Player marche vers droite
Player Wdt :24 player Hgt :9
Guard 0 Wdt :8 Hgt :13
temps +1, temps dans le trou est 3
Guard 1 Wdt :22 Hgt :8
-----

```

Ensuite, le joueur prend le trésor, le garde attend 5 étapes, après il grimpe à droite :

```

-----
Player marche vers droite
Player Wdt :26 player Hgt :9
Guard 0 Wdt :8 Hgt :13
temps +1, temps dans le trou est 5
Guard 1 Wdt :22 Hgt :8
-----
Player marche vers droite
Player Wdt :27 player Hgt :9
Guard 0 Wdt :8 Hgt :13
Le garde 1 grimbe a droite
Guard 1 Wdt :23 Hgt :9
-----

```

## La description formelle de tests MBT effectué

La série de test Junit, élaborés selon la méthode MBT contient 4 grandes parties.  
Du au très grand nombre d'opération à tester, dans chaque partie de Test, on a choisi que certains d'entre eux.

### 1. Test préconditions :

#### 1.1 Initialisation

- 1.1.1 testInitPreCasPos()
- 1.1.2 testInitPrePlayerCasNeg()
- 1.1.3 testInitPreGuardCasNeg()
- 1.1.4 testInitPreItemCasNeg()

#### 1.2 La service Screen

- 1.2.1 testDigPreScreenCasPos()
- 1.2.2 testDigPreScreenCasNeg()
- 1.2.3 testFillPreScreenCasPos()
- 1.2.4 testFillPreScreenCasNeg()

#### 1.3 La service Environment

- 1.3.1 testgetCellContentPreEnviCasPos()
- 1.3.2 testgetCellContentPreEnviCasNeg()

#### 1.4 La service editableScreen

- 1.4.1 testsetNaturePreEditableScreenCasPos()
- 1.4.2 testsetNaturePreEditableScreenCasNeg()

Pour la service Player et Guard, on va tester l'opération Step() dans la partie de

testTransition et test scénario. D'ici on va tester que les préconditions des opérations climbleft() et climright de la service Guard.

## 1.5 La service Guard

1.5.1 testClimbLeftPreGuardCasNeg()

1.5.2 testClimbRightPreGuardCasNeg()

## 2 Tests transitions

Dans cette partie, on teste les post-conditions et Invariants des opérations choisies

### 2.1 EditableScreen

2.1.1 testInitEditableScreenTrans()

2.1.2 testSetNatureEditableScreenTrans()

### 2.2 Environment

2.2.1 testInitEnviTrans()

### 2.3 Player(Caractère)

D'ici on utilise la service Player pour tester les opérations de la service Caractère goLeft(),goRight(),goUp() et goDown()

2.3.1 testGoLeftPlayerTrans()

2.3.2 testGoRightPlayerTrans()

2.3.3 testGoUpPlayerTrans()

2.3.4 testGoDownPlayerTrans()

2.3.5 testDigLeftPlayerTrans()

2.3.6 testDigRightPlayerTrans()

### 2.4 Guard

2.4.1 testInvariantGuardTrans()

2.4.2 testClimbRightGuardTrans()

## 3 Etats remarquables

### 1 TestEtatsRemarquables1()

On maximise les coordonnées de Player à (13,13), il est donc initialisé dans la dernière case de d'écran. Ensuite, on initialise un garde à (12,13) et un trésor à (13,13)

Après le joueur marche vers sa droite, il va rester à (13,13), parce que la limite de longueur d'écran est 14.

Ensuite, le garde va l'attraper.

### 2 TestEtatsRemarquables2()

Dans ce cas, on déplace le joueur dans la case (0,2), et il marche 2 étapes de vers la gauche.

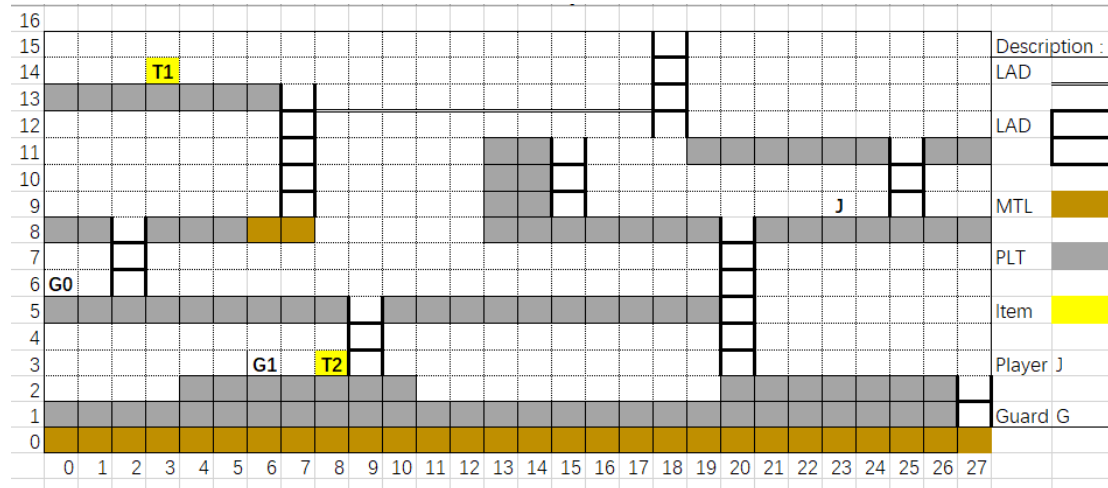
On peut trouver que le joueur reste à (0,2).

La position initiale de garde est (3,2), après 2 étapes il est à (1,2) et il a pris le trésor lequel était dans la case (2,2). Donc, on observe que le trésor se déplace avec le garde.

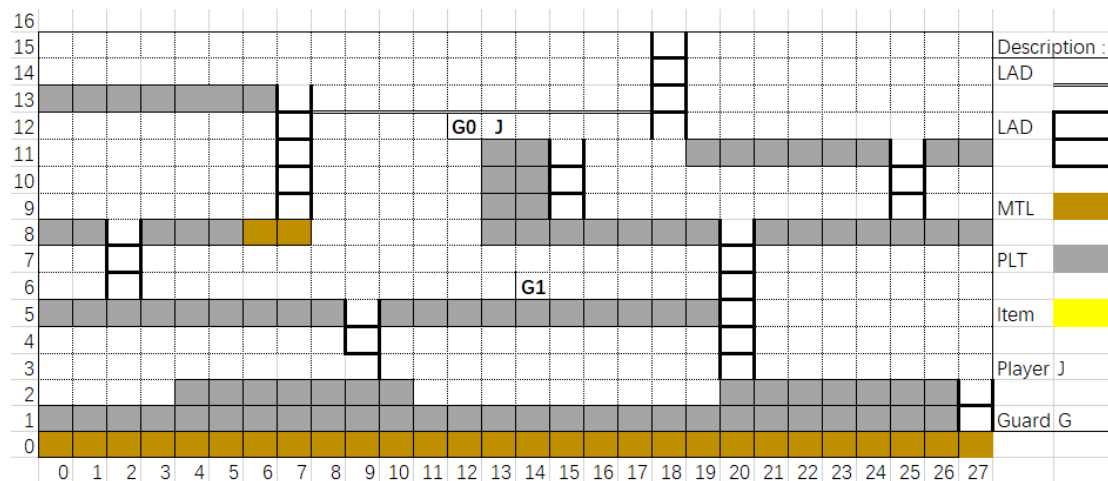
#### 4 Test scénario

##### 4.1 TestScenario1 ()

Maintenant, on teste une série d'opérations du jeu. La cadre est en suivant :



On peut voir que Gard 1 a pris le trésor 1 sur la case (8,3), après le trésor a déplacé avec lui jusqu'à (14,6), et le joueur a été attrapé sur l'échelle (13,12)



Le résultat affiche :

```

-----
Player marche vers gauche
Player Wdt :13 player Hgt :12
Le garde 0 marche vers droite
Player a ete attape par 0
!!! Recommencer le jeu !!!
Status :Playing Score :0 Life :2
Player Wdt :23 player Hgt :9
Guard 0 Wdt :0 Hgt :6
Guard 1 Wdt :6 Hgt :3
Item0 Wdt :3 item Hgt :14
Item1 Wdt :8 item Hgt :3
Guard 0 Wdt :0 Hgt :6
Le garde 1 marche vers droite
Guard 1 Wdt :7 Hgt :3
-----

```

## 1.1 TestSenario2 ()

# Les difficultés du projet

La première difficulté est de synchroniser les observateurs à l'environnement.

On doit mettre à jour à chaque étape, toutes les positions de joueur, de gardes, de trésor et le trou si le joueur creuse à gauche ou à droite.

Ou bien si le joueur attaque ou est attaqué par le garde, on doit réinitialiser à leurs positions initiales.

Donc, dans le moteur du jeu EngineImpl, dans un premier temps, quand tous les items et caractères sont initialisées, on a utilisé les listes de Java pour stocker les informations. Par exemple pour le garde, il y a une liste `public ArrayList<Guard> guardlist;` utilisé pour stocker les services guards, une autre liste pour stocker leurs positions initiales :

```
public ArrayList<Pair<Integer,Integer>> guardinitlist;
```

Une fois qu'on veut initialiser un caractère on peut prendre des coordonnées x et y et utiliser `init()` pour créer un nouveau caractère mais avec ses les positions initiales.

En plus, on a utilisé `HashMap` pour stocker les `CellContent` et les coordonnées de cette case.

```
public HashMap<CellContent, Pair<Integer, Integer>> cellMap;
```

Elle a été mise à jour aussi à chaque étape, et a été envoyé par le moteur à la service Environnement

Dans notre jeu, le garde peut récupérer le trésor, et il va libérer quand il tombe dans le trou. Pour réussir cette opération, on crée un autre `HashMap`

La clé de `HolesMap` est les coordonnées de trou et la valeur de clé est le temps trou depuis sa création

```
public HashMap<Pair<Integer, Integer>, Integer> holesMap;
```

Dans la classe d'implémentation de la service Player, PlayerImpl, si joueur creuse à gauche ou à droite, on va initialiser les nouveaux trous, ensuite il va créer la liaison avec le monte, et envoyer les informations au monte.

```
if (NextCommand() == Command.DIGR) {  
    if (((getEnvi().CellNature(getWdt(), getHgt()-1) != Cell.HOL)  
        && (getEnvi().CellNature(getWdt(), getHgt()-1) != Cell.EMP))  
        || (!getEnvi().getCellContent(getWdt(), getHgt()-1).isEmpty())) {  
        if (getEnvi().CellNature(getWdt()+1, getHgt()-1) == Cell.PLT) {  
            getEnvi().Dig(getWdt()+1, getHgt()-1);  
            //appeler holes  
            Holes hole = new HolesImpl();  
            hole.init(getEnvi(), getWdt()+1, getHgt()-1, 0);  
            engine.bindHolesService(hole);  
            System.out.println("Player creuse a droite");  
        }  
    }  
}
```

Quand le moteur reçoit les informations des trous, il ajoute les coordonnées et le temps de trous dans le HolesMap, s'il un personnage tombe dans le trou qui existe dans HolesMap, on met à jour la valeur de Map, et vérifier si le temps est égal au temps limite reboucher, si oui, on met à jour aussi le personnage.