Devoir de spécification en CPS du jeu : **Lode Runner**

GHALLAB Karim - PIDERY Ale

27 octobre 2019

Introduction

Ce rapport décrit le projet de spécification du jeu ***Lode Runner***.

La gestion du code source

Pour une gestion efficace du code produit durant le devoir, nous avons mis en place un dépôt *GitHub*.
Le dépôt est accessible à cette adresse : https://github.com/apidery/Lode_Runner/ .

Voici la table des matières de notre rapport :

Table des matières

1 Manuel d'utilisation

Dans ce manuel utilisateur, nous indiquerons comment exécuter notre jeu avant de présenter le gameplay de celui-ci.

1.1 Démarrer le jeu

La réalisation du jeu s'est faite en *Java*. Afin de pouvoir facilement compiler et exécuter le jeu, nous avons réalisé à la racine du projet un fichier `build.xml`.

Compiler le jeu

La compilation du jeu se fait avec la commande :

```
ant build
```

Cette commande compilera l'ensemble des sources présentes dans le dossier `src` du projet et placera les binaires résultant de cette compilation dans le dossier `bin`.

Exécuter les tests du jeu

L'exécution de l'ensemble des tests de notre jeu se fait avec la commande :

```
ant junit-tests
```

Cette commande exécutera l'ensemble des classes de tests présentes dans le dossier `src/tests/`. Pour chaque classe de test, un rapport au format texte sera généré et placé dans le dossier `reports` de la racine du projet. Ces rapports permettent ainsi de constater du bon déroulement ou non des tests exécutés.

Lancer les mains du jeu

Pour ce projet, deux *mains* différents on été réalisés. La différence entre ces deux *mains* et que l'un utilisera directement les classes d'implantation du jeu (présentes dans le dossier `src/impl/`, alors que l'autre utilisera les contrats du jeu (présents dans le dossier `src/contracts/`.
L'exécution du main utilisant directement les implantations se fait avec la commande :

```
ant run
```

L'exécution du main utilisant directement les contrats se fait quant à lui avec la commande :

```
ant run-contracts
```

Ces deux commandes exécuteront leurs classes *mains* respectives. Une fois le jeu lancé, une fenêtre similaire à celle présentée en figure 1 devrait apparaître.

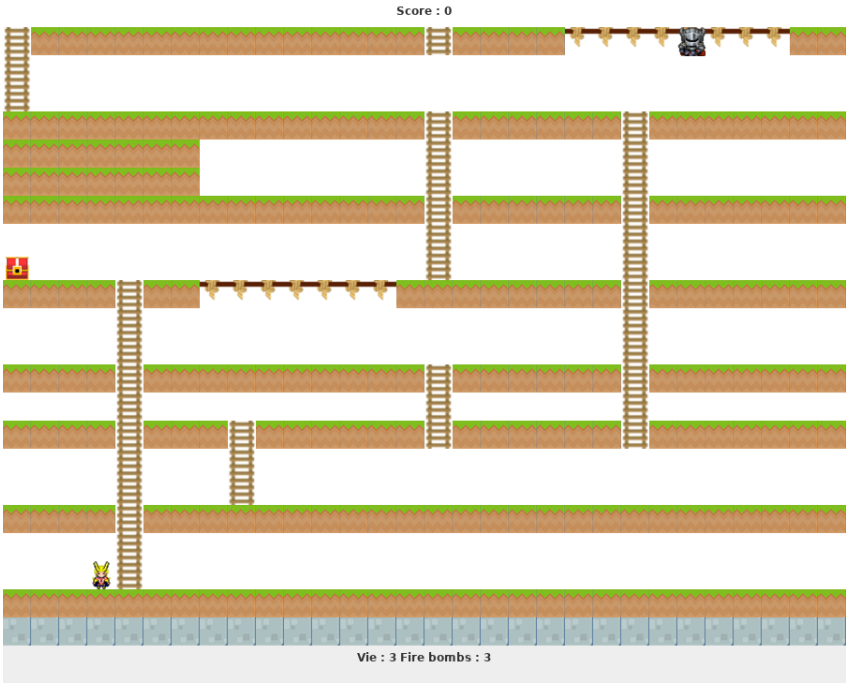

FIGURE 1 – fenêtre graphique du jeu.

Une implémentation "buggée" nous a également été demandé. Afin de lancer le main utilisant les services comportant des bugs, il faut exécuter la commande :

```
ant run-contracts-bugs
```

Maintenant que le jeu est lancé, nous allons pouvoir vous présenter plus en détail les différents éléments affichés sur la fenêtre graphique.

1.2 Présentation de l'interface du jeu

Au sommet de l'interface graphique du jeu se situe un label affichant votre score actuel. Ce score augmentera pour chaque trésor que vous amasserez durant la partie.
Les labels présents en bas de la fenêtre indiquent votre niveau de vie actuel ainsi que le nombre d'attaques vous restant. Nous reviendrons plus en détail sur la gestion du niveau de vie et des attaques dans la partie intitulée "Gameplay du jeu".

1.2.1 Les personnages

All Might

All Might (ou *Toshinori Yagi*) est un des personnages principaux du manga My Hero Academia. Ici, il est aussi le personnage que vous incarnerez durant l'ensemble de la partie.
Sur la fenêtre graphique, *All Might* est représenté par le sprite de la figure 2.

FIGURE 2 – Sprite d'All Might, le symbole du la paix.

Les gardes

Les gardes quant à eux sont les gardiens des trésors que vous essayez de dérober. Dès l'entame de la partie, ils vous poursuivront pour vous infliger des dégâts qu'ils espèrent fatals.
Les gardes sont représentés par le sprite de la figure 3.

FIGURE 3 – Sprite d'un garde.

La présentation des gardes de l'interface conclue cette présentation des personnages, nous allons désormais passer à la présentation des différents types de cases dont qui constituent un niveau de notre partie.

1.2.2 Les types de cases

Une case de type PLT

Les cases du type PLT représentent des plateaux, ce sont des cases sur lesquelles vous pourrez vous déplacer et même creusez afin de piéger vos adversaires.
Une case du type PLT est représentée par le sprite de la figure 4.

FIGURE 4 – Sprite d'un PLT.

Une case de type MTL

Les cases du type MTL représentent des cases métalliques, à l'instar des PLT, vous pourrez vous déplacer librement sur celle-ci, mais inutile d'essayer de creuser dedans, celles-ci sont trop solides.
Les MTL sont représentées par le sprite de la figure 5.

FIGURE 5 – Sprite d'un MTL.

Une case de type LAD

Les *ladders* (ou échelles) permettent à n'importe quel personnage de pouvoir se déplacer vers le haut ou le bas du terrain.
Les échelles sont représentées par le sprite de la figure 6.

FIGURE 6 – Sprite d'un LAD.

Une case de type HDR

Les *handrails* (ou balustrades) permettent à n'importe quel personnage de se déplacer librement horizontalement sur le terrain sans qu'un PLT ou MTL soit sous vos pieds.
Les balustrades sont représentées par le sprite de la figure 7.

FIGURE 7 – Sprite d'un HDR.

Une case de type EMP ou HOL

Les EMP (*empty*) ou HOL(*trou*) n'ont pas de sprite associé, les cases de ces types-là seront donc de la même couleur que le fond de l'interface.

Ceci conclut la présentation des différents types de cases d'un terrain. Nous allons désormais présenter les autres éléments graphiques que vous pourrez trouver durant la partie.

1.2.3 Les autres éléments

Les trésors

Les trésors sont vos objectifs dans une partie, ce sont eux que vous allez devoir essayer de collecter sans vous faire piéger par les gardes environnants.
Un trésor est représenté par le sprite de la figure 8.

FIGURE 8 – Sprite d'un trésor.

Les boules de feu

Les boules de feu seront vos plus fidèles atouts, et avant de vous présenter comment celles-ci fonctionnent dans la partie numéro 1.3, commençons ici par vous présenter leur forme dans notre jeu.
Une boule de feu est représentée par le sprite de la figure 9.

FIGURE 9 – Sprite d'une boule de feu.

Les portails

Les portails sont des entités assez mystiques. Nous ne savons pas vraiment comment ils fonctionnent, mais ceux-ci n'apparaissent que si vous avez récolté l'ensemble des trésors du niveau courant et vous transporteront au prochain niveau.
Nous vous en dirons plus sur les portails dans la prochaine section de ce manuel d'utilisation.
Un portail est représenté graphiquement par le sprite de la figure 10.

.

FIGURE 10 – Sprite d'un portail.

Les éléments graphiques de notre jeu ont été présentés, nous allons pouvoir vous introduire au gameplay de notre jeu.

1.3 Gameplay du jeu

1.3.1 Objectif d'une partie

L'objectif d'une partie est simple, récolter l'ensemble des trésors présents sur le terrain afin de faire apparaître un portail. Une fois ce dernier apparu, empruntez le afin qu'il vous amène au niveau suivant. Au début d'un niveau vous disposerez de trois points de vie et de trois boules de feu. À chaque contact avec un garde, vous perdrez un point de vie. Il en est de même si vous restez dans un trou et que celui-ci se rebouche (dans ce cas-là, vous retrouverez votre position de départ dans le niveau courant).
Les boules de feu quant à elles vous permettent d'infliger de lourds dégâts aux gardes. Celles-ci se déplacent horizontalement sur le terrain. Si un garde croise leurs passages, celui-ci retournera instantanément à sa position initiale. Si vous empruntez un portail afin de passer au niveau suivant, tous vos points de vie et vos boules de feu vous seront restitués.
Maintenant que l'objectif a été présenté, nous allons parler plus en détail des commandes à réaliser pour pouvoir utiliser le plein potentiel de votre personnage.

1.3.2 Les actions du personnage

Bouger

Afin de mouvoir votre personnage comme il se doit, vous pouvez utiliser les flèches directionnelles de votre clavier. Ainsi, une pression sur la flèche `"Droite"` du clavier (respectivement `"Gauche"`) fera bouger votre personnage vers la droite du terrain (respectivement vers la gauche). Il en est de même pour les touches `"Haut"` et `"Bas"`.

Creuser un trou

Pouvoir creuser un trou est une action essentielle à maîtriser afin de pouvoir atteindre votre objectif. En effet, si un garde tombe dedans il sera immobilisé pendant quelques secondes. Il se peut même que le trou se referme sur le garde qui devra alors retourner à sa position initiale.
Attention tout de même a ce que votre technique ne se retourne pas contre vous. Si par mégarde vous tombiez dans un trou, essayez de vite en sortir avec les touches `"Gauche"` et `"Droite"` de votre clavier.
Afin de creuser un trou vers la droite (respectivement vers la gauche), appuyez sur la touche `"Z"` (respectivement `"A"`) de votre clavier.

Lancer une boule de feu

Une boule de feu est une attaque extrêmement puissante. Comme décrit plus haut, celles-ci renverront quiconque aura le malheur de croiser leurs chemins à sa position initiale. Une fois qu'une boule de feu est entrée en contact avec un ennemi ou une case non vide, la bombe en question disparaîtra.
Afin de lancer une boule de feu vers la droite (respectivement vers la gauche), appuyez sur la touche `"S"` (respectivement `"Q"`) de votre clavier.

Ceci conclut le manuel d'utilisation de notre jeu.

2 Spécification formelle du projet

Dans cette partie nous présenterons l'ensemble des spécifications de notre projet. Chaque sous-section correspondra à la spécification formelle d'un service à part entière.
Nous commencerons avec le spécification du service *Screen*

2.1 Screen

Service : Screen
Observators : **const** Height : [Screen] → int
const Width : [Screen] → int
CellNature : [Screen] × int × int → Cell
pre CellNature(S, x, y) **requires** $0 \leq y <$ Height(S) **and** $0 \leq x <$ Width(S)
Constructors : init : int × int → [Screen]
pre init(h, w) **requires** h > 0 **and** $w > 0$
Operators : Dig : [Screen] × int × int → [Screen]
pre Dig(S, x, y) **requires** CellNature(S, x, y) ∈ {PLT}
Fill : [Screen] × int × int → [Screen]
pre Fill(S, x, y) **requires** CellNature(S, x, y) ∈ {HOL}
Observations :
[init] : Height(init(h, w)) = h
Width(init(h, w)) = w
forall x **in** [0 :Width(S)]
forall y **in** [0 :Height(S)]
CellNature(init(h, w), x, y) ∈ {EMP}

[Dig] : CellNature(Dig(S, x, y), x, y) ∈ {HOL}
forall u **in** [0 :Width(S)]
forall v **in** [0 :Height(S)]
$(x \neq u)$ **or** $y \neq v$ **implies** CellNature(Dig(S, u, v), x, y) = CellNature(S, x, y)

[Fill] : CellNature(Fill(S, x, y), x, y) ∈ {PLT}
forall u **in** [0 :Width(S)]
forall v **in** [0 :Height(S)]
$(x \neq u)$ **or** $y \neq v$ **implies** CellNature(Fill(S, u, v), x, y) = CellNature(S, x, y)

2.2 EditableScreen

Service : EditableScreen **includes** Screen
Observators : Playable : [EditableScreen] → bool
Operators : SetNature : [EditableScreen] × int × int × Cell → [EditableScreen]
pre SetNature(S, x, y, C) **requires** $0 \leq$ y $<$ Height(S) **and** $0 \leq$ x $<$ Width(S)
Observations :
[invariant] : Playable(S) **min**
forall x **in** [0 :Width(S)]
forall y **in** [0 :Height(S)]
CellNature(S, x, y) ∉ {HOL}
and forall x **in** [0 :Width(S)]
CellNature(S, x, 0) = {MTL}

[SetNature] : CellNature(SetNature(S, x, y, C), x, y) = C
forall v **in** [0 :Height(S)]

$(x \neq u)$ **or** $y \neq v$ **implies** CellNature(SetNature(S, u, v, C), x, y) = CellNature(S, x, y)

[PlayableEnvi] : **WillReturnPlayableCells(E) defined by**
for x **in** [0 :Width(S)]
for y **in** [0 :Height(S)]
$\Longrightarrow$ CellNature(S,x,y) != HOL
and
for x in [0 :Width(S)]
CellNature(S,x,0) = MTL

2.3 Environnement

Service : Environnement **includes** Screen
Observators : **const** Player : [Environnement] → Player
const Guards : [Environnement] → List<Guard>
const Treasures : [Environnement] → List<Item>
const HgtPortal : [Environnnement] → int
const WdtPortal : [Environnement] → int
CellContent : [Environnement] × int × int → Couple
pre CellContent(E, x, y) **requires** 0 ≤ y < Height(S) **and** 0 ≤ x < Width(S)
Constructors : init : int × int × Player × List<Guard> × List<Item> × int × int → [Environnement]
init : int × int × Player × List<Guard> × List<Item> × Cell[][] × int × int → [Environnement]
pre init(x, y, player, guards, treasures, wdtP, hgtP) **requires** lenght(cells) = wdt **and** lenght(cells[0]) = hgt
Observations :
[invariant] : **forall** x **in** [0 :Width(E)]
forall y **in** [0 :Height(E)]
forall Character c1, c2 in CellContent(E, x, y), c1 = c2

forall x **in** [0 :Width(E)]
forall y **in** [0 :Height(E)]
CellNature(E,x,y) ∈ {MTL,PLT} $\Longrightarrow$ CellContent(E,x,y) = null

forall x **in** [0 :Width(E)]
forall y **in** [0 :Height(E)]
CellConten(E,x,y) = Treasure t $\Longrightarrow$ CellNature(E,x,y) ∈ {EMP} **and** CellNature(E,x,y-1) ∈ {MTL,PLT}

[init] : Player(init(wdt,hgt,player,guards,treasures, wdtPortal, hgtPortal)) = player
WdtPortal(init(wdt, hgt, player, guards, treasures, wdtPortal, hgtPortal)) = wdtPortal
HgtPortal(init(wdt, hgt, player, guards, treasures, wdtPortal, hgtPortal)) = hgtPortal
for i in [0 :List::Size(guards)]
List::Get(Guards(init(wdt, hgt, player, guards, treasures)), i) = List::Get(guards, i)
for i in [0 :List::Size(treasures)]
List::Get(Treasures(init(wdt, hgt, player, guards, treasures)), i) = List::Get(treasures, i)

[init] : Player(init(wdt,hgt,player,guards,treasures,Cell[][] cells)) = player
WdtPortal(init(wdt, hgt, player, guards, treasures, wdtPortal, hgtPortal)) = wdtPortal

HgtPortal(init(wdt, hgt, player, guards, treasures, wdtPortal, hgtPortal)) = hgtPortal
for i in [0 :List::Size(guards)]
    List::Get(Guards(init(wdt, hgt, player, guards, treasures, cells)), i) = List::Get(guards, i)
for i in [0 :List::Size(treasures)]
    List::Get(Treasures(init(wdt, hgt, player, guards, treasures, cells)), i) = List::Get(treasures, i)
for i in [0 :wdt]
    for j in [0 :hgt]
        cells[i][j] = CellNature(E,i,j)

2.4 Character

Service : Character

Observators :
const Envi : [Character] → Environnement
const InitialWdt : [Guard] → int
const InitialHgt : [Guard] → int
Wdt : [Character] → int
Hgt : [Character] → int
LastAction : [Character] → Command

Constructors :
init : Screen × int × int
    pre init(x, y) **requires** Screen::CellNature(S,x,y) ∈ {EMP}

Operators :
GoLeft : [Character] → Character
GoRight : [Character] → Character
GoUp : [Character] → Character
GoDown : [Character] → Character

Observations :

[invariant] :
Screen::CellNature(Envi(C), Wdt(C), Hgt(C)) ∈ {EMP, HOL, LAD, HDR}

[init] :
Wdt(init(e,x,y)) = x
Hgt(init(e,x,y)) = y
Envi(init(e,x,y)) = e
InitialWdt(init(e, x, y)) = x
InitialHgt(init(e, x, y)) = y

[goLeft] :
LastAction(GoLeft(C)) = LEFT

Wdt(C) = 0 ⟹ Wdt(GoLeft(C)) = Wdt(C) **and** Hgt(GoLeft(C)) = Hgt(C)

Screen::CellNature(Envi(C), Wdt(C)-1, Hgt(C)) ∈ {MTL,PLT} **and**
Screen::CellNature(Envi(C), Wdt(C), Hgt(C)) != HDR ⟹
    Wdt(GoLeft(C)) = Wdt(C) **and** Hgt(GoLeft(C)) = Hgt(C)

Screen::CellNature(Envi(C), Wdt(C), Hgt(C)) ∉ {LAD, HDR}
and Screen::CellNature(Envi(C),Wdt(C),Hgt(C)-1) ∉ {PLT, MTL}
and ∄ Character **in** Environnement::CellContent(Envi(C),Wdt(C),Hgt(C)-1)
⟹ Wdt(GoLeft(C)) = Wdt(C) **and** Hgt(GoLeft(C)) = Hgt(C)

∃ Character **in** Environnement::CellContent(Envi(C), Wdt(C)-1, Hgt(C)) ⟹
    Wdt(GoLeft(C)) = Wdt(C) **and** Hgt(GoLeft(C)) = Hgt(C)

Wdt(C) != 0
and Screen::CellNature(Envi(C), Wdt(C)-1, Hgt(C)) ∉ {MTL,PLT}
and (Screen::CellNature(Envi(C), Wdt(C), Hgt(C)) ∈ {LAD,HDR}
    or Screen::CellNature(Envi(C), Wdt(C), Hgt(C)-1) ∈ {MTL,PLT,LAD}

or ∃ Character in Environnement::CellContent(Envi(C), Wdt(C), Hgt(C)-1))
and
∄ Character in Environnement::CellContent(Envi(C), Wdt(C)-1, Hgt(C))
$\implies$ Wdt(GoLeft(C)) = Wdt(C)-1

Screen::CellNature(Envi(C), Wdt(C), Hgt(C)) != HDR
and Screen::CellNature(Envi(C), Wdt(C)-1, Hgt(C)-1) = HDR
$\implies$ Hgt(GoLeft(C)) = Hgt(C)-1

Screen::CellNature(Envi(C),Wdt(C),Hgt(C)) = HDR
and Screen::CellNature(Envi(C), Wdt(C)-1, Hgt(C)) ∈ {PLT,MTL}
and Screen::CellNature(Envi(C), Wdt(C) - 1, Hgt(C) + 1) ∈ {EMP, LAD, HDR, HOL})
and ∄ Character in Environnement::CellContent(Envi(C), Wdt(C) - 1, Hgt(C) + 1)
$\implies$ Hgt(GoLeft(C)) = Hgt(C)+1 **and** Wdt(C)-1

[goRight] : LastAction(GoRight(C)) = RIGHT

Wdt(C) = Screen : :Width(Envi(C)) - 1 $\implies$ Wdt(GoRight(C)) = Wdt(C) **and** Hgt(GoRight(C)) = Hgt(C)

Screen::CellNature(Envi(C), Wdt(C) + 1, Hgt(C)) ∈ {MTL,PLT} **and**
Screen::CellNature(Envi(C), Wdt(C), Hgt(C)) != HDR $\implies$
Wdt(GoRight(C)) = Wdt(C) **and** Hgt(GoRight(C)) = Hgt(C)

Screen::CellNature(Envi(C), Wdt(C), Hgt(C)) ∉ {LAD, HDR}
and Screen::CellNature(Envi(C),Wdt(C),Hgt(C)-1) ∉ {PLT, MTL}
and ∄ Character **in** Environnement::CellContent(Envi(C),Wdt(C),Hgt(C)-1)
$\implies$ Wdt(GoRight(C)) = Wdt(C) **and** Hgt(GoRight(C)) = Hgt(C)

∃ Character **in** Environnement::CellContent(Envi(C), Wdt(C)+1, Hgt(C)) $\implies$
Wdt(GoRight(C)) = Wdt(C) **and** Hgt(GoRight(C)) = Hgt(C)

Wdt(C) != Screen : :Width(Envi(C)) - 1
and Screen::CellNature(Envi(C), Wdt(C)+1, Hgt(C)) ∉ {MTL,PLT}
and (Screen::CellNature(Envi(C), Wdt(C), Hgt(C)) ∈ {LAD,HDR}
or Screen::CellNature(Envi(C), Wdt(C), Hgt(C)-1) ∈ {MTL,PLT,LAD}
or ∃ Character in Environnement::CellContent(Envi(C), Wdt(C), Hgt(C)-1))
and
∄ Character in Environnement::CellContent(Envi(C), Wdt(C)+1, Hgt(C))
$\implies$ Wdt(GoRight(C)) = Wdt(C)+1

Screen::CellNature(Envi(C), Wdt(C), Hgt(C)) != HDR
and Screen::CellNature(Envi(C), Wdt(C)+1, Hgt(C)-1) = HDR
$\implies$ Hgt(GoRight(C)) = Hgt(C)-1

Screen::CellNature(Envi(C),Wdt(C),Hgt(C)) = HDR
and Screen::CellNature(Envi(C), Wdt(C)+1, Hgt(C)) ∈ {PLT,MTL}
and Screen::CellNature(Envi(C), Wdt(C)+1, Hgt(C)+1) ∈ {EMP, LAD, HDR, HOL})
and ∄ Character in Environnement::CellContent(Envi(C), Wdt(C)+1, Hgt(C) + 1)
$\implies$ Hgt(GoLeft(C)) = Hgt(C)+1 **and** Wdt(C)+1

[GoUp] : LastAction(GoUp(C)) = UP

Wdt(GoUp(C)) = Wdt(C)

Hgt(C) = Screen : :Height(Envi(C)) - 1 $\implies$ Hgt(GoUp(C)) = Hgt(C)
$\exists$ Chracter in Environnement : :CellContent(Envi(C), Wdt(C), Hgt(C) + 1)
$\implies$ Hgt(GoUp(C)) = Hgt(C)

(Screen : :CellNature(Envi(C), Wdt(C), Hgt(C)) = LAD) **and**
(Screen : :CellNature(Envi(C), Wdt(C), Hgt(C) + 1) = PLT)
$\implies$ Hgt(GoUp(C)) = Hgt(C)

(Screen : :CellNature(Envi(C), Wdt(C), Hgt(C)) = LAD) **and**
(Screen : :CellNature(Envi(C), Wdt(C), Hgt(C) + 1) $\in$ {LAD, EMP}) **and**
$\nexists$ Character in (Environnement : :CellContent(Envi(C), Wdt(C), Hgt(C) + 1)
$\implies$ Hgt(GoUp(C)) = Hgt(C) + 1

[GoDown] : LastAction(GoDown(C)) = DOWN

Wdt(GoDown(C)) = Wdt(C)

Hgt(C) = 1 $\implies$ Hgt(GoDown(C)) = Hgt(C)
$\exists$ Chracter in Environnement : :CellContent(Envi(C), Wdt(C), Hgt(C) - 1)
$\implies$ Hgt(GoDown(C)) = Hgt(C)

(Screen : :CellNature(Envi(C), Wdt(C), Hgt(C)) = LAD) **and**
(Screen : :CellNature(Envi(C), Wdt(C), Hgt(C) - 1) = PLT)
$\implies$ Hgt(GoDown(C)) = Hgt(C)

(Screen : :CellNature(Envi(C), Wdt(C), Hgt(C)) = $\in$ {LAD, EMP, HDR}) **and**
(Screen : :CellNature(Envi(C), Wdt(C), Hgt(C) - 1) $\in$ {LAD, EMP, HDR, HOL}) **and**
$\nexists$ Character in (Environnement : :CellContent(Envi(C), Wdt(C), Hgt(C) - 1)
$\implies$ Hgt(GoDown(C)) = Hgt(C) - 1

[BackInitialPosition] : Wdt(BackInitialPosition(C)) = initialWdt(C) **and** Hgt(BackInitialPosition(C)) = initialHgt(C)

2.5 Player

Service : Player **includes** Character
Observators : **const** Engine : [Player] $\rightarrow$ Engine
Vie : [Player] $\rightarrow$ int
NbAttacks : [Player] $\rightarrow$ int
Constructors : init : Screen $\times$ int $\times$ int $\times$ int $\times$ Engine $\rightarrow$ [Player]
Operators : step : [Player] $\rightarrow$ [Player]
decreaseVie : [Player] $\rightarrow$ [Player]
pre decreaseVie(P) **requires** Vie(P) $>$ 0
decreaseNbAttacks : [Player] $\rightarrow$ [Player]
pre decreaseNbAttacks(P) **requires** NbAttacks(P) $>$ 0
Observations :
[init] : Engine(init(e, x, y, engine)) = engine
Vie(init(e, x, y, engine)) = 3
NbAttacks(init(e, x, y, engine)) = 3

[step] : WillGoDown **definedby** :
Environnement::CellNature(Envi(P), Wdt(P), Hgt(P)) $\notin$ {HDR, LAD, HOL}

and Environnement::CellNature(Envi(P), Wdt(P), Hgt(P) - 1) ∈ {EMP, HOL, HDR}
and ∄ Character in Environnement::CellContent(Envi(P), Wdt(P), Hgt(P) - 1)
⟹ LastAction(Step(P)) = DOWN
and Step(P) = GoDown(P)
or definedby :
Engine::NextCommand(P) = DOWN
⟹ LastAction(Step(P)) = DOWN
and Step(P) = GoDown(P)
WillGoRight **definedby** :
Engine::NextCommand(P) = RIGHT
⟹ LastAction(Step(P)) = RIGHT
and Step(P) = GoRight(P)
WillGoLeft **definedby** :
Engine::NextCommand(P) = LEFT
⟹ LastAction(Step(P)) = LEFT
and Step(P) = GoLeft(P)
WillGoUp **definedby** :
Engine::NextCommand(P) = UP
⟹ LastAction(Step(P)) = UP
and Step(P) = GoUp(P)
WillDigLeft **definedby** :
(Engine::NextCommand(P) = DigL
and (Environnement::CellNature(Envi(P), Wdt(P), Hgt(P) - 1) ∈ {PLT, MTL, LAD} **or** ∃ Environnement::CellContent(Envi(P), Wdt(P), Hgt(P) - 1)))
and (Environnement::CellNature(Envi(P), Wdt(P) - 1, Hgt(P)) = EMP
and ∄ Character in Environnement : :CellContent(Envi(P), Wdt(P) - 1, Hgt(P))
and ∄ Treasure in Environnement::CellContent(Envi(P), Wdt(P) - 1, Hgt(P)))
and Environnement::CellNature(Envi(P), Wdt(P) - 1 , Hgt(P) - 1) = PLT
⟹ Environnement::dig(Envi(P), Wdt(P) - 1, Hgt(P) - 1)
and Engine::AddHole(Wdt(P) - 1, Hgt(P) - 1)
and LastAction(Step(P)) = DigL
WillDigRight **definedby** :
(Engine::NextCommand(P) = DigR
and (Environnement::CellNature(Envi(P), Wdt(P), Hgt(P) - 1) ∈ {PLT, MTL, LAD} **or** ∃ Environnement::CellContent(Envi(P), Wdt(P), Hgt(P) - 1)))
and (Environnement::CellNature(Envi(P), Wdt(P) + 1, Hgt(P)) = EMP
and ∄ Character in Environnement::CellContent(Envi(P), Wdt(P) + 1, Hgt(P))
and ∄ Treasure in Environnement::CellContent(Envi(P), Wdt(P) + 1, Hgt(P)))
and Environnement::CellNature(Envi(P), Wdt(P) + 1 , Hgt(P) - 1) = PLT
⟹ Environnement::dig(Envi(P), Wdt(P) + 1, Hgt(P) - 1)
and Engine::AddHole(Wdt(P) + 1, Hgt(P) - 1)
and LastAction(Step(P)) = DigL

[decreaseVie] :
Vie(DecreaseVie(E)) = Vie(E) - 1
Wdt(DecreaseVie(E)) = Wdt(E)
Hgt(DecreaseVie(E)) = Hgt(E)
NbAttack(DecreaseVie(E)) = NbAttack(E)
Screen(DecreaseVie(E)) = Screen(E)
Engine(DecreaseVie(E)) = Engine(E)

[decreaseNbAttacks] :
Vie(DecreaseNbAttacks(E)) = Vie(E)
Wdt(DecreaseNbAttacks(E)) = Wdt(E)
Hgt(DecreaseNbAttacks(E)) = Hgt(E)
NbAttack(DecreaseNbAttacks(E)) = NbAttack(E) - 1
Screen(DecreaseNbAttacks(E)) = Screen(E)

Engine(DecreaseNbAttacks(E)) = Engine(E)

2.6 Engine

Service : Engine

Observators :
const Player : [Engine] → Player
const nbTreasureTotal : [Engine] → int
const Board : [Engine] → Board
const Levels : [Engine] → Map<Integer,Environnement>
const Editors : [Engine] → Map<Integer,EditableScreen>
const PathResolver : [Engine] → Map<Integer, PathResolver>
const ScreenWdt : [Engine] → int
const ScreenHgt : [Engine] → int
const Frame : [Engine] → LodeRunnerFrame
const PlayableLevels : [Engine] → Map<Integer, Boolean>
Guards : [Engine] → List<Guard>
Treasures : [Engine] → List<Item>
CurrentEnvironnement : [Engine] → Environnement
CurrentPathResolver : [Engine] → PathResolver
CurrentNbTreasures : [Engine] → int
NextCommand : [Engine] → Command
Status : [Engine] → Status
Holes : [Engine] → Map<GameObject, Integer>
Score : [Engine] → int
NbStep : [Engine] → int
CurrentLevel : [Engine] → int
Attacks : [Engine] → List<Attack>

Constructors :
init : int × int × int × int → [Engine]
pre init(wdtScreen, hgtScreen, playerX, playerY) **requires** 0 < playerX < wdtScreen **and** 0 < playerY < hgtScreen

initWithContract : int × int × int × int → [Engine]
pre initWithContract(wdtScreen, hgtScreen, playerX, playerY) **requires** 0 < playerX < wdtScreen **and** 0 < playerY < hgtScreen

initForTest : int × int × int × int → [Engine]
pre initForTest(wdtScreen, hgtScreen, playerX, playerY) **requires** 0 < playerX < wdtScreen **and** 0 < playerY < hgtScreen

Operators :
AddHole : [Engine] × int × int → [Engine]
pre AddHole(E, x, y)
requires 0 ≤ x < Environnnement::Wdt(CurrentEnvironnement(E))
and 0 ≤ y < Environnnement::Hgt(CurrentEnvironnement(E))
and ∄ hole in Holes(E) **with** GameObject::Wdt(hole) = x
and GameObject::Hgt(hole) = y

Step : [Engine] → [Engine]
pre Step(E) **requires** Status(E) = PLAYING

SetNextCommand : [Engine] × Command → [Engine]
pre SetNextCommand(E, command) **requires** command ∈ {LEFT, RIGHT, UP, DOWN, DIGL, DIGR, NEUTRAL}

CreateEnvironnement : [Engine] × int × int → [Engine]

pre CreateEnvironnement(E, wdt, hgt) **requires** wdt = ScreenWdt(E) **and** hgt = ScreenHgt(E)

CreateEnvironnementWithContract : [Engine] × int × int → [Engine]
pre CreateEnvironnement(E, wdt, hgt) **requires** wdt = ScreenWdt(E) **and** hgt = ScreenHgt(E)

NextLevel : [Engine] → [Engine]

AddAttack : [Engine] × int × int × Direction → [Engine]
pre AddAttack(E, x, y, direction)
requires 0 ≤ x < Environnnement::Wdt(CurrentEnvironnement(E))
and 0 ≤ y < Environnnement::Hgt(CurrentEnvironnement(E))
and ∄ attack in Attack(E) **with**
and GameObject::Wdt(attack) = x
and GameObject::Hgt(attack) = y
and Direction(attack) = direction

Observations :

[invariant] :

forall g in Guards(E)
Screen::CellContent(Character::Wdt(g), Character::Hdt(g)) = g
forall t in Treasures(E)
Screen::CellContent(GameObject::Wdt(t), GameObject::Hdt(t)) = t
forall h in Holes(E)
Screen::CellNature(GameObject::Wdt(h), GameObject::Hdt(h)) = h

[invariants] :

HasAttack(E, wdt, hgt, direction) **min**
forall a in Attacks(E)
GameObject::Wdt(a) = wdt
and GameObject::Hgt(a) = hgt
and Attack::Direction(a) = direction

[Init] :

Math::Min(Map::Keys(Levels(init(wdtScreen, hgtScreen, playerX, playerY))) **where** Map::Get(Levels) = true) = ∞ ⟹
Status(init(wdtScreen, hgtScreen, playerX, playerY)) = WIN
and CurrentLevel(init(wdtScreen, hgtScreen, playerX, playerY)) = ∞
Math::Min(Map::Keys(Levels(init(wdtScreen, hgtScreen, playerX, playerY))) **where** Map::Get(Levels) = true) != ∞ ⟹
CurrentLevel(init(wdtScreen, hgtScreen, playerX, playerY)) = Math::Min(Map::Keys(Levels(init(wdtScreen, hgtScreen, playerX, playerY))) **where** Map::Get(Levels) = true)

CurrentLevel(init(wdtScreen, hgtScreen, playerX, playerY)) != ∞ ⟹
Status(init(wdtScreen, hgtScreen, playerX, playerY)) = PLAYING

CurrentLevel(init(wdtScreen, hgtScreen, playerX, playerY)) != ∞ ⟹
CurrentEnvironnement(init(wdtScreen, hgtScreen, playerX, playerY)) =
Map::Get(Levels(init(wdtScreen, hgtScreen, playerX, playerY)),
CurrentLevel(init(wdtScreen, hgtScreen, playerX, playerY)))

CurrentLevel(init(wdtScreen, hgtScreen, playerX, playerY)) != ∞ ⟹
Board(init(wdtScreen, hgtScreen, playerX, playerY)) != null

CurrentLevel(init(wdtScreen, hgtScreen, playerX, playerY)) != ∞ ⟹
Player(init(wdtScreen, hgtScreen, playerX, playerY)) != null

CurrentLevel(init(wdtScreen, hgtScreen, playerX, playerY)) != ∞ $\Longrightarrow$
CurrentPathResolver(init(wdtScreen, hgtScreen, playerX, playerY)) =
Map::Get(Levels(init(wdtScreen, hgtScreen, playerX, playerY)),
CurrentLevel(init(wdtScreen, hgtScreen, playerX, playerY)))

CurrentLevel(init(wdtScreen, hgtScreen, playerX, playerY)) != ∞ $\Longrightarrow$
forall g in Guards(init(wdtScreen, hgtScreen, playerX, playerY))
Screen::CellNature(CurrentEnvironnement(init(wdtScreen, hgtScreen,
playerX, playerY)), Character::Wdt(g), Character::Hgt(g)) = EMP

CurrentLevel(init(wdtScreen, hgtScreen, playerX, playerY)) != ∞ $\Longrightarrow$
forall t in Treasures(init(wdtScreen, hgtScreen, playerX, playerY))
Screen::CellNature(CurrentEnvironnement(init(wdtScreen, hgtScreen,
playerX, playerY)), GameObject::Wdt(t), GameObject::Hgt(t)) = EMP
and Screen::CellNature(GameObject::Wdt(t), GameObject::Hgt(t) - 1) $\in$
{PLT, MTL}

CurrentLevel(init(wdtScreen, hgtScreen, playerX, playerY)) != ∞ $\Longrightarrow$
forall g1 in Guards(init(wdtScreen, hgtScreen, playerX, playerY))
forall g2 in Guards(init(wdtScreen, hgtScreen, playerX, playerY))
Character::Wdt(g1) != Character::Wdt(g2) **or**
Character::Hgt(g1) != Character::Hgt(g2)

CurrentLevel(init(wdtScreen, hgtScreen, playerX, playerY)) != ∞ $\Longrightarrow$
forall t1 in Treasures(init(wdtScreen, hgtScreen, playerX, playerY))
forall t2 in Treasures(init(wdtScreen, hgtScreen, playerX, playerY))
GameObject::Wdt(t1) != GameObject::Wdt(t2) **or**
GameObject::Hgt(t1) != GameObject::Hgt(t2)

CurrentLevel(init(wdtScreen, hgtScreen, playerX, playerY)) != ∞ $\Longrightarrow$
forall t in Treasures(init(wdtScreen, hgtScreen, playerX, playerY))
(GameObject::Wdt(t) = playerX **and**
GameObject::Hgt(t) = playerY) $\Longrightarrow$
List::Remove(Treasures(init(wdtScreen, hgtScreen, playerX, playerY)),
t)

CurrentLevel(init(wdtScreen, hgtScreen, playerX, playerY)) != ∞ $\Longrightarrow$
forall g in Guards(init(wdtScreen, hgtScreen, playerX, playerY))
Character : :Wdt(g) = playerX **and**
Character : :Hgt(g) = playerY $\Longrightarrow$
Vie(Player(init(wdtScreen, hgtScreen, playerX, playerY))) = 3 - 1

CurrentLevel(init(wdtScreen, hgtScreen, playerX, playerY)) != ∞ $\Longrightarrow$
NbTreasuresTotal(init(wdtScreen, hgtScreen, playerX, playerY))
= Math::Sum(All environnement in Levels(init(wdtScreen,
hgtScreen, playerX, playerY) **where** environnement **is** playable))

CurrentLevel(init(wdtScreen, hgtScreen, playerX, playerY)) != ∞ $\Longrightarrow$
CurrentNbTreasures(init(wdtScreen, hgtScreen, playerX, playerY))
= List::Size(Treasures(init(wdtScreen, hgtScreen, playerX, playerY)))

CurrentLevel(init(wdtScreen, hgtScreen, playerX, playerY)) != ∞ $\Longrightarrow$
Score(init(wdtScreen, hgtScreen, playerX, playerY)) = 0

CurrentLevel(init(wdtScreen, hgtScreen, playerX, playerY)) != ∞ $\Longrightarrow$
NbStep(init(wdtScreen, hgtScreen, playerX, playerY)) = 0

CurrentLevel(init(wdtScreen, hgtScreen, playerX, playerY)) != ∞ $\Longrightarrow$
ScreenWdt(init(wdtScreen, hgtScreen, playerX, playerY)) = wdtScreen

CurrentLevel(init(wdtScreen, hgtScreen, playerX, playerY)) != ∞ $\Longrightarrow$
ScreenHgt(init(wdtScreen, hgtScreen, playerX, playerY)) = hgtScreen

CurrentLevel(init(wdtScreen, hgtScreen, playerX, playerY)) != ∞ $\Longrightarrow$
Empty(Attack(init(wdtScreen, hgtScreen, playerX, playerY))) = true

CurrentLevel(init(wdtScreen, hgtScreen, playerX, playerY)) != ∞ $\Longrightarrow$
Environnement::Wdt(Envi(init(wdtScreen, hgtScreen, playerX, playerY))) = wdtScreen

CurrentLevel(init(wdtScreen, hgtScreen, playerX, playerY)) != ∞ $\Longrightarrow$
Environnement::Hgt(Envi(init(wdtScreen, hgtScreen, playerX, playerY))) = hgtScreen

CurrentLevel(init(wdtScreen, hgtScreen, playerX, playerY)) != ∞ $\Longrightarrow$
Character::Wdt(Player(init(wdtScreen, hgtScreen, playerX, playerY))) = playerX

CurrentLevel(init(wdtScreen, hgtScreen, playerX, playerY)) != ∞ $\Longrightarrow$
Character::Hgt(Player(init(wdtScreen, hgtScreen, playerX, playerY))) = playerY

[InitWithContract] : Math::Min(Map::Keys(Levels(initWithContract(wdtScreen, hgtScreen, playerX, playerY))) **where** Map::Get(Levels) = true) = ∞ $\Longrightarrow$
Status(initWithContract(wdtScreen, hgtScreen, playerX, playerY)) = WIN **and** CurrentLevel(initWithContract(wdtScreen, hgtScreen, playerX, playerY)) = ∞
Math::Min(Map::Keys(Levels(initWithContract(wdtScreen, hgtScreen, playerX, playerY))) **where** Map::Get(Levels) = true) != ∞ $\Longrightarrow$
CurrentLevel(initWithContract(wdtScreen, hgtScreen, playerX, playerY)) = Math::Min(Map::Keys(Levels(initWithContract(wdtScreen, hgtScreen, , playerY))) **where** Map::Get(Levels) = true)

CurrentLevel(initWithContract(wdtScreen, hgtScreen, playerX, playerY)) != ∞ $\Longrightarrow$
Status(initWithContract(wdtScreen, hgtScreen, playerX, playerY)) = PLAYING

CurrentLevel(initWithContract(wdtScreen, hgtScreen, playerX, playerY)) != ∞ $\Longrightarrow$
CurrentEnvironnement(initWithContract(wdtScreen, hgtScreen, playerX, playerY)) =
Map::Get(Levels(initWithContract(wdtScreen, hgtScreen, playerX, playerY)),
CurrentLevel(initWithContract(wdtScreen, hgtScreen, playerX, playerY)))

CurrentLevel(initWithContract(wdtScreen, hgtScreen, playerX, playerY)) != ∞ $\Longrightarrow$
Board(initWithContract(wdtScreen, hgtScreen, playerX, playerY)) != null

CurrentLevel(initWithContract(wdtScreen, hgtScreen, playerX, playerY)) != ∞ $\Longrightarrow$

Player(initWithContract(wdtScreen, hgtScreen, playerX, playerY)) != null

CurrentLevel(initWithContract(wdtScreen, hgtScreen, playerX, playerY)) != ∞
$\Longrightarrow$
CurrentPathResolver(initWithContract(wdtScreen, hgtScreen, playerX, playerY)) =
Map::Get(Levels(initWithContract(wdtScreen, hgtScreen, playerX, playerY)),
CurrentLevel(initWithContract(wdtScreen, hgtScreen, playerX, playerY)))

CurrentLevel(initWithContract(wdtScreen, hgtScreen, playerX, playerY)) != ∞
$\Longrightarrow$
forall g in Guards(initWithContract(wdtScreen, hgtScreen, playerX, playerY))
Screen::CellNature(CurrentEnvironnement(initWithContract(wdtScreen, hgtScreen,playerX, playerY)), Character::Wdt(g), Character::Hgt(g)) = EMP

CurrentLevel(initWithContract(wdtScreen, hgtScreen, playerX, playerY)) != ∞
$\Longrightarrow$
forall t in Treasures(initWithContract(wdtScreen, hgtScreen, playerX, playerY))
Screen::CellNature(CurrentEnvironnement(initWithContract(wdtScreen, hgtScreen,
playerX, playerY)), GameObject::Wdt(t), GameObject::Hgt(t)) = EMP
and Screen::CellNature(GameObject::Wdt(t), GameObject::Hgt(t) - 1) $\in$ {PLT, MTL}

CurrentLevel(initWithContract(wdtScreen, hgtScreen, playerX, playerY)) != ∞
$\Longrightarrow$
forall g1 in Guards(initWithContract(wdtScreen, hgtScreen, playerX, playerY))
forall g2 in Guards(initWithContract(wdtScreen, hgtScreen, playerX, playerY))
Character::Wdt(g1) != Character::Wdt(g2) **or**
Character::Hgt(g1) != Character::Hgt(g2)

CurrentLevel(initWithContract(wdtScreen, hgtScreen, playerX, playerY)) != ∞
$\Longrightarrow$
forall t1 in Treasures(initWithContract(wdtScreen, hgtScreen, playerX, playerY))
forall t2 in Treasures(initWithContract(wdtScreen, hgtScreen, playerX, playerY))
GameObject::Wdt(t1) != GameObject::Wdt(t2) **or**
GameObject::Hgt(t1) != GameObject::Hgt(t2)

CurrentLevel(initWithContract(wdtScreen, hgtScreen, playerX, playerY)) != ∞
$\Longrightarrow$
forall t in Treasures(initWithContract(wdtScreen, hgtScreen, playerX, playerY))
(GameObject::Wdt(t) = playerX **and**
GameObject::Hgt(t) = playerY) $\Longrightarrow$
List::Remove(Treasures(initWithContract(wdtScreen, hgtScreen, playerX, playerY)),
t)

CurrentLevel(initWithContract(wdtScreen, hgtScreen, playerX, playerY)) != ∞
$\Longrightarrow$
forall g in Guards(initWithContract(wdtScreen, hgtScreen, playerX, playerY))
Character::Wdt(g) = playerX **and**
Character::Hgt(g) = playerY $\Longrightarrow$
Vie(Player(initWithContract(wdtScreen, hgtScreen, playerX, playerY))) = 3 - 1

CurrentLevel(initWithContract(wdtScreen, hgtScreen, playerX, playerY)) != ∞
$\Longrightarrow$
NbTreasuresTotal(initWithContract(wdtScreen, hgtScreen, playerX, playerY))
= Math::Sum(All environnement in Levels(initWithContract(wdtScreen, hgtScreen, playerX, playerY) **where** environnement **is** playable))

CurrentLevel(initWithContract(wdtScreen, hgtScreen, playerX, playerY)) != ∞
$\Longrightarrow$
CurrentNbTreasures(initWithContract(wdtScreen, hgtScreen, playerX, playerY)) = List::Size(Treasures(initWithContract(wdtScreen, hgtScreen, playerX, playerY)))

CurrentLevel(initWithContract(wdtScreen, hgtScreen, playerX, playerY)) != ∞
$\Longrightarrow$
Score(initWithContract(wdtScreen, hgtScreen, playerX, playerY)) = 0

CurrentLevel(initWithContract(wdtScreen, hgtScreen, playerX, playerY)) != ∞
$\Longrightarrow$
NbStep(initWithContract(wdtScreen, hgtScreen, playerX, playerY)) = 0

CurrentLevel(initWithContract(wdtScreen, hgtScreen, playerX, playerY)) != ∞
$\Longrightarrow$
ScreenWdt(initWithContract(wdtScreen, hgtScreen, playerX, playerY)) = wdtScreen

CurrentLevel(initWithContract(wdtScreen, hgtScreen, playerX, playerY)) != ∞
$\Longrightarrow$
ScreenHgt(initWithContract(wdtScreen, hgtScreen, playerX, playerY)) = hgtScreen

CurrentLevel(initWithContract(wdtScreen, hgtScreen, playerX, playerY)) != ∞
$\Longrightarrow$
Empty(Attack(initWithContract(wdtScreen, hgtScreen, playerX, playerY))) = true

CurrentLevel(initWithContract(wdtScreen, hgtScreen, playerX, playerY)) != ∞
$\Longrightarrow$
Environnement::Wdt(Envi(initWithContract(wdtScreen, hgtScreen, playerX, playerY))) = wdtScreen

CurrentLevel(initWithContract(wdtScreen, hgtScreen, playerX, playerY)) != ∞
$\Longrightarrow$
Environnement::Hgt(Envi(initWithContract(wdtScreen, hgtScreen, playerX, playerY))) = hgtScreen

CurrentLevel(initWithContract(wdtScreen, hgtScreen, playerX, playerY)) != ∞
$\Longrightarrow$
Character::Wdt(Player(initWithContract(wdtScreen, hgtScreen, playerX, playerY))) = playerX

CurrentLevel(initWithContract(wdtScreen, hgtScreen, playerX, playerY)) != ∞
$\Longrightarrow$
Character::Hgt(Player(initWithContract(wdtScreen, hgtScreen, playerX, playerY))) = playerY

CurrentLevel(initWithContract(wdtScreen, hgtScreen, playerX, playerY)) != ∞ $\Longrightarrow$ Class(CurrentEnvironnement(initWithContract(wdtScreen, hgtScreen, playerX, playerY))) = Class(EnvironnementContract)

CurrentLevel(initWithContract(wdtScreen, hgtScreen, playerX, playerY)) != ∞ $\Longrightarrow$ Class(CurrentPathResolver(initWithContract(wdtScreen, hgtScreen, playerX, playerY))) = Class(PathResolverContract)

CurrentLevel(initWithContract(wdtScreen, hgtScreen, playerX, playerY)) != ∞ $\Longrightarrow$ Class(Player(initWithContract(wdtScreen, hgtScreen, playerX, playerY)))
= Class(PlayerContract)

CurrentLevel(initWithContract(wdtScreen, hgtScreen, playerX, playerY)) != ∞ $\Longrightarrow$ **forall** g in Guards(initWithContract(wdtScreen, hgtScreen, playerX,playerY))
Class(g) = Class(GuardContract)

CurrentLevel(initWithContract(wdtScreen, hgtScreen, playerX, playerY)) != ∞ $\Longrightarrow$ Class(Board(initWithContract(wdtScreen, hgtScreen, playerX, playerY)))
= Class(BoardContract)

[InitForTest] :

Math::Min(Map::Keys(Levels(initForTest(wdtScreen, hgtScreen, playerX, playerY))) **where** Map::Get(Levels) = true) = ∞ $\Longrightarrow$
Status(initForTest(wdtScreen, hgtScreen, playerX, playerY)) = WIN
and CurrentLevel(initForTest(wdtScreen, hgtScreen, playerX, playerY)) = ∞
Math::Min(Map::Keys(Levels(initForTest(wdtScreen, hgtScreen, playerX, playerY))) **where** Map::Get(Levels) = true) != ∞ $\Longrightarrow$
CurrentLevel(initForTest(wdtScreen, hgtScreen, playerX, playerY)) = Math::Min(Map::Keys(Levels(initForTest(wdtScreen, hgtScreen, , playerY))) **where** Map::Get(Levels) = true)

CurrentLevel(initForTest(wdtScreen, hgtScreen, playerX, playerY)) != ∞ $\Longrightarrow$
Status(initForTest(wdtScreen, hgtScreen, playerX, playerY)) = PLAYING

CurrentLevel(initForTest(wdtScreen, hgtScreen, playerX, playerY)) != ∞ $\Longrightarrow$
CurrentEnvironnement(initForTest(wdtScreen, hgtScreen, playerX, playerY))
= Map::Get(Levels(initForTest(wdtScreen, hgtScreen, playerX, playerY)),
CurrentLevel(initForTest(wdtScreen, hgtScreen, playerX, playerY)))

CurrentLevel(initForTest(wdtScreen, hgtScreen, playerX, playerY)) != ∞ $\Longrightarrow$
Board(initForTest(wdtScreen, hgtScreen, playerX, playerY)) = null

CurrentLevel(initForTest(wdtScreen, hgtScreen, playerX, playerY)) != ∞ $\Longrightarrow$
Player(initForTest(wdtScreen, hgtScreen, playerX, playerY)) != null

CurrentLevel(initForTest(wdtScreen, hgtScreen, playerX, playerY)) != ∞ $\Longrightarrow$
CurrentPathResolver(initForTest(wdtScreen, hgtScreen, playerX, playerY))
= Map::Get(Levels(initForTest(wdtScreen, hgtScreen, playerX, playerY)),
CurrentLevel(initForTest(wdtScreen, hgtScreen, playerX, playerY)))

CurrentLevel(initForTest(wdtScreen, hgtScreen, playerX, playerY)) != ∞ $\Longrightarrow$
forall g in Guards(initForTest(wdtScreen, hgtScreen, playerX, playerY))
Screen::CellNature(CurrentEnvironnement(initForTest(wdtScreen,
hgtScreen,playerX, playerY)), Character::Wdt(g), Character::Hgt(g))
= EMP

CurrentLevel(initForTest(wdtScreen, hgtScreen, playerX, playerY)) != ∞ $\Longrightarrow$
forall t in Treasures(initForTest(wdtScreen, hgtScreen, playerX, playerY))
Screen::CellNature(CurrentEnvironnement(initForTest(wdtScreen,
hgtScreen, playerX, playerY)), GameObject::Wdt(t),
GameObject::Hgt(t)) = EMP
and Screen::CellNature(GameObject::Wdt(t), GameObject::Hgt(t) - 1) $\in$
{PLT, MTL}

CurrentLevel(initForTest(wdtScreen, hgtScreen, playerX, playerY)) != ∞ $\Longrightarrow$
forall g1 in Guards(initForTest(wdtScreen, hgtScreen, playerX, playerY))
forall g2 in Guards(initForTest(wdtScreen, hgtScreen, playerX, playerY))
Character::Wdt(g1) != Character::Wdt(g2) **or**
Character::Hgt(g1) != Character::Hgt(g2)

CurrentLevel(initWithContract(wdtScreen, hgtScreen, playerX, playerY)) != ∞
$\Longrightarrow$
forall t1 in Treasures(initForTest(wdtScreen, hgtScreen, playerX, playerY))
forall t2 in Treasures(initForTest(wdtScreen, hgtScreen, playerX,
playerY))
GameObject::Wdt(t1) != GameObject::Wdt(t2) **or**
GameObject::Hgt(t1) != GameObject::Hgt(t2)

CurrentLevel(initForTest(wdtScreen, hgtScreen, playerX, playerY)) != ∞ $\Longrightarrow$
forall t in Treasures(initForTest(wdtScreen, hgtScreen, playerX, playerY))
(GameObject::Wdt(t) = playerX **and**
GameObject::Hgt(t) = playerY) $\Longrightarrow$
List::Remove(Treasures(initForTest(wdtScreen, hgtScreen, playerX,
playerY)), t)

CurrentLevel(initForTest(wdtScreen, hgtScreen, playerX, playerY)) != ∞ $\Longrightarrow$
forall g in Guards(initForTest(wdtScreen, hgtScreen, playerX, playerY))
Character::Wdt(g) = playerX **and**
Character::Hgt(g) = playerY $\Longrightarrow$
Vie(Player(initForTest(wdtScreen, hgtScreen, playerX, playerY)))
= 3 - 1

CurrentLevel(initForTest(wdtScreen, hgtScreen, playerX, playerY)) != ∞ $\Longrightarrow$
NbTreasuresTotal(initForTest(wdtScreen, hgtScreen, playerX, playerY))
= Math::Sum(All environnement in Levels(initForTest(wdtScreen,
hgtScreen, playerX, playerY) **where** environnement **is** playable))

CurrentLevel(initForTest(wdtScreen, hgtScreen, playerX, playerY)) != ∞ $\Longrightarrow$
CurrentNbTreasures(initForTest(wdtScreen, hgtScreen, playerX,
playerY)) = List::Size(Treasures(initForTest(wdtScreen, hgtScreen, playerX,
playerY)))

CurrentLevel(initForTest(wdtScreen, hgtScreen, playerX, playerY)) != ∞ $\Longrightarrow$
Score(initForTest(wdtScreen, hgtScreen, playerX, playerY)) = 0

CurrentLevel(initForTest(wdtScreen, hgtScreen, playerX, playerY)) != ∞ $\Longrightarrow$
NbStep(initForTest(wdtScreen, hgtScreen, playerX, playerY)) = 0

CurrentLevel(initForTest(wdtScreen, hgtScreen, playerX, playerY)) != ∞ $\Longrightarrow$
ScreenWdt(initForTest(wdtScreen, hgtScreen, playerX, playerY)) = wdtScreen

CurrentLevel(initForTest(wdtScreen, hgtScreen, playerX, playerY)) != ∞ $\Longrightarrow$
ScreenHgt(initForTest(wdtScreen, hgtScreen, playerX, playerY)) = hgtScreen

CurrentLevel(initForTest(wdtScreen, hgtScreen, playerX, playerY)) != ∞ $\Longrightarrow$
Empty(Attack(initForTest(wdtScreen, hgtScreen, playerX, playerY))) = true

CurrentLevel(initForTest(wdtScreen, hgtScreen, playerX, playerY)) != ∞ $\Longrightarrow$
Environnement::Wdt(Envi(initForTest(wdtScreen, hgtScreen, playerX, playerY))) = wdtScreen

CurrentLevel(initForTest(wdtScreen, hgtScreen, playerX, playerY)) != ∞ $\Longrightarrow$
Environnement::Hgt(Envi(initForTest(wdtScreen, hgtScreen, playerX, playerY))) = hgtScreen

CurrentLevel(initForTest(wdtScreen, hgtScreen, playerX, playerY)) != ∞ $\Longrightarrow$
Character::Wdt(Player(initForTest(wdtScreen, hgtScreen, playerX, playerY))) = playerX

CurrentLevel(initForTest(wdtScreen, hgtScreen, playerX, playerY)) != ∞ $\Longrightarrow$
Character::Hgt(Player(initForTest(wdtScreen, hgtScreen, playerX, playerY))) = playerY

CurrentLevel(initForTest(wdtScreen, hgtScreen, playerX, playerY)) != ∞ $\Longrightarrow$
Class(CurrentEnvironnement(initForTest(wdtScreen, hgtScreen, playerX, playerY))) = Class(EnvironnementContract)

CurrentLevel(initForTest(wdtScreen, hgtScreen, playerX, playerY)) != ∞ $\Longrightarrow$
Class(CurrentPathResolver(initForTest(wdtScreen, hgtScreen, playerX, playerY))) = Class(PathResolverContract)

CurrentLevel(initForTest(wdtScreen, hgtScreen, playerX, playerY)) != ∞ $\Longrightarrow$
Class(Player(initForTest(wdtScreen, hgtScreen, playerX, playerY)))
= Class(PlayerContract)

CurrentLevel(initForTest(wdtScreen, hgtScreen, playerX, playerY)) != ∞ $\Longrightarrow$
forall g in Guards(initForTest(wdtScreen, hgtScreen,
playerX,playerY))
Class(g) = Class(GuardContract)

CurrentLevel(initForTest(wdtScreen, hgtScreen, playerX, playerY)) != ∞ $\Longrightarrow$
Class(Board(initForTest(wdtScreen, hgtScreen, playerX, playerY)))
= Class(BoardContract)

[AddHole] : **forall** <h, t> in KeySet(Holes(E, x, y))
GameObject::Wdt(h) = x **and** GameObject::Hgt(h) = y $\Longrightarrow$ t = 0

forall <h, t> in KeySet(Holes(E, x, y))
GameObject::Wdt(h) != x **and** GameObject::Hgt(h) != y
$\implies$ t **of** Holes(AddHole(E, x, y)) = t of Holes(E)

Score(AddHole(E, x, y)) = Score(E)

CurrentNbTreasures(AddHole(E, x, y)) = CurrentNbTreasures(E)

CurrentLevel(AddHole(E, x, y)) = CurrentLevel(E)

CurrentEnvironnement(AddHole(E, x, y)) = CurrentEnvironnement(E)

CurrentPathResolver(AddHole(E, x, y)) = CurrentPathResolver(E)

Attack(AddHole(E, x, y)) = Attack(E)

NextCommand(AddHole(E, x, y)) = NextCommand(E)

Status(AddHole(E, x, y)) = Status(E)

NbStep(AddHole(E, x, y)) = NbStep(E)

Guards(AddHole(E, x, y)) = Guards(E)

Treasures(AddHole(E, x, y)) = Treasures(E)

[Step] :

forall <hole, I> in Holes(E)
I = 15 $\implies$
Screen::CellNature(step(E), GameObject::Wdt(hole),
GameObject::Hgt(hole)) = PLT
(Character::Wdt(Player(E)) = GameObject::Wdt(hole)
and Character::Hgt(Player(E)) = GameObject::Hgt(hole)) $\implies$
Character::Wdt(Player(E)) = Character::InitialWdt(Player(E))
and Character::Hgt(Player(E)) = Character::InitialHgt(Player(E))
and Vie(Player(Step(E)) = Vie(Player(E)) - 1
forall g in Guards(Step(E))
Character::Wdt(g) = GameObject::Wdt(hole)
and Character::Hgt(g) = GameObject::Hgt(hole) $\implies$
Character::Wdt(g) = Character::InitialWdt(g)
and Character::Hgt(g) = Character::initialHgt(g)
I < 15 $\implies$ I(Step(E)) = I(E) + 1

forall currentTrs in Treasures(E)
GameObject::Wdt(currentTrs) = Character::Wdt(Player(E))
and GameObject::Hgt(currentTrs) = Character::Hdt(Player(E))) $\implies$
Score(step(E)) = Score(E) + 1
forall t2 in Treasures(step(E))
(GameObject::Wgt(t2) = GameObject::Wgt(currentTrs) **and**
GameObject::Hgt(t2) = GameObject::Hgt(currentTrs)) = false

nextCommand(step(E)) = NEUTRAL

NbStep(Step(E)) = NbStep(E) + 1

forall g in Guards(Step(E))

Character::Wdt(g) = Character::Wdt(Player(Step(E))) **and**
Character::Hgt(g) = Character::Hgt(Player(Step(E))) $\implies$
Vie(Player(Step(E)) = Vie(Player(E)) - 1

CurrentLevel(Step(E)) < 3
and Score(Step(E)) = CurrentNbTreasures(E)
and Character::Wdt(Player(Step(E))) =
Environnement::WdtPortal(CurrentEnvironnement(Step(E)))
and Character::Hgt(Player(Step(E))) =
Environnement::HgtPortal(CurrentEnvironnement(Step(E))) $\implies$
E = NextLevel(E)

Vie(Player(Step(E))) = 0 $\implies$ Status(Step(E)) = LOSS

forall a in Attacks(E)
Attack::Direction(a) = LEFT $\implies$
GameObject::Wdt(a) - 1 < 0 **or**
Screen::CellNature(CurrentEnvironnement(E), GameObject::Wdt(a) - 1,
GameObject::Hgt(a)) ∈ {PLT, MTL} $\implies$
List::Contains(Attacks(Step(E), a)) = false
Screen::CellNature(CurrentEnvironnement(E), GameObject::Wdt(a) - 1,
GameObject::Hgt(a)) = EMP $\implies$
GameObject : :Wdt(Attack::Step(a)) = GameObject::Wdt(a) - 1
Attack::Direction(a) = RIGHT $\implies$
GameObject::Wdt(a) + 1 >=
Screen : :Width(CurrentEnvironnement(E)) **or**
Screen::CellNature(CurrentEnvironnement(E),
GameObject::Wdt(a) + 1, GameObject::Hgt(a)) ∈ {PLT, MTL} $\implies$
List::Contains(Attacks(Step(E), a)) = false
Screen::CellNature(CurrentEnvironnement(E), GameObject::Wdt(a) + 1,
GameObject::Hgt(a)) = EMP $\implies$
GameObject::Wdt(Attack::Step(a)) = GameObject::Wdt(a) + 1

forall a in Attacks(Step(E))
forall g in Guards(Step(E))
Attack::Wdt(a) = Character::Wdt(g)
and Attack::Hgt(a) = Character::Hgt(g) $\implies$
Character::Wdt(g) = Character::InitialWdt(g) **and**
Character::Hgt(g) = Character::initialHgt(g) **and**
List::Contains(Attacks(Step(E), a)) = false

[SetNextCommand] : NextCommand(SetNextCommand(E, command)) = command

Score(SetNextCommand(E, command)) = Score(E)

NbStep(SetNextCommand(E, command)) = NbStep(E)

Guards(SetNextCommand(E, command)) = Guards(E)

Treasures(SetNextCommand(E, command)) = Treasures(E)

CurrentEnvironnement(SetNextCommand(E, command))
= CurrentEnvironnement(E)

CurrentPathResolver(SetNextCommand(E, command))
= CurrentPathResolver(E)

Status(SetNextCommand(E, command)) = Status(E)

Holes(SetNextCommand(E, command)) = Holes(E)

CurrentNbTreasures(SetNextCommand(E, command))
= CurrentNbTreasures(E)

CurrentLevel(SetNextCommand(E, command)) = CurrentLevel(E)

Attack(SetNextCommand(E, command)) = Attack(E)

[CreateEnvironnement] :

for nbNiveau in [0..3]
PathResolver::PlayerCanReachAllTreasuresAndPortal(
Map::Get(PathResolvers(CreateEnvironnements(E, wdt, hgt)), nbNiveau),
Treasures(Map::Get(Levels(CreateEnvironnements(E, wdt, hgt),
nbNiveau))),
WdtPortal(Map::Get(Levels(CreateEnvironnements(E, wdt, hgt),
nbNiveau))),
HgtPortal(Map::Get(Levels(CreateEnvironnements(E, wdt, hgt),
nbNiveau))))
and EditableScreen::Playable(Map::Get(Editors(CreateEnvironnements(E,
wdt, hgt)), nbNiveau)) then
$\Longrightarrow$ Map::Get(PlayableLevels(CreateEnvironnements(E, wdt, hgt)),
nbNiveau) = true

!(PathResolver::PlayerCanReachAllTreasuresAndPortal(
Map::Get(PathResolvers(CreateEnvironnements(E, wdt, hgt)), nbNiveau),
Treasures(Map::Get(Levels(CreateEnvironnements(E, wdt, hgt),
nbNiveau))),
WdtPortal(Map::Get(Levels(CreateEnvironnements(E, wdt, hgt),
nbNiveau))),
HgtPortal(Map::Get(Levels(CreateEnvironnements(E, wdt, hgt),
nbNiveau))))
and EditableScreen::Playable(Map::Get(Editors(CreateEnvironnements(E,
wdt, hgt)), nbNiveau))) then
$\Longrightarrow$ Map::Get(PlayableLevels(CreateEnvironnements(E, wdt, hgt)),
nbNiveau) = false

for line [List::Length(File::Lines(F))..1]
for cell [0..List::Length(List::Get(File::Lines(F), line))]
Content(File, cell, line) = "EMP" $\Longrightarrow$
Environnement::CellNature(Map::Get(
Levels(CreateEnvironnements(E, wdt, hgt)), nbNiveau), cell, line)
= "EMP"
and EditableScreen::CellNature(Map::Get(
Editors(CreateEnvironnements(E, wdt, hgt)), nbNiveau), cell, line)
= "EMP"
Content(File, cell, line) = "HOL" $\Longrightarrow$
Environnement::CellNature(Map::Get(
Levels(CreateEnvironnements(E, wdt, hgt)), nbNiveau), cell, line)
= "HOL"
and EditableScreen::CellNature(Map::Get(
Editors(CreateEnvironnements(E, wdt, hgt)), nbNiveau), cell, line)
= "HOL"

Content(File, cell, line) = "PLT" ⟹
Environnement::CellNature(Map::Get(
Levels(CreateEnvironnements(E, wdt, hgt)), nbNiveau), cell, line)
= "PLT"
and EditableScreen::CellNature(Map::Get(
Editors(CreateEnvironnements(E, wdt, hgt)), nbNiveau), cell, line)
= "PLT"
Content(File, cell, line) = "MTL" ⟹
Environnement::CellNature(Map::Get(
Levels(CreateEnvironnements(E, wdt, hgt)), nbNiveau), cell, line)
= "MTL"
and EditableScreen::CellNature(Map::Get(
Editors(CreateEnvironnements(E, wdt, hgt)), nbNiveau), cell, line)
= "MTL"
Content(File, cell, line) = "LAD" ⟹
Environnement::CellNature(Map::Get(
Levels(CreateEnvironnements(E, wdt, hgt)), nbNiveau), cell, line)
= "LAD"
and EditableScreen::CellNature(Map::Get(
Editors(CreateEnvironnements(E, wdt, hgt)), nbNiveau), cell, line)
= "LAD"
Content(File, cell, line) = "HDR" ⟹
Environnement::CellNature(Map::Get(
Levels(CreateEnvironnements(E, wdt, hgt)), nbNiveau), cell, line)
= "HDR"
and EditableScreen::CellNature(Map::Get(
Editors(CreateEnvironnements(E, wdt, hgt)), nbNiveau), cell, line)
= "HDR"
Content(File, cell, line) = "GRD" ⟹
Environnement::CellNature(Map::Get(
Levels(CreateEnvironnements(E, wdt, hgt)), nbNiveau), cell, line)
= "EMP"
and EditableScreen::CellNature(Map::Get(
Editors(CreateEnvironnements(E, wdt, hgt)), nbNiveau), cell, line)
= "EMP"
and ∃ Character in Environnement::CellContent(Map::Get(
Levels(CreateEnvironnements(E, wdt, hgt)), nbNiveau), cell, line)
and ∃ Character in Guards(Map::Get(
Levels(CreateEnvironnements(E, wdt, hgt)), nbNiveau))
Content(File, cell, line) = "TRS" ⟹
Environnement::CellNature(Map::Get(
Levels(CreateEnvironnements(E, wdt, hgt)), nbNiveau), cell, line)
= "EMP"
and EditableScreen::CellNature(Map::Get(
Editors(CreateEnvironnements(E, wdt, hgt)), nbNiveau), cell, line)
= "EMP"
and ∃ Treasures in Environnement::CellContent(Map::Get(
Levels(CreateEnvironnements(E, wdt, hgt)), nbNiveau), cell, line)
and ∃ Treasures in Treasures(Map::Get(Levels(
CreateEnvironnements(E, wdt, hgt)), nbNiveau))
Content(File, cell, line) = "PRT" ⟹
Environnement::CellNature(Map::Get(
Levels(CreateEnvironnements(E, wdt, hgt)), nbNiveau), cell, line)
= "PRT"
and EditableScreen::CellNature(Map::Get(
Editors(CreateEnvironnements(E, wdt, hgt)), nbNiveau), cell, line)

= "PRT"
and Environnement::WdtPortail(Map::Get(
Levels(CreateEnvironnements(E, wdt, hgt)), nbNiveau)) = cell
and Environnement::HgtPortail(Map::Get(
Levels(CreateEnvironnements(E, wdt, hgt)), nbNiveau)) = line

Environnement::Wdt(CurrentEnvironnement(CreateEnvironnements(E, wdt,
hgt))) = wdt

Environnement::Hgt(CurrentEnvironnement(CreateEnvironnements(E, wdt,
hgt))) = hgt

[CreateEnvironnementWithContract] :
for nbNiveau in [0..3]
PathResolver::PlayerCanReachAllTreasuresAndPortal(
Map::Get(PathResolvers(CreateEnvironnementsWithContract(E, wdt,
hgt)), nbNiveau),
Treasures(Map::Get(Levels(CreateEnvironnementsWithContract(E, wdt,
hgt), nbNiveau))),
WdtPortal(Map::Get(Levels(CreateEnvironnementsWithContract(E, wdt,
hgt), nbNiveau))),
HgtPortal(Map::Get(Levels(CreateEnvironnementsWithContract(E, wdt,
hgt), nbNiveau))))
and EditableScreen::Playable(Map::Get(Editors(CreateEnvironnements(E,
wdt, hgt)), nbNiveau)) then
$\Longrightarrow$ Map::Get(PlayableLevels(CreateEnvironnementsWithContract(E, wdt,
hgt)),
nbNiveau) = true

!(PathResolver::PlayerCanReachAllTreasuresAndPortal(
Map::Get(PathResolvers(CreateEnvironnementsWithContract(E, wdt,
hgt)), nbNiveau),
Treasures(Map::Get(Levels(CreateEnvironnementsWithContract(E, wdt,
hgt), nbNiveau))),
WdtPortal(Map::Get(Levels(CreateEnvironnementsWithContract(E, wdt,
hgt), nbNiveau))),
HgtPortal(Map::Get(Levels(CreateEnvironnementsWithContract(E, wdt,
hgt), nbNiveau))))
and EditableScreen::Playable(Map::Get(Editors(CreateEnvironnements(E,
wdt, hgt)), nbNiveau))) then
$\Longrightarrow$ Map::Get(PlayableLevels(CreateEnvironnementsWithContract(E, wdt,
hgt)),
nbNiveau) = false

for line [List::Length(File::Lines(F))..1]
for cell [0..List::Length(List::Get(File::Lines(F), line))]
Class(Map::Get(Levels(Environnement(
CreateEnvironnementsWithContract(E, wdt, hgt)), nbNiveau)))
= Class(EnvironnementContract)
Class(Map::Get(Editors(Environnement(
CreateEnvironnementsWithContract(E, wdt, hgt)), nbNiveau)))
= Class(EditableScreenContract)
Class(Map::Get(PathResolvers(Environnement(
CreateEnvironnementsWithContract(E, wdt, hgt)), nbNiveau)))
= Class(PathResolverContract)
Content(File, cell, line) = "EMP" $\Longrightarrow$

```
                Environnement::CellNature(Map::Get(
                Levels(CreateEnvironnementsWithContract(E, wdt, hgt)),
nbNiveau), cell, line)
                = "EMP"
                and EditableScreen::CellNature(Map::Get(
                Editors(CreateEnvironnementsWithContract(E, wdt, hgt)),
nbNiveau), cell, line)
                = "EMP"
            Content(File, cell, line) = "HOL" ⟹
                Environnement::CellNature(Map::Get(
                Levels(CreateEnvironnementsWithContract(E, wdt, hgt)),
nbNiveau), cell, line)
                = "HOL"
                and EditableScreen::CellNature(Map::Get(
                Editors(CreateEnvironnementsWithContract(E, wdt, hgt)),
nbNiveau), cell, line)
                = "HOL"
            Content(File, cell, line) = "PLT" ⟹
                Environnement::CellNature(Map::Get(
                Levels(CreateEnvironnementsWithContract(E, wdt, hgt)),
nbNiveau), cell, line)
                = "PLT"
                and EditableScreen::CellNature(Map::Get(
                Editors(CreateEnvironnementsWithContract(E, wdt, hgt)),
nbNiveau), cell, line)
                = "PLT"
            Content(File, cell, line) = "MTL" ⟹
                Environnement::CellNature(Map::Get(
                Levels(CreateEnvironnementsWithContract(E, wdt, hgt)),
nbNiveau), cell, line)
                = "MTL"
                and EditableScreen::CellNature(Map::Get(
                Editors(CreateEnvironnementsWithContract(E, wdt, hgt)),
nbNiveau), cell, line)
                = "MTL"
            Content(File, cell, line) = "LAD" ⟹
                Environnement::CellNature(Map::Get(
                Levels(CreateEnvironnementsWithContract(E, wdt, hgt)),
nbNiveau), cell, line)
                = "LAD"
                and EditableScreen::CellNature(Map::Get(
                Editors(CreateEnvironnementsWithContract(E, wdt, hgt)),
nbNiveau), cell, line)
                = "LAD"
            Content(File, cell, line) = "HDR" ⟹
                Environnement::CellNature(Map::Get(
                Levels(CreateEnvironnementsWithContract(E, wdt, hgt)),
nbNiveau), cell, line)
                = "HDR"
                and EditableScreen::CellNature(Map::Get(
                Editors(CreateEnvironnementsWithContract(E, wdt, hgt)),
nbNiveau), cell, line)
                = "HDR"
            Content(File, cell, line) = "GRD" ⟹
                Environnement::CellNature(Map::Get(
```

Levels(CreateEnvironnementsWithContract(E, wdt, hgt)),
nbNiveau), cell, line)
= "EMP"
and EditableScreen::CellNature(Map::Get(
Editors(CreateEnvironnementsWithContract(E, wdt, hgt)),
nbNiveau), cell, line)
= "EMP"
and ∃ Character in Environnement : :CellContent(Map::Get(
Levels(CreateEnvironnementsWithContract(E, wdt, hgt)),
nbNiveau), cell, line)
and ∃ Character in Guards(Map::Get(
Levels(CreateEnvironnementsWithContract(E, wdt, hgt)),
nbNiveau))
and Class(g) = Class(GuardContract)
Content(File, cell, line) = "TRS" ⟹
Environnement::CellNature(Map::Get(
Levels(CreateEnvironnementsWithContract(E, wdt, hgt)),
nbNiveau), cell, line)
= "EMP"
and EditableScreen::CellNature(Map::Get(
Editors(CreateEnvironnementsWithContract(E, wdt, hgt)),
nbNiveau), cell, line)
= "EMP"
and ∃ Treasures in Environnement::CellContent(Map::Get(
Levels(CreateEnvironnementsWithContract(E, wdt, hgt)),
nbNiveau), cell, line)
and ∃ Treasures in Treasures(Map::Get(Levels(
CreateEnvironnementsWithContract(E, wdt, hgt)), nbNiveau))
and Class(t) = Class(ItemContract)
Content(File, cell, line) = "PRT" ⟹
Environnement::CellNature(Map::Get(
Levels(CreateEnvironnementsWithContract(E, wdt, hgt)),
nbNiveau), cell, line)
= "PRT"
and EditableScreen::CellNature(Map::Get(
Editors(CreateEnvironnementsWithContract(E, wdt, hgt)),
nbNiveau), cell, line)
= "PRT"
and Environnement::WdtPortail(Map::Get(
Levels(CreateEnvironnementsWithContract(E, wdt, hgt)),
nbNiveau)) = cell
and Environnement::HgtPortail(Map::Get(
Levels(CreateEnvironnementsWithContract(E, wdt, hgt)),
nbNiveau)) = line

Environnement::Wdt(CurrentEnvironnement(CreateEnvironnementsWithContract(E,
wdt, hgt))) = wdt

Environnement::Hgt(CurrentEnvironnement(CreateEnvironnementsWithContract(E,
wdt, hgt))) = hgt

[NextLevel] : CurrentLevel(NextLevel(E)) = Math::Min(Map::Keys(Levels(NextLevel(E)))
where (Map::Get(PlayableLevels(NextLevel(E)), key) = true)
and and (key > CurrentLevel(E))

CurrentLevel(NextLevel(E)) = ∞ ⟹

Status(NextLevel(E)) = WIN

CurrentLevel(NextLevel(E)) = ∞ $\Longrightarrow$
CurrentEnvironnement(nextLevel(E)) = Map::Get(Levels(E),
CurrentLevel(NextLevel(E)))

CurrentLevel(NextLevel(E)) = ∞ $\Longrightarrow$
CurrentPathResolver(nextLevel(E)) = Map::Get(PathResolvers(E),
CurrentLevel(NextLevel(E)))

CurrentLevel(NextLevel(E)) = ∞ $\Longrightarrow$
Holes(nextLevel(E)) = Empty

CurrentLevel(NextLevel(E)) = ∞ $\Longrightarrow$
Environnement(Board(nextLevel(E))) = Environnement(nextLevel(E))

CurrentLevel(NextLevel(E)) = ∞ $\Longrightarrow$
Treasures(nextLevel(E)) = Treasures(Environnement(nextLevel(E)))

CurrentLevel(NextLevel(E)) = ∞ $\Longrightarrow$
Guards(nextLevel(E)) = Guards(Environnement(nextLevel(E)))

CurrentLevel(NextLevel(E)) = ∞ $\Longrightarrow$
CurrentNbTreasures(nextLevel(E))
= List::Size(Treasures(nextLevel(E))) + CurrentNbTreasures(E)

CurrentLevel(NextLevel(E)) = ∞ $\Longrightarrow$
Character : :Wdt(Player(NextLevel(E))) =
Environnement : :WdtPortal(CurrentEnvironnement(nextLevel(E)))
and Character : :Hgt(Player(NextLevel(E))) =
Environnement : :HgtPortal(CurrentEnvironnement(nextLevel(E)))

Score(nextLevel(E)) = Score(E)

NbStep(nextLevel(E)) = NbStep(E)

CurrentLevel(NextLevel(E)) = ∞ $\Longrightarrow$
Attack(nextLevel(E)) = Empty

CurrentLevel(NextLevel(E)) = ∞ $\Longrightarrow$
Player::Vie(Player(NextLevel(E))) = 3

CurrentLevel(NextLevel(E)) = ∞ $\Longrightarrow$
Player::NbAttacks(Player(NextLevel(E))) = 3

[AddAttack] : NextCommand(AddAttack(E, x, y, direction)) = command

Score(AddAttack(E, x, y, direction)) = Score(E)

NbStep(AddAttack(E, x, y, direction)) = NbStep(E)

Treasures(AddAttack(E, x, y, direction)) = Treasures(E)

Guards(AddAttack(E, x, y, direction)) = Guards(E)

CurrentEnvironnement(AddAttack(E, x, y, direction))

= CurrentEnvironnement(E)

CurrentPathResolver(AddAttack(E, x, y, direction)) = CurrentPathResolver(E)

Status(AddAttack(E, x, y, direction)) = Status(E)

Holes(AddAttack(E, x, y, direction)) = Holes(E)

CurrentNbTreasures(AddAttack(E, x, y, direction))
= CurrentNbTreasures(E)

CurrentLevel(AddAttack(E, x, y, direction)) = CurrentLevel(E)

NbAttacks(Player(E)) > 0 $\Longrightarrow$
∃ attack in Attacks(AddAttack(E, x, y, direction)) **with**
Wdt(attack) = x
and Hgt(attack) = y
and Screen(attack) = Environnement(attack)
and Direction(attack) = direction
NbAttacks(Player(AddAttack(E, x, y, direction)))
= NbAttacks(Player(E)) - 1

2.7 Guard

Service : Guard **includes** Character

Observators :
const Id : [Guard] → int
const Resolver : [Guard] → PathResolver
Behaviour : [Guard] → Move
TimeInHole : [Guard] → int

Constructors :
init : Screen × int × int × int × PathResolver → [Guard]

Operators :
ClimbLeft : [Guard] → [Guard]
pre ClimbLeft(G) **requires** Screen::CellNature(Character::Envi(G), Character::Wdt(G), Character::Hgt(G)) ∈ {HOL}
ClimbRight : [Guard] → [Guard]
pre ClimbRight(G) **requires** Screen::CellNature(Character::Envi(G), Character::Wdt(G), Character::Hgt(G)) ∈ {HOL}

Observations :

[init] :
Id(init(e, x, y, id, resolver)) = id
Resolver(init(e, x, y, id, resolver)) = resolver
TimeInHole(init(e, x, y, id, resolver)) = 0

[ClimbLeft] :
Character::Wdt(G) = 0
$\Longrightarrow$ Character::Wdt(ClimbLeft(G)) = Character::Wdt(G)
and Character::Hgt(ClimbLeft(G)) = Character::Hgt(G)

Screen::CellNature(Character::Envi(G), Character::Wgt(G)-1, Character::Hgt(G)+1) ∈ {MTL, PLT}
$\Longrightarrow$ Character::Wdt(ClimbLeft(G))=Character::Wgt(G)
and Character::Hgt(ClimbLeft(G)) = Hgt(G)

Environnement::CellContent(Character::Envi(G), Character::Wgt(G)-1, Character::Hgt(G)+1) = Character c
$\Longrightarrow$ Character::Wdt(ClimbLeft(G)) = Character::Wgt(G)
and Character::Hgt(ClimbLeft(G)) = Character::Hgt(G)

Character::Wdt(G) $\neq$ 0
and Screen::CellNature(Character::Envi(G), Character::Wdt(G)-1,
Character::Hgt(G)+1) $\notin$ {PLT, MTL}
and Environnement::CellContent(Character::Envi(G), Character::Wdt(G)-1,
Character::Hgt(G)+1) $\neq$ Character c
$\Longrightarrow$ Character::Wdt(ClimbLeft(G)) = Character::Wdt(G)-1
and Character::Hgt(ClimbLeft(G)) = Character::Hgt(G)+1

[ClimbRight] : Character::Wdt(G) = Screen::Wdt(Character::Envi(G))-1
$\Longrightarrow$ Character::Wdt(ClimbLeft(G)) = Character::Wdt(G)
and Character::Hgt(ClimbLeft(G)) = Character::Hgt(G)

Screen::CellNature(Character::Envi(G), Character::Wgt(G)+1,
Character::Hgt(G)+1) $\in$ {MTL, PLT}
$\Longrightarrow$ Character::Wdt(ClimbLeft(G))=Character::Wgt(G)
and Character::Hgt(ClimbLeft(G)) = Hgt(G)

Environnement::CellContent(Character::Envi(G), Character::Wgt(G)+1,
Character::Hgt(G)+1) = Character c
$\Longrightarrow$ Character::Wdt(ClimbLeft(G)) = Character::Wgt(G)
and Character::Hgt(ClimbLeft(G)) = Character::Hgt(G)

Character::Wdt(G) $\neq$ Screen::Wdt(Character::Envi(G))-1
and Screen::CellNature(Character::Envi(G), Character::Wdt(G)+1,
Character::Hgt(G)+1) $\notin$ {PLT, MTL}
and Environnement::CellContent(Character::Envi(G), Character::Wdt(G)+1,
Character::Hgt(G)+1) $\neq$ Character c
$\Longrightarrow$ Character::Wdt(ClimbLeft(G)) = Character::Wdt(G)+1
and Character::Hgt(ClimbLeft(G)) = Character::Hgt(G)+1

[Step] : **WillFall(G) defined by**
Screen::CellNature(Character::Envi(G), Character::Wdt(G),
Character::Hgt(G)) $\notin$ {HDR, LAD, HOL}
and Screen::CellNature(Character::Envi(G), Character::Wdt(G),
Character::Hgt(G)-1) $\in$ {EMP, HOL, HDR}
and Environnement::CellContent(Character::Envi(G), Character::Wdt(G),
Character::Hgt(G)-1) $\neq$ Character c
$\Longrightarrow$ Step(G) = GoDown(G)
and LastAction(Step(G)) = DOWN

Screen::CellNature(Character::Envi(G), Character::Wdt(G), Character::Hgt(G))
$\in$ {HOL}
and TimeInHole(G) $<$ 2
$\Longrightarrow$ TimeInHole(Step(G)) = TimeInHole(G) + 1

WillClimbLeft(G) defined by
Screen::CellNature(Character::Envi(G), Character::Wdt(G),
Character::Hgt(G)) $\in$ {HOL}
and TimeInHole(G) = 2
and Behaviour(G) = LEFT
$\Longrightarrow$ Step(G) = ClimbLeft(G)
and LastAction(Step(G)) = LEFT
and TimeInHole(Step(G)) = 0

WillClimbRight(G) defined by

Screen::CellNature(Character::Envi(G), Character::Wdt(G), Character::Hgt(G)) ∈ {HOL}
and TimeInHole(G) = 2
and Behaviour(G) = RIGHT
⟹ Step(G) = ClimbRight(G)
and LastAction(Step(G)) = RIGHT
and TimeInHole(Step(G)) = 0

WillNotMove(G) defined by
Screen::CellNature(Character::Envi(G), Character::Wdt(G), Character::Hgt(G)) ∈ {HOL}
and TimeInHole(G) = 2
and Behaviour(G) = NEUTRAL
⟹ Behaviour(Step(G)) = Behaviour(G)
and TimeInHole(Step(G)) = 0

WillGoLeft(G) defined by
Screen::CellNature(Character::Envi(G), Character::Wdt(G), Character::Hgt(G)) ∉ {HOL}
and Behaviour(G) = LEFT
⟹ Step(G) = GoLeft(G)
and LastAction(Step(G)) = LEFT

WillGoRight(G) defined by
Screen::CellNature(Character::Envi(G), Character::Wdt(G), Character::Hgt(G)) ∉ {HOL}
and Behaviour(G) = Right
⟹ Step(G) = GoRight(G)
and LastAction(Step(G)) = RIGHT

2.8 PathResolver

Service : PathResolver

Observators :
const Player : [PathResolver] → Player
const Envi : [PathResolver] → Environnement
NodeIdOfPosition : [Guard] × int × int → int
pre NodeIdOfPosition(P, x, y) **requires** $0 \leq y <$ Screen::Height(Envi(P))
and $0 \leq x <$ Screen::Width(Envi(P))
WdtOfNode : [PathResolver] × int → int
HgtOfNode : [PathResolver] × int → int
NatureOfNode : [PathResolver] × int → Cell
Graphe : [PathResolver] → List<Integer>
ShortestPathToPlayer : [PathResolver] × Guard → int[]
NextMoveToReachPlayer : [PathResolver] × Guard → Move
PlayerCanReachPos : [PathResolver] × int × int → boolean

Constructors : init : Environnement × Player → [PathResolver]

Operators : RecomputeGraph : [PathResolver] → [PathResolver]

Observations :

[Invariants] :
forall *node* in [Array::Length(Graphe(P))]
∃ *neighbour* ∈ Graphe(P)[*node*] ⟹
(
(WdtOfNode(P, *node*) > 0
and WdtOfNode(P, *neighbour*) = WdtOfNode(P, *node*)-1
and HgtOfNode(P, *neighbour*) = HgtOfNode(P, *node*)
and NatureOfNode(P, *node*) ∉ {PLT,MTL}
and NatureOfNode(P, *neighbour*) ∉ {PLT,MTL}

and (
NatureOfNode(P, *node*) ∈ {LAD,HDR,HOL}
or Screen::CellNature(Envi(P), WdtOfNode(P, *node*), HgtOfNode(P, *node*)-1) *in* {PLT,MTL,LAD}
or Environnement::CellContent(Envi(P), WdtOfNode(P, *node*), HgtOfNode(P, *node*)-1) = Character otherCaract
)
)
or
(WdtOfNode(P, *node*) < Screen::Wdth(Envi(P)-1)
and WdtOfNode(P, *neighbour*) = WdtOfNode(P, *node*)+1
and HgtOfNode(P, *neighbour*) = HgtOfNode(P, *node*)
and NatureOfNode(P, *node*) ∉ {PLT,MTL}
and NatureOfNode(P, *neighbour*) ∉ {PLT,MTL}
and (
NatureOfNode(P, *node*) ∈ {LAD,HDR,HOL}
or Screen::CellNature(Envi(P), WdtOfNode(P, *node*), HgtOfNode(P, *node*)-1) *in* {PLT,MTL,LAD}
or Environnement::CellContent(Envi(P), WdtOfNode(P, *node*), HgtOfNode(P, *node*)-1) = Character otherCaract
)
)
or
(HgtOfNode(P, *node*) < Screen::Height(Envi(P)-1)
and WdtOfNode(P, *neighbour*) = WdtOfNode(P, *node*)
and HgtOfNode(P, *neighbour*) = HgtOfNode(P, *node*)+1
and NatureOfNode(P, *node*) ∈ {LAD}
and NatureOfNode(P, *neighbour*) ∈ {LAD,EMP}
)
or
(HgtOfNode(P, *node*) > 0
and WdtOfNode(P, *neighbour*) = WdtOfNode(P, *node*)
and HgtOfNode(P, *neighbour*) = HgtOfNode(P, *node*)-1
and NatureOfNode(P, *node*) ∈ {LAD,EMP,HDR}
and NatureOfNode(P, *neighbour*) ∈ {LAD,HDR,EMP,HOL}
)
or
(WdtOfNode(P, *node*) > 0
and HgtOfNode(P, *neighbour*) = HgtOfNode(P, *node*)-1
and WdtOfNode(P, *neighbour*) = WdtOfNode(P, *node*)-1
and NatureOfNode(P, *node*) ∉ {PLT,MTL,HDR,LAD,HOL}
and NatureOfNode(P, *neighbour*) ∈ {HDR}
)
or
(WdtOfNode(P, *node*) < Screen::Wdth(Envi(P)-1)
and HgtOfNode(P, *neighbour*) = HgtOfNode(P, *node*)-1
and WdtOfNode(P, *neighbour*) = WdtOfNode(P, *node*)+1
and NatureOfNode(P, *node*) ∉ {PLT,MTL,HDR,LAD,HOL}
and NatureOfNode(P, *neighbour*) ∈ {HDR}
)
or
(WdtOfNode(P, *node*) > 0
and HgtOfNode(P, *neighbour*) = HgtOfNode(P, *node*)+1
and WdtOfNode(P, *neighbour*) = WdtOfNode(P, *node*)-1
and NatureOfNode(P, *node*) ∈ {HDR,HOL}

and Screen::CellNature(Envi(P), Wdt(P, *node*)-1, Hgt(P, *node*)) ∈ {PLT,MTL}
and NatureOfNode(P, *neighbour*) ∈ {EMP,LAD,HDR,HOL}
)
or
(WdtOfNode(P, *node*) < Screen::Wdth(Envi(P)-1)
and HgtOfNode(P, *neighbour*) = HgtOfNode(P, *node*)+1
and WdtOfNode(P, *neighbour*) = WdtOfNode(P, *node*)+1
and NatureOfNode(P, *node*) ∈ {HDR,HOL}
and Screen::CellNature(Envi(P), Wdt(P, *node*)+1, Hgt(P, *node*)) ∈ {PLT,MTL}
and NatureOfNode(P, *neighbour*) ∈ {EMP,LAD,HDR,HOL}
)
)

[Invariants] : ShortestPathToPlayer(P, g) **min**
Soit $playerNode \stackrel{\text{def}}{=}$ NodeIdOfPosition(P, Character::Wdt(Player(P)), Character::Hgt(Player(P))),
Soit $guardNode \stackrel{\text{def}}{=}$ NodeIdOfPosition(P, Character::Wdt(g), Character::Hgt(g)),
∃ ShortestPath($guardNode$, $playerNode$ ⟺
Soit $currentNode \stackrel{\text{def}}{=} playerNode$,
tant que $currentNode \neq guardNode$
$currentNode \longleftarrow$ ShortestPathToPlayer(P, g)[$currentNode$]

[Invariants] : NextMoveToReachPlayer(P, g) **min**
NextMoveToReachPlayer(P, g) = LEFT ⟺
(
(
(
(
NatureOfNode(P, NodeIdOfPosition(P, Character::Wdt(g), Character::Hgt(g))) ∉ {PLT,MTL,HDR,LAD,HOL}
and NatureOfNode(P, NodeIdOfPosition(P, Character::Wdt(g)-1, Character::Hgt(g)-1)) ∈ {HDR}
)
⟹ ShortestPathToPlayer(P, g)[NodeIdOfPosition(P, Character::Wdt(g)-1, Character::Hgt(g)-1)] = NodeIdOfPosition(P, Character::Wdt(g), Character::Hgt(g)))
)
and
(
(
NatureOfNode(P, NodeIdOfPosition(P, Character::Wdt(g), Character::Hgt(g))) ∈ {HDR,HOL}
and NatureOfNode(P, NodeIdOfPosition(P, Character::Wdt(g)-1, Character::Hgt(g))) ∈ {PLT,MTL}
and NatureOfNode(P, NodeIdOfPosition(P, Character::Wdt(g)-1, Character::Hgt(g)+1)) ∈ {EMP,LAD,HDR,HOL}
)
⟹ ShortestPathToPlayer(P, g)[NodeIdOfPosition(P, Character::Wdt(g)-1, Character::Hgt(g)+1)] = NodeIdOfPosition(P, Character::Wdt(g), Character::Hgt(g)))
)
)

or ShortestPathToPlayer(P, g)[NodeIdOfPosition(P, Character::Wdt(g)-1, Character::Hgt(g))] = NodeIdOfPosition(P, Character::Wdt(g), Character::Hgt(g)))
)
NextMoveToReachPlayer(P, g) = RIGHT $\iff$
(
(
(
(
NatureOfNode(P, NodeIdOfPosition(P, Character::Wdt(g), Character::Hgt(g))) $\notin$ {PLT,MTL,HDR,LAD,HOL}
and NatureOfNode(P, NodeIdOfPosition(P, Character::Wdt(g)+1, Character::Hgt(g)-1)) $\in$ {HDR}
)
$\implies$ ShortestPathToPlayer(P, g)[NodeIdOfPosition(P, Character::Wdt(g)+1, Character::Hgt(g)-1)] = NodeIdOfPosition(P, Character::Wdt(g), Character::Hgt(g)))
)
and
(
(
NatureOfNode(P, NodeIdOfPosition(P, Character::Wdt(g), Character::Hgt(g))) $\in$ {HDR,HOL}
and NatureOfNode(P, NodeIdOfPosition(P, Character::Wdt(g)+1, Character::Hgt(g))) $\in$ {PLT,MTL}
and NatureOfNode(P, NodeIdOfPosition(P, Character::Wdt(g)+1, Character::Hgt(g)+1)) $\in$ {EMP,LAD,HDR,HOL}
)
$\implies$ ShortestPathToPlayer(P, g)[NodeIdOfPosition(P, Character::Wdt(g)+1, Character::Hgt(g)+1)] = NodeIdOfPosition(P, Character::Wdt(g), Character::Hgt(g)))
)
)
or ShortestPathToPlayer(P, g)[NodeIdOfPosition(P, Character::Wdt(g)+1, Character::Hgt(g))] = NodeIdOfPosition(P, Character::Wdt(g), Character::Hgt(g)))
)
NextMoveToReachPlayer(P, g) = DOWN $\iff$
ShortestPathToPlayer(P, g)[NodeIdOfPosition(P, Character::Wdt(g), Character::Hgt(g)-1)] = NodeIdOfPosition(P, Character::Wdt(g), Character::Hgt(g)))
NextMoveToReachPlayer(P, g) = UP $\iff$
ShortestPathToPlayer(P, g)[NodeIdOfPosition(P, Character::Wdt(g), Character::Hgt(g)+1)] = NodeIdOfPosition(P, Character::Wdt(g), Character::Hgt(g)))

[Invariants] : PlayerCanReachPos(P, posX, posY) **min**
Soit $playerNode \stackrel{\text{def}}{=}$ NodeIdOfPosition(P, Character::Wdt(Player(P)), Character::Hgt(Player(P)))
Soit $dstNode \stackrel{\text{def}}{=}$ NodeIdOfPosition(P, posX, posY)
playerCanReachPos $\iff \exists$ ShortestPath($playerNode$, $dstNode$) $\in$ Graphe(P)

[Invariants] : PlayerCanReachAllTreasuresAndPortal(P, treasures, portalWdt, portalHgt) **min**

Soit $playerNode \stackrel{\text{def}}{=}$ NodeIdOfPosition(P, Character::Wdt(Player(P)), Character::Hgt(Player(P)))
Soit $portalNode \stackrel{\text{def}}{=}$ NodeIdOfPosition(P, portalWdt, portalHgt)
∃ ShortestPath($playerNode$, $portalNode$) ∈ Graphe(P)
and forall t in treasures
Soit $treasureNode \stackrel{\text{def}}{=}$ NodeIdOfPosition(P, GameObject::Wdt(t), GameObject::Hgt(t))
∃ ShortestPath($playerNode$, $treasureNode$) ∈ Graphe(P)
and ∃ ShortestPath($portalNode$, $treasureNode$) ∈ Graphe(P)
and ∃ ShortestPath($treasureNode$, $portalNode$) ∈ Graphe(P)
and forall t in treasures
Soit $srcNode \stackrel{\text{def}}{=}$ NodeIdOfPosition(P, GameObject::Wdt(t), GameObject::Hgt(t))
and forall $t2$ in treasures
Soit $dstNode \stackrel{\text{def}}{=}$ NodeIdOfPosition(P, GameObject::Wdt($t2$), GameObject::Hgt($t2$))
and ∃ ShortestPath($srcNode$, $dstNode$) ∈ Graphe(P)

[init] :
Envi(init(e, player)) = e
Player(init(e, player)) = player

2.9 GameObject

Service : GameObject
Observators : Hgt : [GameObject] → int
Wdt : [GameObject] → int
const Screen : [GameObject] → Screen
Constructors : init : Screen × int × int → [GameObject]
pre init(S, x, y) **requires** 0 > x > Screen : :Wdt(S) **and** 0 > y > Screen : :Hgt(S)
Observations :
[init] :
Wdt(init(e, x, y)) = x
Hgt(init(e, x, y)) = y
Screen(init(e, x, y)) = e

2.10 Attack

Service : Attack **includes** GameObject
Observators : **const** Direction : [Attack] → Direction
Operators : step : [Attack] → [Attack]
pre Step(A) **requires** 0 ≤ GameObject::wdt(A) < Screen::Wdt(Screen(A))
Constructors : init : Screen × int × int × Direction → [Attack]
pre init(S, x, y, direction) **requires** direction ∈ { LEFT, RIGHT }
Observations :
[init] : Direction(init(S, x, y, direction)) = direction

[step] : Hgt(Step(E)) = Hgt(A)
Direction(A) = LEFT ⟹ Wdt(Step(A)) = Wdt(A) - 1
Direction(A) = RIGHT ⟹ Wdt(Step(A)) = Wdt(A) + 1

2.11 Board

Service : Board
Observators : **const** Engine : [Board] → Engine
const LoadRunnerFrame : [Board] → LoadRunnerFrame
const BWidth : [Board] → int
const BHeight : [Board] → int
const BLOCS_SIZE : [Board] → int
const PLAYER_SIZE : [Board] → int
const GUARD_SIZE : [Board] → int
const TREASURE_SIZE : [Board] → int
const PTL_SIZE : [Board] → int
const MTL_SIZE : [Board] → int
const HDR_SIZE : [Board] → int
const LAD_SIZE : [Board] → int
const PRT_SIZE : [Board] → int
const ATK_SIZE : [Board] → int
const DELAY : [Board] → int
Envi : [Board] → Environnement
DisplayPortal : [Board] → Boolean
Operators : nextLevel : [Board] → [Board]
Constructors : init : Engine × Environnement × LodeRunnerFrame → [Board]
Observations :
[init] : Engine(init(eng, envi, frame)) = eng
Environnement(init(eng, envi, frame)) = envi
Frame(init(eng, envi, frame)) = frame
WIDTH(init(eng, envi, frame)) = BLOC_SIZE × Wdt(eng)
HEIGHT(init(eng, envi, frame)) = BLOC_SIZE × Hgt(eng)
BLOC_SIZE(init(eng, envi, frame)) = 30
PLAYER_SIZE(init(eng, envi, frame)) = 30
GUARD_SIZE(init(eng, envi, frame)) = 30
TREASURE_SIZE(init(eng, envi, frame)) = 30
PLT_SIZE(init(eng, envi, frame)) = 30
MTL_SIZE(init(eng, envi, frame)) = 30
LAD_SIZE(init(eng, envi, frame)) = 30
HDR_SIZE(init(eng, envi, frame)) = 30
DELAY_SIZE(init(eng, envi, frame)) = 30
DisplayPortal(init(eng, envi, frame)) = false

[nextLevel] : DisplayPortal(nextLevel(B, envi)) = false

	Envi(nextLevel(B, envi)) = envi
[displayPortal] :	DisplayPortal(displayPortal(B, dis)) = dis Envi(displayPortal(B, dis)) = Envi(B)

3 Description formelle des tests *MBT*

3.1 Screen

Les tests de *dig* et *fill* sont reportés dans les tests du service EditableScreen. Cela est dû au fait qu'il est impossible de modifier les cellules d'un Screen (contrairement à celle d'un EditableScreen). Il est ainsi impossible de mettre en place un Screen contenant des HOL et des PLT nécessaire à la réalisation des tests de Dig et Fill.

Objectif 1 : Tests de Init(width, height)

Cas de test 1	Screen::testInitPositif1
Objectif du test	S'assurer que le init avec des valeurs positives ne renvoie pas d'erreur.
Condition initiales	aucune
Opérations	$S_1 \stackrel{\text{def}}{=} init(7, 12);$
Oracle	Pas d'exception levée

Cas de test 2	Screen::testInitPositif2
Objectif du test	S'assurer que le init avec les valeurs positives minimales ne renvoie pas d'erreur.
Condition initiales	aucune
Opérations	$S_1 \stackrel{\text{def}}{=} init(1, 1);$
Oracle	Pas d'exception levée

Cas de test 3	Screen::testInitNegatif1
Objectif du test	S'assurer que le init avec une hauteur nulle renvoie une erreur.
Condition initiales	aucune
Opérations	$S_1 \stackrel{\text{def}}{=} init(7, 0);$
Oracle	Exception levée

Cas de test 4	Screen::testInitNegatif2
Objectif du test	S'assurer que le init avec une largeur nulle renvoie une erreur.
Condition initiales	aucune
Opérations	$S_1 \stackrel{\text{def}}{=} init(0, 12);$
Oracle	Exception levée

Cas de test 5	Screen::testInitNegatif3
Objectif du test	S'assurer que le init avec des valeurs négatives renvoie une erreur.

Condition initiales	aucune
Opérations	$S_1 \stackrel{\text{def}}{=} init(-1, -1);$
Oracle	Exception levée

Description des tests du Init du service Screen

Objectif 2 : Tests de CellNature(S, x, y)

Cas de test 1	Screen::testCellNaturePositif1
Objectif du test	S'assurer que le CellNature avec des valeurs positives ne renvoie pas d'erreur.
Condition initiales	$S_1 \stackrel{\text{def}}{=} init(7, 12);$
Opérations	$S_2 \stackrel{\text{def}}{=} CellNature(S_1, 3, 6);$
Oracle	Pas d'exception levée

Cas de test 2	Screen::testCellNaturePositif2
Objectif du test	S'assurer qu'un CellNature à la bordure maximale du Screen ne renvoie pas d'erreur.
Condition initiales	$S_1 \stackrel{\text{def}}{=} init(7, 12);$
Opérations	$S_2 \stackrel{\text{def}}{=} CellNature(S_1, 6, 11);$
Oracle	Pas d'exception levée

Cas de test 3	Screen::testCellNaturePositif3
Objectif du test	S'assurer qu'un CellNature à la bordure minimale du Screen ne renvoie pas d'erreur.
Condition initiales	$S_1 \stackrel{\text{def}}{=} init(7, 12);$
Opérations	$S_2 \stackrel{\text{def}}{=} CellNature(S_1, 0, 0);$
Oracle	Pas d'exception levée

Cas de test 4	Screen::testCellNatureNegatif1
Objectif du test	S'assurer que le CellNature avec un x négatif renvoie une erreur.
Condition initiales	$S_1 \stackrel{\text{def}}{=} init(7, 12);$
Opérations	$S_2 \stackrel{\text{def}}{=} CellNature(S_1, -1, 0);$
Oracle	Exception levée

Cas de test 5	Screen::testCellNatureNegatif2
Objectif du test	S'assurer que le CellNature avec un y négatif renvoie une erreur.
Condition initiales	$S_1 \stackrel{\text{def}}{=} init(7, 12);$
Opérations	$S_2 \stackrel{\text{def}}{=} CellNature(S_1, 0, -1);$
Oracle	Exception levée

Cas de test 6	Screen::testCellNatureNegatif3
Objectif du test	S'assurer que le CellNature avec un x et un y négatifs renvoie une erreur.
Condition initiales	$S_1 \stackrel{\text{def}}{=} init(7, 12)$;
Opérations	$S_2 \stackrel{\text{def}}{=} CellNature(S_1, -1, -1)$;
Oracle	Exception levée

Cas de test 7	Screen::testCellNatureNegatif4
Objectif du test	S'assurer que le CellNature avec un x égal à la largeur du Screen renvoie une erreur.
Condition initiales	$S_1 \stackrel{\text{def}}{=} init(7, 12)$;
Opérations	$S_2 \stackrel{\text{def}}{=} CellNature(S_1, 7, 0)$;
Oracle	Exception levée

Cas de test 8	Screen::testCellNatureNegatif5
Objectif du test	S'assurer que le CellNature avec un y égal à la hauteur du Screen renvoie une erreur.
Condition initiales	$S_1 \stackrel{\text{def}}{=} init(7, 12)$;
Opérations	$S_2 \stackrel{\text{def}}{=} CellNature(S_1, 0, 12)$;
Oracle	Exception levée

Cas de test 9	Screen::testCellNatureNegatif6
Objectif du test	S'assurer que le CellNature avec un x égal à la largeur du Screen et un y égale à la hauteur du Screen renvoie une erreur.
Condition initiales	$S_1 \stackrel{\text{def}}{=} init(7, 12)$;
Opérations	$S_2 \stackrel{\text{def}}{=} CellNature(S_1, 7, 12)$;
Oracle	Exception levée

Description des tests de CellNature du service Screen

3.2 EditableScreen

Objectif 1 : Tests de Playable(E)

Cas de test 1	EditableScreen::testPlayablePositif
Objectif du test	S'assurer qu'à l'initialisation, un EditableScreen est jouable.
Condition initiales	$E_1 \stackrel{\text{def}}{=} init(7, 12)$;
Opérations	$Playable(E_1)$;
Oracle	L'opération renvoie $true$

Cas de test 2	EditableScreen::testPlayableNegatif1
Objectif du test	S'assurer qu'en plaçant un PLT en bas du Screen, l'EditableScreen n'est pas jouable.

Condition initiales	$E_1 \stackrel{\text{def}}{=} SetNature(init(7,12),0,0,PLT)$;
Opérations	$Playable(E_1)$;
Oracle	L'opération renvoie $false$

Cas de test 3	EditableScreen::testPlayableNegatif2
Objectif du test	S'assurer qu'en plaçant un PLT à la dernière case de la première ligne, l'EditableScreen n'est pas jouable.
Condition initiales	$E_1 \stackrel{\text{def}}{=} SetNature(init(7,12),6,0,PLT)$;
Opérations	$Playable(E_1)$;
Oracle	L'opération renvoie $false$
Cas de test 4	EditableScreen::testPlayableNegatif3
Objectif du test	S'assurer qu'en plaçant un HOL en haut à droite, l'EditableScreen n'est pas jouable.
Condition initiales	$E_1 \stackrel{\text{def}}{=} SetNature(init(7,12),6,11,HOL)$;
Opérations	$Playable(E_1)$;
Oracle	L'opération renvoie $false$

Cas de test 5	EditableScreen::testPlayableNegatif4
Objectif du test	S'assurer qu'en plaçant un HOL en bas à gauche, l'EditableScreen n'est pas jouable.
Condition initiales	$E_1 \stackrel{\text{def}}{=} SetNature(init(7,12),0,0,HOL)$;
Opérations	$Playable(E_1)$;
Oracle	L'opération renvoie $false$

Description des tests de Playable du service EditableScreen

Objectif 2 : Tests de SetNature(E, x, y, c)

Cas de test 1	EditableScreen::testSetNaturePositif1
Objectif du test	S'assurer qu'un SetNature à la borne inférieure du Screen ne renvoie pas d'erreur.
Condition initiales	$E_1 \stackrel{\text{def}}{=} init(7,12)$;
Opérations	$SetNature(E_1,0,0PLT)$;
Oracle	Pas d'exception levée.

Cas de test 2	EditableScreen::testSetNaturePositif2
Objectif du test	S'assurer qu'un SetNature à la borne supérieure du Screen ne renvoie pas d'erreur.
Condition initiales	$E_1 \stackrel{\text{def}}{=} init(7,12)$;
Opérations	$SetNature(E_1,6,11LAD)$;
Oracle	Pas d'exception levée.

Cas de test 3	EditableScreen::testSetNatureNegatif1

Objectif du test	S'assurer qu'un SetNature au delà de la borne supérieure du Screen renvoie une erreur.
Condition initiales	$E_1 \stackrel{\text{def}}{=} init(7, 12)$;
Opérations	$SetNature(E_1, 7, 12LAD)$;
Oracle	Exception levée.

Cas de test 4	EditableScreen::testSetNatureNegatif2
Objectif du test	S'assurer qu'un SetNature avec un y supérieur à la hauteur du Screen renvoie une erreur.
Condition initiales	$E_1 \stackrel{\text{def}}{=} init(7, 12)$;
Opérations	$SetNature(E_1, 0, 12LAD)$;
Oracle	Exception levée.

Cas de test 5	EditableScreen::testSetNatureNegatif3
Objectif du test	S'assurer qu'un SetNature avec un x supérieur à la largeur du Screen renvoie une erreur.
Condition initiales	$E_1 \stackrel{\text{def}}{=} init(7, 12)$;
Opérations	$SetNature(E_1, 7, 0LAD)$;
Oracle	Exception levée.

Cas de test 6	EditableScreen::testSetNatureNegatif4
Objectif du test	S'assurer qu'un SetNature avec un x négatif renvoie une erreur.
Condition initiales	$E_1 \stackrel{\text{def}}{=} init(7, 12)$;
Opérations	$SetNature(E_1, -1, 11LAD)$;
Oracle	Exception levée.

Cas de test 7	EditableScreen::testSetNatureNegatif5
Objectif du test	S'assurer qu'un SetNature avec un y négatif renvoie une erreur.
Condition initiales	$E_1 \stackrel{\text{def}}{=} init(7, 12)$;
Opérations	$SetNature(E_1, 0, -1LAD)$;
Oracle	Exception levée.

Cas de test 8	EditableScreen::testSetNatureNegatif6
Objectif du test	S'assurer qu'un SetNature avec un x et un y tous deux négatifs renvoie une erreur.
Condition initiales	$E_1 \stackrel{\text{def}}{=} init(7, 12)$;
Opérations	$SetNature(E_1, -1, -1LAD)$;
Oracle	Exception levée.

Description des tests de SetNature du service EditableScreen

Objectif 3 : Tests de Dig(S, x, y)

Cas de test 1	Screen::testDigPositif1
Objectif du test	S'assurer qu'un Dig sur un PLT ne renvoie pas d'erreur et change bien la case en un HOL.
Condition initiales	$E_1 \stackrel{\text{def}}{=} SetNature(init(7,12),1,1PLT)$;
Opérations	$E_2 \stackrel{\text{def}}{=} Dig(E_1,1,1)$;
Oracle	$CellNature(E_2,1,1) = HOL$
Cas de test 2	Screen::testDigPositif2
Objectif du test	S'assurer qu'un Dig sur un PLT ne renvoie pas d'erreur et change bien la case en un HOL.
Condition initiales	$E_1 \stackrel{\text{def}}{=} SetNature(init(7,12),1,1PLT)$;
Opérations	$E_2 \stackrel{\text{def}}{=} Dig(E_1,6,11)$;
Oracle	$CellNature(E_2,6,11) = HOL$
Cas de test 3	Screen::testDigNegatif
Objectif du test	S'assurer qu'un Dig sur autre chose qu'un PLT renvoie une d'erreur.
Condition initiales	$E_1 \stackrel{\text{def}}{=} init(7,12)$;
Opérations	$E_2 \stackrel{\text{def}}{=} Dig(E_1,0,0)$;
Oracle	Exception levée.

Description des tests de Dig du service Screen

Objectif 3 : Tests de Fill(S, x, y)

Cas de test 1	Screen::testFillPositif1
Objectif du test	S'assurer qu'un Fill sur un HOL ne renvoie pas d'erreur et change bien la case en un PLT.
Condition initiales	$E_1 \stackrel{\text{def}}{=} SetNature(init(7,12),1,1HOL)$;
Opérations	$E_2 \stackrel{\text{def}}{=} Fill(E_1,1,1)$;
Oracle	$CellNature(E_2,1,1) = PLT$
Cas de test 2	Screen::testFillPositif2
Objectif du test	S'assurer qu'un Fill sur un Hol ne renvoie pas d'erreur et change bien la case en un PLT.
Condition initiales	$E_1 \stackrel{\text{def}}{=} SetNature(init(7,12),1,1HOL)$;
Opérations	$E_2 \stackrel{\text{def}}{=} Fill(E_1,6,11)$;
Oracle	$CellNature(E_2,6,11) = PLT$
Cas de test 3	Screen::testFillNegatif
Objectif du test	S'assurer qu'un Fill sur autre chose qu'un Hol renvoie une d'erreur.

Condition initiales	$E_1 \stackrel{\text{def}}{=} init(7, 12)$;
Opérations	$E_2 \stackrel{\text{def}}{=} Fill(E_1, 0, 0)$;
Oracle	Exception levée.

Description des tests de Fill du service Screen

3.3 Environnement

Pour les tests d'environnement, le terrain est de la forme suivante :

$$cells \stackrel{\text{def}}{=} \begin{bmatrix} EMP & EMP & LAD & EMP & EMP \\ EMP & EMP & EMP & EMP & EMP \\ PLT & PLT & LAD & PLT & PLT \\ EMP & EMP & LAD & EMP & EMP \\ MTL & MTL & MTL & MTL & MTL \end{bmatrix}$$

Dans l'ensemble de ces tests, la variable E_0 représente un environnement avant son appel à *init*. En *Java*, cela s'apparenterait à la création d'un objet avec un `new`.

Objectif 1 : Test de init(w, h, player, guards, treasures, cells)

Cas de test 1	EditableScreen::testInitPositif
Objectif du test	S'assurer que le init ne renvoie pas d'erreur.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: Create(Guard :: init(E_0, 1, 1, 1, null), Guard :: init(E_0, 1, 3, 2, null))$ $treasures \stackrel{\text{def}}{=} List :: Create(Treasure :: init(E_0, 0, 3), Treasure :: init(E_0, 1, 3))$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.3
Opérations	$E_1 \stackrel{\text{def}}{=} init(5, 6, player, guards, treasures, cells)$
Oracle	Pas d'exception levée.

Description du test du Init du service Environnement

Objectif 2 : Tests de CellContent(E, x, y)

Cas de test 1	EditableScreen::testCellContentPositif1
Objectif du test	S'assurer qu'un CellContent en bas à gauche ne renvoie pas d'erreur.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: Create(Guard :: init(E_0, 1, 1, 1, null), Guard :: init(E_0, 1, 3, 2, null))$ $treasures \stackrel{\text{def}}{=} List :: Create(Treasure :: init(E_0, 0, 3), Treasure :: init(E_0, 1, 3))$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.3

	$E_1 \stackrel{\text{def}}{=} init(5, 6, player, guards, treasures, cells)$
Opérations	$E_2 \stackrel{\text{def}}{=} CellContent(E_1, 0, 0)$
Oracle	Pas d'exception levée.

Cas de test 2	EditableScreen::testCellContentPositif2
Objectif du test	S'assurer qu'un CellContent sur une case sans Character ni trésor renvoie bien *null* pour ces deux cas là.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: Create(Guard :: init(E_0, 1, 1, 1, null), Guard :: init(E_0, 1, 3, 2, null))$ $treasures \stackrel{\text{def}}{=} List :: Create(Treasure :: init(E_0, 0, 3), Treasure :: init(E_0, 1, 3))$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.3 $E_1 \stackrel{\text{def}}{=} init(5, 6, player, guards, treasures, cells)$
Opérations	$hasChar \stackrel{\text{def}}{=} Couple :: Character(CellContent(E_1, 4, 5)) \neq null$ $hasItem \stackrel{\text{def}}{=} Couple :: Item(CellContent(E_1, 4, 5)) \neq null$
Oracle	$hasChar = false$ **and** $hasItem = false$.

Cas de test 3	EditableScreen::testCellContentPositif3
Objectif du test	S'assurer qu'un CellContent sur une case contenant un garde mais pas de trésor renvoie bien le garde en question et *null* pour le trésor.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: Create(Guard :: init(E_0, 1, 1, 1, null), Guard :: init(E_0, 1, 3, 2, null))$ $treasures \stackrel{\text{def}}{=} List :: Create(Treasure :: init(E_0, 0, 3), Treasure :: init(E_0, 1, 3))$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.3 $E_1 \stackrel{\text{def}}{=} init(5, 6, player, guards, treasures, cells)$
Opérations	$hasChar \stackrel{\text{def}}{=} Couple :: Character(CellContent(E_1, 1, 1)) \neq null$ $hasItem \stackrel{\text{def}}{=} Couple :: Item(CellContent(E_1, 1, 1)) \neq null$
Oracle	$hasChar = true$ **and** $hasItem = false$.

Cas de test 4	EditableScreen::testCellContentPositif4
Objectif du test	S'assurer qu'un CellContent sur une case contenant un trésor mais pas de garde renvoie bien le trésor en question et *null* pour le garde.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: Create(Guard :: init(E_0, 1, 1, 1, null), Guard :: init(E_0, 1, 3, 2, null))$

	$treasures \stackrel{\text{def}}{=} List :: Create(Treasure :: init(E_0, 0, 3), Treasure :: init(E_0, 1, 3))$
	$player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$
	voir valeur *cells* 3.3
	$E_1 \stackrel{\text{def}}{=} init(5, 6, player, guards, treasures, cells)$
Opérations	$hasChar \stackrel{\text{def}}{=} Couple :: Character(CellContent(E_1, 0, 3)) \neq null$
	$hasItem \stackrel{\text{def}}{=} Couple :: Item(CellContent(E_1, 0, 3)) \neq null$
Oracle	$hasChar = false$ **and** $hasItem = true$.

Cas de test 5	EditableScreen::testCellContentPositif5
Objectif du test	S'assurer qu'un CellContent sur une case contenant un garde et un trésor renverra les références vers ces deux valeurs.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: Create(Guard :: init(E_0, 1, 1, 1, null), Guard :: init(E_0, 1, 3, 2, null))$
	$treasures \stackrel{\text{def}}{=} List :: Create(Treasure :: init(E_0, 0, 3), Treasure :: init(E_0, 1, 3))$
	$player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$
	voir valeur *cells* 3.3
	$E_1 \stackrel{\text{def}}{=} init(5, 6, player, guards, treasures, cells)$
Opérations	$hasChar \stackrel{\text{def}}{=} Couple :: Character(CellContent(E_1, 1, 3)) \neq null$
	$hasItem \stackrel{\text{def}}{=} Couple :: Item(CellContent(E_1, 1, 3)) \neq null$
Oracle	$hasChar = true$ **and** $hasItem = true$.

Cas de test 6	EditableScreen::testCellContentNegatif1
Objectif du test	S'assurer qu'un CellContent sur une case en dehors du Screen (de coordonnée inférieure aux bornes) renvoie une erreur.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: Create(Guard :: init(E_0, 1, 1, 1, null), Guard :: init(E_0, 1, 3, 2, null))$
	$treasures \stackrel{\text{def}}{=} List :: Create(Treasure :: init(E_0, 0, 3), Treasure :: init(E_0, 1, 3))$
	$player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$
	voir valeur *cells* 3.3
	$E_1 \stackrel{\text{def}}{=} init(5, 6, player, guards, treasures, cells)$
Opérations	$CellContent(E_1, -1, -1)$
Oracle	Exception levée.

Cas de test 7	EditableScreen::testCellContentNegatif2

Objectif du test	S'assurer qu'un CellContent sur une case en dehors du Screen (de coordonnée supérieure aux bornes) renvoie une erreur.
Condition initiales	$guards \overset{\text{def}}{=} List :: Create(Guard :: init(E_0, 1, 1, 1, null), Guard :: init(E_0, 1, 3, 2, null))$ $treasures \overset{\text{def}}{=} List :: Create(Treasure :: init(E_0, 0, 3), Treasure :: init(E_0, 1, 3))$ $player \overset{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.3 $E_1 \overset{\text{def}}{=} init(5, 6, player, guards, treasures, cells)$
Opérations	$CellContent(E_1, 5, 6)$
Oracle	Exception levée.

Description des tests de CellContent du service Environnement

3.4 Character

Pour les tests du service Character, le terrain est de la forme suivante :

$$cells \overset{\text{def}}{=} \begin{bmatrix} EMP & EMP & LAD & EMP & EMP & EMP & EMP \\ HDR & HDR & LAD & HDR & HDR & EMP & EMP \\ HDR & EMP & LAD & EMP & HDR & EMP & EMP \\ PLT & PLT & LAD & PLT & HDR & HDR & PLT \\ EMP & EMP & LAD & EMP & EMP & EMP & EMP \\ MLT & MLT & MLT & MLT & MLT & MLT & MLT \end{bmatrix}$$

L'ensemble des tests est divisé en deux parties, la première représente les tests d'un character dans un environnement vide (sans garde ni trésor), et la seconde est constituée de tests dans un environnement contenant des gardes et des trésors.
Pour pouvoir utiliser un environnement, il est nécessaire d'avoir un objet Player. Nous utilisons donc en plus du Character testé, un Player qui est la représentation du Character dans l'environnement.

Dans l'ensemble de ces tests, la variable E_0 représente un environnement avant son appel à *init*.

3.4.1 Tests dans un environnement vide

Objectif 1 : Tests de init(environnement, x, y)

Cas de test 1	Character::testInitPositif1
Objectif du test	S'assurer que le init ne renvoie pas d'erreur.
Condition initiales	$guards \overset{\text{def}}{=} List :: CreateEmpty()$ $treasures \overset{\text{def}}{=} List :: CreateEmpty()$ $player \overset{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.4 $envi \overset{\text{def}}{=} Environnement :: init(7, 6, player, guards, treasures, cells)$
Opérations	$C_1 \overset{\text{def}}{=} init(envi, 1, 1)$

Oracle	Pas d'exception levée.

Cas de test 2	Character::testInitPositif2
Objectif du test	S'assurer que le init ne renvoie pas d'erreur lorsque l'on est à la borne supérieure du Screen.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: CreateEmpty()$ $treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.4 $envi \stackrel{\text{def}}{=} Environnement ::$ $init(7, 6, player, guards, treasures, cells)$
Opérations	$C_1 \stackrel{\text{def}}{=} init(envi, 6, 5)$
Oracle	Pas d'exception levée.

Cas de test 3	Character::testInitNegatif1
Objectif du test	S'assurer que le init renvoie une erreur si on initialise le personnage avec un x négatif.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: CreateEmpty()$ $treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.4 $envi \stackrel{\text{def}}{=} Environnement ::$ $init(7, 6, player, guards, treasures, cells)$
Opérations	$C_1 \stackrel{\text{def}}{=} init(envi, -1, 0)$
Oracle	Exception levée.

Cas de test 4	Character::testInitNegatif2
Objectif du test	S'assurer que le init renvoie une erreur si on initialise le personnage avec un y négatif.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: CreateEmpty()$ $treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.4 $envi \stackrel{\text{def}}{=} Environnement ::$ $init(7, 6, player, guards, treasures, cells)$
Opérations	$C_1 \stackrel{\text{def}}{=} init(envi, 0, -1)$
Oracle	Exception levée.

Cas de test 5	Character::testInitNegatif3
Objectif du test	S'assurer que le init renvoie une erreur si on initialise le personnage avec un x et un y négatifs.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: CreateEmpty()$

	$treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.4 $envi \stackrel{\text{def}}{=} Environnement ::$ $init(7, 6, player, guards, treasures, cells)$
Opérations	$C_1 \stackrel{\text{def}}{=} init(envi, -1, -1)$
Oracle	Exception levée.

Cas de test 6	Character::testInitNegatif4
Objectif du test	S'assurer que le init renvoie une erreur si on initialise le personnage avec un x égal à la largeur du Screen.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: CreateEmpty()$ $treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.4 $envi \stackrel{\text{def}}{=} Environnement ::$ $init(7, 6, player, guards, treasures, cells)$
Opérations	$C_1 \stackrel{\text{def}}{=} init(envi, 7, 0)$
Oracle	Exception levée.

Cas de test 7	Character::testInitNegatif5
Objectif du test	S'assurer que le init renvoie une erreur si on initialise le personnage avec un y égal à la hauteur du Screen.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: CreateEmpty()$ $treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.4 $envi \stackrel{\text{def}}{=} Environnement ::$ $init(7, 6, player, guards, treasures, cells)$
Opérations	$C_1 \stackrel{\text{def}}{=} init(envi, 0, 6)$
Oracle	Exception levée.

Cas de test 8	Character::testInitNegatif6
Objectif du test	S'assurer que le init renvoie une erreur si on initialise le personnage avec x égal à la largeur du Screen, et un y égal à la hauteur du Screen.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: CreateEmpty()$ $treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.4

	$envi \stackrel{\text{def}}{=} Environnement :: init(7, 6, player, guards, treasures, cells)$
Opérations	$C_1 \stackrel{\text{def}}{=} init(envi, 7, 6)$
Oracle	Exception levée.

Cas de test 9	Character::testInitNegatif7
Objectif du test	S'assurer que le init renvoie une erreur si on initialise le personnage dans une case différente d'un EMP.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: CreateEmpty()$ $treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.4 $envi \stackrel{\text{def}}{=} Environnement :: init(7, 6, player, guards, treasures, cells)$
Opérations	$C_1 \stackrel{\text{def}}{=} init(envi, 0, 0)$
Oracle	Exception levée.

Description des tests du Init du service Character dans un environnement vide

Objectif 2 : Tests de GoLeft(C)

Cas de test 1	Character::testGoLeft1
Objectif du test	S'assurer que le GoLeft ne renvoie pas d'erreur.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: CreateEmpty()$ $treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.4 $envi \stackrel{\text{def}}{=} Environnement :: init(7, 6, player, guards, treasures, cells)$ $C_1 \stackrel{\text{def}}{=} init(envi, 1, 1)$
Opérations	$GoLeft(C_1)$;
Oracle	Pas d'exception levée.

Cas de test 2	Character::testGoLeft2
Objectif du test	S'assurer que les deux GoLeft feront bouger le personnage de deux cases vers la gauche.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: CreateEmpty()$ $treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.4

	$envi \stackrel{\text{def}}{=} Environnement :: init(7, 6, player, guards, treasures, cells)$
	$C_1 \stackrel{\text{def}}{=} GoRight(GoRight(init(envi, 1, 1)))$
Opérations	$C_2 \stackrel{\text{def}}{=} GoLeft(GoLeft(C_1))$;
Oracle	$Wdt(C_2) = 1$ **and** $Hgt(C_2) = 1$.

Cas de test 3	Character::testGoLeft3
Objectif du test	S'assurer que des GoLeft face à un mur à gauche du joueur ne le feront pas bouger.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: CreateEmpty()$
	$treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$
	$player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$
	voir valeur *cells* 3.4
	$envi \stackrel{\text{def}}{=} Environnement :: init(7, 6, player, guards, treasures, cells)$
	$C_1 \stackrel{\text{def}}{=} init(envi, 1, 1)$
Opérations	$C_2 \stackrel{\text{def}}{=} GoLeft(GoLeft(GoLeft(GoLeft(C_1))))$;
Oracle	$Wdt(C_2) = 0$ **and** $Hgt(C_2) = 1$.

Cas de test 4	Character::testGoLeft4
Objectif du test	S'assurer que des GoLeft face à un MTL à gauche du joueur ne le feront pas bouger.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: CreateEmpty()$
	$treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$
	$player \stackrel{\text{def}}{=} Player :: init(E_0, 3, 1)$
	voir valeur *cells* 3.4
	$envi \stackrel{\text{def}}{=} Environnement :: init(7, 6, player, guards, treasures, cells)$
	$C_1 \stackrel{\text{def}}{=} init(envi, 3, 1)$
	$SetNature(Envi(C_1), 1, 1, MTL)$
Opérations	$C_2 \stackrel{\text{def}}{=} GoLeft(GoLeft(GoLeft(GoLeft(C_1))))$;
Oracle	$Wdt(C_2) = 2$ **and** $Hgt(C_2) = 1$.

Cas de test 5	Character::testGoLeft5
Objectif du test	S'assurer que un GoLeft sur un HDR face à un mur fera remonter le joueur sur la plate-forme à gauche.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: CreateEmpty()$
	$treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$
	$player \stackrel{\text{def}}{=} Player :: init(E_0, 3, 1)$
	voir valeur *cells* 3.4
	$envi \stackrel{\text{def}}{=} Environnement :: init(7, 6, player, guards, treasures, cells)$

	$C_1 \stackrel{\text{def}}{=} init(envi, 4, 2)$
Opérations	$C_2 \stackrel{\text{def}}{=} GoLeft(C_1)$;
Oracle	$Wdt(C_2) = 3$ **and** $Hgt(C_2) = 3$.

Cas de test 6	Character::testGoLeft6
Objectif du test	S'assurer que si un Character est sur un HDR, etque deux autres HDR sont présents à gauche et en bas à gauche, alors un GoLeft fait rester le Character à la même hauteur.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: CreateEmpty()$ $treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 3, 1)$ voir valeur *cells* 3.4 $envi \stackrel{\text{def}}{=} Environnement ::$ $init(7, 6, player, guards, treasures, cells)$ $C_1 \stackrel{\text{def}}{=} init(envi, 1, 4)$
Opérations	$C_2 \stackrel{\text{def}}{=} GoLeft(C_1)$;
Oracle	$Wdt(C_2) = 0$ **and** $Hgt(C_2) = 4$.

Cas de test 7	Character::testGoLeft7
Objectif du test	S'assurer que le Character peut remonter par la gauche s'il est dans un HOL.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: CreateEmpty()$ $treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 3, 1)$ voir valeur *cells* 3.4 $E_1 \stackrel{\text{def}}{=} Environnement ::$ $init(7, 6, player, guards, treasures)$ $envi = SetNature(E_1, 1, 2, HOL)$ $C_1 \stackrel{\text{def}}{=} init(envi, 1, 2)$
Opérations	$C_2 \stackrel{\text{def}}{=} GoLeft(C_1)$;
Oracle	$Wdt(C_2) = 0$ **and** $Hgt(C_2) = 3$.

Description des tests du GoLeft du service Character dans un environnement vide

Objectif 3 : Tests de GoRight(C)

Cas de test 1	Character::testGoRight1
Objectif du test	S'assurer que le GoRight ne renvoie pas d'erreur.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: CreateEmpty()$ $treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$

	$player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.4 $envi \stackrel{\text{def}}{=} Environnement ::$ $init(7, 6, player, guards, treasures, cells)$ $C_1 \stackrel{\text{def}}{=} init(envi, 1, 1)$
Opérations	$GoRight(C_1)$;
Oracle	Pas d'exception levée.

Cas de test 2	Character::testGoRight2
Objectif du test	S'assurer que le personnage peut avancer jusqu'au bout du plateau.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: CreateEmpty()$ $treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.4 $envi \stackrel{\text{def}}{=} Environnement ::$ $init(7, 6, player, guards, treasures, cells)$ $C_1 \stackrel{\text{def}}{=} init(envi, 1, 1)$
Opérations	$C_2 \stackrel{\text{def}}{=}$ $GoRight(GoRight(GoRight(GoRight(GoRight(C_1)))));$
Oracle	$Wdt(C_2) = 6$ **and** $Hgt(C_2) = 1$.

Cas de test 3	Character::testGoRight3
Objectif du test	S'assurer que le personnage ne peut forcer le bout du plateau avec des GoRight.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: CreateEmpty()$ $treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.4 $envi \stackrel{\text{def}}{=} Environnement ::$ $init(7, 6, player, guards, treasures, cells)$ $C_1 \stackrel{\text{def}}{=} init(envi, 1, 1)$
Opérations	$C_2 \stackrel{\text{def}}{=}$ $GoRight(GoRight(GoRight(GoRight(GoRight(GoRight(GoRight(Go$
Oracle	$Wdt(C_2) = 6$ **and** $Hgt(C_2) = 1$.

Cas de test 4	Character::testGoRight4
Objectif du test	S'assurer que des GoRight face à un MTL à gauche du joueur ne le feront pas bouger.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: CreateEmpty()$ $treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 3, 1)$

	voir valeur *cells* 3.4
	$envi \stackrel{\text{def}}{=} Environnement :: init(7, 6, player, guards, treasures, cells)$
	$C_1 \stackrel{\text{def}}{=} init(envi, 3, 1)$
	$SetNature(Envi(C_1), 2, 1, MTL)$
Opérations	$C_2 \stackrel{\text{def}}{=} GoRight(GoRight(GoRight(GoRight(C_1))))$;
Oracle	$Wdt(C_2) = 1$ **and** $Hgt(C_2) = 1$.

Cas de test 5	Character::testGoRight5
Objectif du test	S'assurer que un GoRight sur un HDR face à un mur fera remonter le joueur sur la plate-forme à droite.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: CreateEmpty()$
	$treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$
	$player \stackrel{\text{def}}{=} Player :: init(E_0, 3, 1)$
	voir valeur *cells* 3.4
	$envi \stackrel{\text{def}}{=} Environnement :: init(7, 6, player, guards, treasures, cells)$
	$C_1 \stackrel{\text{def}}{=} init(envi, 5, 2)$
Opérations	$C_2 \stackrel{\text{def}}{=} GoRight(C_1)$;
Oracle	$Wdt(C_2) = 6$ **and** $Hgt(C_2) = 3$.

Cas de test 6	Character::testGoRight6
Objectif du test	S'assurer que un GoRight avec un HDR à droite fera se déplacer le joueur sur cet HDR.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: CreateEmpty()$
	$treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$
	$player \stackrel{\text{def}}{=} Player :: init(E_0, 3, 1)$
	voir valeur *cells* 3.4
	$envi \stackrel{\text{def}}{=} Environnement :: init(7, 6, player, guards, treasures, cells)$
	$C_1 \stackrel{\text{def}}{=} init(envi, 4, 2)$
Opérations	$C_2 \stackrel{\text{def}}{=} GoRight(C_1)$;
Oracle	$Wdt(C_2) = 5$ **and** $Hgt(C_2) = 2$.

Cas de test 7	Character::testGoRight7
Objectif du test	S'assurer que le Character peut remonter par la droite s'il est dans un HOL.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: CreateEmpty()$
	$treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$
	$player \stackrel{\text{def}}{=} Player :: init(E_0, 3, 1)$
	voir valeur *cells* 3.4

	$E_1 \stackrel{\text{def}}{=} Environnement :: init(7, 6, player, guards, treasures)$ $envi = SetNature(E_1, 1, 2, HOL)$ $C_1 \stackrel{\text{def}}{=} init(envi, 1, 2)$
Opérations	$C_2 \stackrel{\text{def}}{=} GoRight(C_1)$;
Oracle	$Wdt(C_2) = 2$ **and** $Hgt(C_2) = 3$.

Description des tests du GoRight du service Character dans un environnement vide

Objectif 4 : Tests de GoUp(C)

Cas de test 1	Character::testGoUp1
Objectif du test	S'assurer que le Character peut prendre une échelle.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: CreateEmpty()$ $treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 2, 1)$ voir valeur *cells* 3.4 $envi \stackrel{\text{def}}{=} Environnement :: init(7, 6, player, guards, treasures, cells)$ $C_1 \stackrel{\text{def}}{=} init(envi, 2, 1)$
Opérations	$C_2 \stackrel{\text{def}}{=} GoUp(C_1)$;
Oracle	$Hgt(C_2) = 2$

Cas de test 2	Character::testGoUp2
Objectif du test	S'assurer que le Character peut prendre une échelle et monter de trois cases.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: CreateEmpty()$ $treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 2, 1)$ voir valeur *cells* 3.4 $envi \stackrel{\text{def}}{=} Environnement :: init(7, 6, player, guards, treasures, cells)$ $C_1 \stackrel{\text{def}}{=} init(envi, 2, 1)$
Opérations	$C_2 \stackrel{\text{def}}{=} GoUp(GoUp(GoUp(C_1)))$;
Oracle	$Hgt(C_2) = 4$

Cas de test 3	Character::testGoUp3
Objectif du test	S'assurer que le Character ne monte pas s'il n'est pas sur une échelle.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: CreateEmpty()$ $treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$

	$player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$
	voir valeur *cells* 3.4
	$envi \stackrel{\text{def}}{=} Environnement ::$ $init(7, 6, player, guards, treasures, cells)$
	$C_1 \stackrel{\text{def}}{=} init(envi, 1, 1)$
Opérations	$C_2 \stackrel{\text{def}}{=} GoUp(C_1);$
Oracle	$Hgt(C_2) = 1$

Cas de test 4	Character::testGoUp4
Objectif du test	S'assurer que le Character ne peut pas monter de plus de quatre cases sur l'échelle (bordure du plateau).
Condition initiales	$guards \stackrel{\text{def}}{=} List :: CreateEmpty()$
	$treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$
	$player \stackrel{\text{def}}{=} Player :: init(E_0, 2, 1)$
	voir valeur *cells* 3.4
	$envi \stackrel{\text{def}}{=} Environnement ::$ $init(7, 6, player, guards, treasures, cells)$
	$C_1 \stackrel{\text{def}}{=} init(envi, 2, 1)$
Opérations	$C_2 \stackrel{\text{def}}{=} GoUp(GoUp(GoUp(GoUp(GoUp(C_1)))));$
Oracle	$Hgt(C_2) = 5$

Cas de test 5	Character::testGoUp5
Objectif du test	S'assurer que le Character ne peut pas monter s'il est dans un HOL.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: CreateEmpty()$
	$treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$
	$player \stackrel{\text{def}}{=} Player :: init(E_0, 2, 1)$
	voir valeur *cells* 3.4
	$envi \stackrel{\text{def}}{=} Environnement ::$ $init(7, 6, player, guards, treasures, cells)$
	$SetNature(Envi(C_1), 2, 1, HOL)$
	$C_1 \stackrel{\text{def}}{=} init(envi, 2, 1)$
Opérations	$C_2 \stackrel{\text{def}}{=} GoUp(C_1);$
Oracle	$Hgt(C_2) = 1$

Description des tests du GoUp du service Character dans un environnement vide

Objectif 5 : Tests de GoDown(C)

Cas de test 1	Character::testGoDown1

Objectif du test	S'assurer que le Character peut descendre en étant sur une échelle.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: CreateEmpty()$ $treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 2, 2)$ voir valeur *cells* 3.4 $envi \stackrel{\text{def}}{=} Environnement ::$ $init(7, 6, player, guards, treasures, cells)$ $C_1 \stackrel{\text{def}}{=} init(envi, 2, 2)$
Opérations	$C_2 \stackrel{\text{def}}{=} GoDown(C_1);$
Oracle	$Hgt(C_2) = 1$

Cas de test 2	Character::testGoDown2
Objectif du test	S'assurer que le Character peut descendre de trois cases.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: CreateEmpty()$ $treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 2, 4)$ voir valeur *cells* 3.4 $envi \stackrel{\text{def}}{=} Environnement ::$ $init(7, 6, player, guards, treasures, cells)$ $C_1 \stackrel{\text{def}}{=} init(envi, 2, 4)$
Opérations	$C_2 \stackrel{\text{def}}{=} GoDown(GoDown(GoDown(C_1)));$
Oracle	$Hgt(C_2) = 1$

Cas de test 3	Character::testGoDown3
Objectif du test	S'assurer que le Character ne descend pas si un MTL est en dessous.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: CreateEmpty()$ $treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.4 $envi \stackrel{\text{def}}{=} Environnement ::$ $init(7, 6, player, guards, treasures, cells)$ $C_1 \stackrel{\text{def}}{=} init(envi, 1, 1)$
Opérations	$C_2 \stackrel{\text{def}}{=} GoDown(C_1);$
Oracle	$Hgt(C_2) = 1$

Cas de test 4	Character::testGoDown4
Objectif du test	S'assurer que le Character ne peut pas traverser le sol s'il est sur une échelle.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: CreateEmpty()$

	$treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$
	$player \stackrel{\text{def}}{=} Player :: init(E_0, 2, 1)$
	voir valeur *cells* 3.4
	$envi \stackrel{\text{def}}{=} Environnement ::$ $init(7, 6, player, guards, treasures, cells)$
	$C_1 \stackrel{\text{def}}{=} init(envi, 2, 1)$
Opérations	$C_2 \stackrel{\text{def}}{=}$ $GoDown(GoDown(GoDown(GoDown(GoDown(C_1)))));$
Oracle	$Hgt(C_2) = 1$

Cas de test 5	Character::testGoDown5
Objectif du test	S'assurer que le Character peut descendre s'il est sur un HDR.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: CreateEmpty()$
	$treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$
	$player \stackrel{\text{def}}{=} Player :: init(E_0, 4, 2)$
	voir valeur *cells* 3.4
	$envi \stackrel{\text{def}}{=} Environnement ::$ $init(7, 6, player, guards, treasures, cells)$
	$C_1 \stackrel{\text{def}}{=} init(envi, 4, 2)$
Opérations	$C_2 \stackrel{\text{def}}{=} GoDown(C_1);$
Oracle	$Hgt(C_2) = 1$

Cas de test 6	Character::testGoDown6
Objectif du test	S'assurer que le Character peut tomber dans un HOL et ne peut pas continuer à tomber dans celui-ci.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: CreateEmpty()$
	$treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$
	$player \stackrel{\text{def}}{=} Player :: init(E_0, 2, 2)$
	voir valeur *cells* 3.4
	$envi \stackrel{\text{def}}{=} Environnement ::$ $init(7, 6, player, guards, treasures, cells)$
	$C_1 \stackrel{\text{def}}{=} init(envi, 2, 2)$
	$SetNature(Envi(C_1), 2, 1, HOL)$
Opérations	$C_2 \stackrel{\text{def}}{=} GoDown(GoDown(C_1));$
Oracle	$Hgt(C_2) = 1$

Description des tests du GoDown du service Character dans un environnement vide

Objectif 6 : Tests de BackInitialPosition(C)

Cas de test 1	Character::testBackInitialPositionPositif1
Objectif du test	S'assurer que le Character retourne bien à sa position initiale, même s'il est déjà présent sur celle-ci.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: CreateEmpty()$ $treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 2, 2)$ voir valeur *cells* 3.4 $envi \stackrel{\text{def}}{=} Environnement ::$ $init(7, 6, player, guards, treasures, cells)$ $C_1 \stackrel{\text{def}}{=} init(envi, 1, 1)$
Opérations	$C_2 \stackrel{\text{def}}{=} BackInitialPosition(C_1)$
Oracle	$Wdt(C_2) = 1$ **and** $Hgt(C_2) = 1$.

Cas de test 2	Character::testBackInitialPositionPositif2
Objectif du test	S'assurer que le Character retourne bien à sa position initiale après avoir bougé.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: CreateEmpty()$ $treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 2, 2)$ voir valeur *cells* 3.4 $envi \stackrel{\text{def}}{=} Environnement ::$ $init(7, 6, player, guards, treasures, cells)$ $C_1 \stackrel{\text{def}}{=} init(envi, 1, 1)$
Opérations	$C_2 \stackrel{\text{def}}{=}$ $BackInitialPosition(GoRight(GoUp(GoUp(GoRight(C_1)))))$
Oracle	$Wdt(C_2) = 1$ **and** $Hgt(C_2) = 1$.

Description des tests de BackInitialPosition du service Character

3.4.2 Test dans un environnement non-vide

Objectif 1 : Tests de init(environnement, x, y)

Cas de test 1	Character::testInitPositif1
Objectif du test	S'assurer que le init ne renvoie pas d'erreur.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: Create(Guard ::$ $init(E_0, 4, 1), Guard :: init(E_0, 0, 1))$ $treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.4

	$envi \stackrel{\text{def}}{=} Environnement :: init(7, 6, player, guards, treasures, cells)$
Opérations	$C_1 \stackrel{\text{def}}{=} init(envi, 1, 1)$
Oracle	Pas d'exception levée.

Cas de test 2	Character::testInitPositif2
Objectif du test	S'assurer que le init ne renvoie pas d'erreur lorsque l'on est à la borne supérieure du Screen.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: Create(Guard :: init(E_0, 4, 1), Guard :: init(E_0, 0, 1))$ $treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.4 $envi \stackrel{\text{def}}{=} Environnement :: init(7, 6, player, guards, treasures, cells)$
Opérations	$C_1 \stackrel{\text{def}}{=} init(envi, 6, 5)$
Oracle	Pas d'exception levée.

Cas de test 3	Character::testInitNegatif1
Objectif du test	S'assurer que le init renvoie une erreur si on initialise le personnage avec un x négatif.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: Create(Guard :: init(E_0, 4, 1), Guard :: init(E_0, 0, 1))$ $treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.4 $envi \stackrel{\text{def}}{=} Environnement :: init(7, 6, player, guards, treasures, cells)$
Opérations	$C_1 \stackrel{\text{def}}{=} init(envi, -1, 0)$
Oracle	Exception levée.

Cas de test 4	Character::testInitNegatif2
Objectif du test	S'assurer que le init renvoie une erreur si on initialise le personnage avec un y négatif.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: Create(Guard :: init(E_0, 4, 1), Guard :: init(E_0, 0, 1))$ $treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.4 $envi \stackrel{\text{def}}{=} Environnement :: init(7, 6, player, guards, treasures, cells)$
Opérations	$C_1 \stackrel{\text{def}}{=} init(envi, 0, -1)$
Oracle	Exception levée.

Cas de test 5	Character::testInitNegatif3
Objectif du test	S'assurer que le init renvoie une erreur si on initialise le personnage avec un x et un y négatifs.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: Create(Guard :: init(E_0, 4, 1), Guard :: init(E_0, 0, 1))$ $treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.4 $envi \stackrel{\text{def}}{=} Environnement :: init(7, 6, player, guards, treasures, cells)$
Opérations	$C_1 \stackrel{\text{def}}{=} init(envi, -1, -1)$
Oracle	Exception levée.

Cas de test 6	Character::testInitNegatif4
Objectif du test	S'assurer que le init renvoie une erreur si on initialise le personnage avec un x égal à la largeur du Screen.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: Create(Guard :: init(E_0, 4, 1), Guard :: init(E_0, 0, 1))$ $treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.4 $envi \stackrel{\text{def}}{=} Environnement :: init(7, 6, player, guards, treasures, cells)$
Opérations	$C_1 \stackrel{\text{def}}{=} init(envi, 7, 0)$
Oracle	Exception levée.

Cas de test 7	Character::testInitNegatif5
Objectif du test	S'assurer que le init renvoie une erreur si on initialise le personnage avec un y égal à la hauteur du Screen.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: Create(Guard :: init(E_0, 4, 1), Guard :: init(E_0, 0, 1))$ $treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.4 $envi \stackrel{\text{def}}{=} Environnement :: init(7, 6, player, guards, treasures, cells)$
Opérations	$C_1 \stackrel{\text{def}}{=} init(envi, 0, 6)$
Oracle	Exception levée.

Cas de test 8	Character::testInitNegatif6

Objectif du test	S'assurer que le init renvoie une erreur si on initialise le personnage avec x égal à la largeur du Screen, et un y égal à la hauteur du Screen.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: Create(Guard :: init(E_0, 4, 1), Guard :: init(E_0, 0, 1))$ $treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.4 $envi \stackrel{\text{def}}{=} Environnement :: init(7, 6, player, guards, treasures, cells)$
Opérations	$C_1 \stackrel{\text{def}}{=} init(envi, 7, 6)$
Oracle	Exception levée.

Cas de test 9	Character::testInitNegatif7
Objectif du test	S'assurer que le init renvoie une erreur si on initialise le personnage dans une case différente d'un EMP.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: Create(Guard :: init(E_0, 4, 1), Guard :: init(E_0, 0, 1))$ $treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.4 $envi \stackrel{\text{def}}{=} Environnement :: init(7, 6, player, guards, treasures, cells)$
Opérations	$C_1 \stackrel{\text{def}}{=} init(envi, 0, 0)$
Oracle	Exception levée.

Description des tests du Init du service Character dans un environnement non-vide

Objectif 2 : Tests de GoLeft(C)

Cas de test 1	Character::testGoLeft1
Objectif du test	S'assurer que le GoLeft ne renvoie pas d'erreur.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: Create(Guard :: init(E_0, 4, 1), Guard :: init(E_0, 0, 1))$ $treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.4 $envi \stackrel{\text{def}}{=} Environnement :: init(7, 6, player, guards, treasures, cells)$ $C_1 \stackrel{\text{def}}{=} init(envi, 1, 1)$
Opérations	$GoLeft(C_1)$;

Oracle	Pas d'exception levée.

Cas de test 2	Character::testGoLeft2
Objectif du test	S'assurer que les deux GoLeft feront bouger le personnage de deux cases vers la gauche.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: Create(Guard :: init(E_0, 4, 1), Guard :: init(E_0, 0, 1))$ $treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.4 $envi \stackrel{\text{def}}{=} Environnement :: init(7, 6, player, guards, treasures, cells)$ $C_1 \stackrel{\text{def}}{=} GoRight(GoRight(init(envi, 1, 1)))$
Opérations	$C_2 \stackrel{\text{def}}{=} GoLeft(GoLeft(C_1))$;
Oracle	$Wdt(C_2) = 1$ **and** $Hgt(C_2) = 1$.

Cas de test 3	Character::testGoLeft3
Objectif du test	S'assurer que des GoLeft face à un mur à gauche du joueur ne le feront pas bouger.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: CreateEmpty()$ $treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.4 $envi \stackrel{\text{def}}{=} Environnement :: init(7, 6, player, guards, treasures, cells)$ $C_1 \stackrel{\text{def}}{=} init(envi, 1, 1)$
Opérations	$C_2 \stackrel{\text{def}}{=} GoLeft(GoLeft(GoLeft(GoLeft(C_1))))$;
Oracle	$Wdt(C_2) = 0$ **and** $Hgt(C_2) = 1$.

Cas de test 4	Character::testGoLeft4
Objectif du test	S'assurer que des GoLeft face à un MTL à gauche du joueur ne le feront pas bouger.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: Create(Guard :: init(E_0, 4, 1), Guard :: init(E_0, 0, 1))$ $treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 3, 1)$ voir valeur *cells* 3.4 $envi \stackrel{\text{def}}{=} Environnement :: init(7, 6, player, guards, treasures, cells)$ $C_1 \stackrel{\text{def}}{=} init(envi, 3, 1)$ $SetNature(Envi(C_1), 1, 1, MTL)$
Opérations	$C_2 \stackrel{\text{def}}{=} GoLeft(GoLeft(GoLeft(GoLeft(C_1))))$;

Oracle	$Wdt(C_2) = 2$ **and** $Hgt(C_2) = 1$.

Cas de test 5	Character::testGoLeft5
Objectif du test	S'assurer le joueur sera bloqué par un garde présent sur la plate-forme à la sortie de l'HDR.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: Create(Guard :: init(E_0, 4, 2), Guard :: init(E_0, 0, 1))$ $treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 4, 2)$ voir valeur *cells* 3.4 $envi \stackrel{\text{def}}{=} Environnement :: init(7, 6, player, guards, treasures, cells)$ $C_1 \stackrel{\text{def}}{=} init(envi, 4, 2)$
Opérations	$C_2 \stackrel{\text{def}}{=} GoLeft(GoLeft(GoLeft(C_1)))$;
Oracle	$Wdt(C_2) = 4$ **and** $Hgt(C_2) = 2$.

Description des tests du GoLeft du service Character dans un environnement non-vide

Objectif 3 : Tests de GoRight(C)

Cas de test 1	Character::testGoRight1
Objectif du test	S'assurer que le GoRight ne renvoie pas d'erreur.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: Create(Guard :: init(E_0, 4, 1), Guard :: init(E_0, 0, 1))$ $treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.4 $envi \stackrel{\text{def}}{=} Environnement :: init(7, 6, player, guards, treasures, cells)$ $C_1 \stackrel{\text{def}}{=} init(envi, 1, 1)$
Opérations	$GoRight(C_1)$;
Oracle	Pas d'exception levée.

Cas de test 2	Character::testGoRight2
Objectif du test	S'assurer que le Character peut avancer jusqu'au garde.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: Create(Guard :: init(E_0, 4, 1), Guard :: init(E_0, 0, 1))$ $treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.4

	$envi \stackrel{\text{def}}{=} Environnement ::$ $init(7, 6, player, guards, treasures, cells)$ $C_1 \stackrel{\text{def}}{=} init(envi, 1, 1))$
Opérations	$C_2 \stackrel{\text{def}}{=} GoRight(GoRight(C_1))$;
Oracle	$Wdt(C_2) = 3$ **and** $Hgt(C_2) = 1$.

Cas de test 3	Character::testGoRight3
Objectif du test	S'assurer que le Character ne peut traverser le garde à sa droite avancer jusqu'au garde.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: Create(Guard ::$ $init(E_0, 4, 1), Guard :: init(E_0, 0, 1))$ $treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.4 $envi \stackrel{\text{def}}{=} Environnement ::$ $init(7, 6, player, guards, treasures, cells)$ $C_1 \stackrel{\text{def}}{=} init(envi, 1, 1))$
Opérations	$C_2 \stackrel{\text{def}}{=}$ $GoRight(GoRight(GoRight(GoRight(GoRight(C_1)))))$;
Oracle	$Wdt(C_2) = 3$ **and** $Hgt(C_2) = 1$.

Cas de test 4	Character::testGoRight4
Objectif du test	S'assurer que des GoRight face à un MTL à gauche du joueur ne le feront pas bouger.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: Create(Guard ::$ $init(E_0, 4, 1), Guard :: init(E_0, 0, 1))$ $treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.4 $envi \stackrel{\text{def}}{=} Environnement ::$ $init(7, 6, player, guards, treasures, cells)$ $C_1 \stackrel{\text{def}}{=} init(envi, 1, 1)$ $SetNature(Envi(C_1), 2, 1, MTL)$
Opérations	$C_2 \stackrel{\text{def}}{=} GoRight(GoRight(GoRight(GoRight(C_1))))$;
Oracle	$Wdt(C_2) = 1$ **and** $Hgt(C_2) = 1$.

Cas de test 5	Character::testGoRight5
Objectif du test	S'assurer le joueur sera bloqué par un garde présent sur le HDR à sa droite.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: Create(Guard ::$ $init(E_0, 5, 2), Guard :: init(E_0, 0, 1))$ $treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 4, 2)$

	voir valeur *cells* 3.4 $envi \stackrel{\text{def}}{=} Environnement ::$ $init(7, 6, player, guards, treasures, cells)$ $C_1 \stackrel{\text{def}}{=} init(envi, 4, 2)$
Opérations	$C_2 \stackrel{\text{def}}{=} GoRight(GoRight(GoRight(C_1)))$;
Oracle	$Wdt(C_2) = 4$ **and** $Hgt(C_2) = 2$.

Description des tests du GoRight du service Character dans un environnement non-vide

Objectif 4 : Tests de GoUp(C)

Cas de test 1	Character::testGoUp1
Objectif du test	S'assurer que le Character ne peut pas prendre une échelle si un garde est déjà sur la case au-dessus du Character.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: Create(Guard ::$ $init(E_0, 2, 2), Guard :: init(E_0, 0, 1))$ $treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 2, 1)$ voir valeur *cells* 3.4 $envi \stackrel{\text{def}}{=} Environnement ::$ $init(7, 6, player, guards, treasures, cells)$ $C_1 \stackrel{\text{def}}{=} init(envi, 2, 1)$
Opérations	$C_2 \stackrel{\text{def}}{=} GoUp(C_1)$;
Oracle	$Hgt(C_2) = 1$

Cas de test 2	Character::testGoUp2
Objectif du test	S'assurer que le Character ne peut pas prendre une échelle si un garde est déjà sur la case au-dessus du Character. Ici, le Character peut quand même monter d'une case
Condition initiales	$guards \stackrel{\text{def}}{=} List :: Create(Guard ::$ $init(E_0, 2, 3), Guard :: init(E_0, 0, 1))$ $treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 2, 1)$ voir valeur *cells* 3.4 $envi \stackrel{\text{def}}{=} Environnement ::$ $init(7, 6, player, guards, treasures, cells)$ $C_1 \stackrel{\text{def}}{=} init(envi, 2, 1)$
Opérations	$C_2 \stackrel{\text{def}}{=} GoUp(GoUp(C_1))$;
Oracle	$Hgt(C_2) = 2$

Cas de test 3	Character::testGoUp3

Objectif du test	S'assurer que le Character ne monte pas s'il n'est pas sur une échelle.
Condition initiales	$guards \overset{\text{def}}{=} List :: Create(Guard :: init(E_0, 4, 1), Guard :: init(E_0, 0, 1))$ $treasures \overset{\text{def}}{=} List :: CreateEmpty()$ $player \overset{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.4 $envi \overset{\text{def}}{=} Environnement :: init(7, 6, player, guards, treasures, cells)$ $C_1 \overset{\text{def}}{=} init(envi, 1, 1)$
Opérations	$C_2 \overset{\text{def}}{=} GoUp(C_1);$
Oracle	$Hgt(C_2) = 1$

Cas de test 4	Character::testGoUp4
Objectif du test	S'assurer que le Character ne peut pas monter de plus de quatre cases sur l'échelle (bordure du plateau).
Condition initiales	$guards \overset{\text{def}}{=} List :: Create(Guard :: init(E_0, 4, 1), Guard :: init(E_0, 0, 1))$ $treasures \overset{\text{def}}{=} List :: CreateEmpty()$ $player \overset{\text{def}}{=} Player :: init(E_0, 2, 1)$ voir valeur *cells* 3.4 $envi \overset{\text{def}}{=} Environnement :: init(7, 6, player, guards, treasures, cells)$ $C_1 \overset{\text{def}}{=} init(envi, 2, 1)$
Opérations	$C_2 \overset{\text{def}}{=} GoUp(GoUp(GoUp(GoUp(GoUp(C_1)))));$
Oracle	$Hgt(C_2) = 5$

Description des tests du GoUp du service Character dans un environnement non-vide

Objectif 5 : Tests de GoDown(C)

Cas de test 1	Character::testGoDown1
Objectif du test	S'assurer que le Character ne peut pas descendre si un garde le bloque.
Condition initiales	$guards \overset{\text{def}}{=} List :: Create(Guard :: init(E_0, 2, 1), Guard :: init(E_0, 0, 1))$ $treasures \overset{\text{def}}{=} List :: CreateEmpty()$ $player \overset{\text{def}}{=} Player :: init(E_0, 2, 2)$ voir valeur *cells* 3.4 $envi \overset{\text{def}}{=} Environnement :: init(7, 6, player, guards, treasures, cells)$

	$C_1 \stackrel{\text{def}}{=} init(envi, 2, 2)$
Opérations	$C_2 \stackrel{\text{def}}{=} GoDown(C_1);$
Oracle	$Hgt(C_2) = 2$

Cas de test 2	Character::testGoDown2
Objectif du test	S'assurer que le Character ne peut pas descendre si un garde le bloque. Ici, le joueur va essayer de forcer.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: Create(Guard :: init(E_0, 2, 1), Guard :: init(E_0, 0, 1))$ $treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 2, 2)$ voir valeur *cells* 3.4 $envi \stackrel{\text{def}}{=} Environnement :: init(7, 6, player, guards, treasures, cells)$ $C_1 \stackrel{\text{def}}{=} init(envi, 2, 2)$
Opérations	$C_2 \stackrel{\text{def}}{=} GoDown(GoDown(GoDown(C_1)));$
Oracle	$Hgt(C_2) = 2$

Cas de test 3	Character::testGoDown3
Objectif du test	S'assurer que le Character ne descend pas si un MTL est en dessous.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: Create(Guard :: init(E_0, 2, 2), Guard :: init(E_0, 0, 1))$ $treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.4 $envi \stackrel{\text{def}}{=} Environnement :: init(7, 6, player, guards, treasures, cells)$ $C_1 \stackrel{\text{def}}{=} init(envi, 1, 1)$
Opérations	$C_2 \stackrel{\text{def}}{=} GoDown(C_1);$
Oracle	$Hgt(C_2) = 1$

Cas de test 4	Character::testGoDown4
Objectif du test	S'assurer que le Character ne peut pas traverser le sol s'il est sur une échelle.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: Create(Guard :: init(E_0, 2, 2), Guard :: init(E_0, 0, 1))$ $treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 2, 1)$ voir valeur *cells* 3.4 $envi \stackrel{\text{def}}{=} Environnement :: init(7, 6, player, guards, treasures, cells)$

Opérations	$C_1 \overset{\text{def}}{=} init(envi, 2, 1)$ $C_2 \overset{\text{def}}{=}$ $GoDown(GoDown(GoDown(GoDown(GoDown(C_1)))));$
Oracle	$Hgt(C_2) = 1$

Cas de test 5	Character::testGoDown5
Objectif du test	S'assurer que le Character ne peut pas descendre s'il d' un HDR si un garde est en dessous.
Condition initiales	$guards \overset{\text{def}}{=} List :: Create(Guard ::$ $init(E_0, 4, 1), Guard :: init(E_0, 0, 1))$ $treasures \overset{\text{def}}{=} List :: CreateEmpty()$ $player \overset{\text{def}}{=} Player :: init(E_0, 4, 2)$ voir valeur *cells* 3.4 $envi \overset{\text{def}}{=} Environnement ::$ $init(7, 6, player, guards, treasures, cells)$ $C_1 \overset{\text{def}}{=} init(envi, 4, 2)$
Opérations	$C_2 \overset{\text{def}}{=} GoDown(C_1);$
Oracle	$Hgt(C_2) = 2$

Description des tests du GoDown du service Character dans un environnement non-vide

3.5 Player

L'ensemble des tests du Service Player se déroule dans l'environnement indiqué par la figure 11.

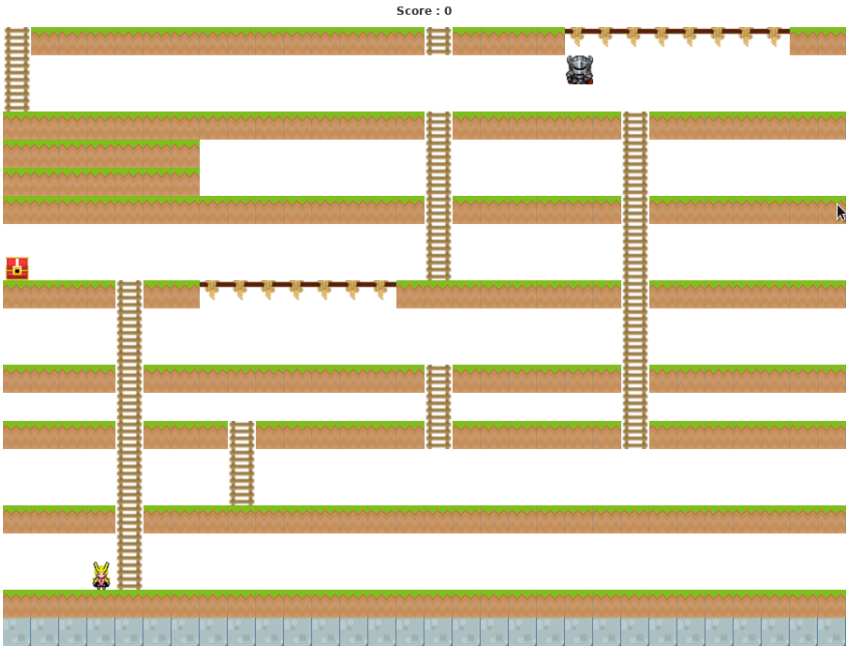

FIGURE 11 – Terrain des tests de Player.

Objectif 1 : Test de Init(envi, playerX, playerY, engine)

Cas de test 1	Player::testInitPositif
Objectif du test	S'assurer que le init ne renvoie pas d'erreur.
Condition initiales	$engine \stackrel{\text{def}}{=} Engine :: init(30, 23, 3, 2)$
Opérations	$P_1 \stackrel{\text{def}}{=} init(Engine :: CurrentEnvironnement(engine), 3, 2, engine)$
Oracle	Pas d'exception levée

Description du test du Init du service Player

Objectif 2 : Test de DecreaseVie(P)

Cas de test 1	Player::testDecreaseViePositif
Objectif du test	S'assurer que le DecreaseVie diminue bien la vie du Player.
Condition initiales	$engine \stackrel{\text{def}}{=} Engine :: init(30, 23, 3, 2)$ $P_1 \stackrel{\text{def}}{=} init(Engine::CurrentEnvironnement(engine), 3, 2, engine)$
Opérations	$P_2 \stackrel{\text{def}}{=} DecreaseVie(P_1)$
Oracle	$Vie(P_2) = 2$

Description du test du DecreaseVie du service Player

Objectif 3 : Test de Step(P)

Cas de test 1	Player::testStepPositif1
Objectif du test	S'assurer que si un Player est au dessus du vide, il tombe.
Condition initiales	$engine \stackrel{\text{def}}{=} Engine :: init(30, 23, 3, 3)$ $P_1 \stackrel{\text{def}}{=} init(Engine::CurrentEnvironnement(engine), 3, 3, engine)$
Opérations	$P_2 \stackrel{\text{def}}{=} Step(P_1)$
Oracle	$Character :: LastAction(P_2) = DOWN$

Cas de test 2	Player::testStepPositif2
Objectif du test	S'assurer que si l'engine demande au Player de faire une commande DOWN, alors il doit l'exécuter.
Condition initiales	$engine \stackrel{\text{def}}{=} Engine :: SetNextCommand(Engine :: init(30, 23, 3, 2), DOWN)$ $P_1 \stackrel{\text{def}}{=} init(Engine::CurrentEnvironnement(engine), 3, 2, engine)$
Opérations	$P_2 \stackrel{\text{def}}{=} Step(P_1)$

Oracle	$Character :: LastAction(P_2) = DOWN$

Cas de test 3	Player::testStepPositif3
Objectif du test	S'assurer que si l'engine demande au Player de faire une commande UP, alors il doit l'exécuter.
Condition initiales	$engine \stackrel{\text{def}}{=} Engine :: SetNextCommand(Engine :: init(30, 23, 3, 2), UP)$ $P_1 \stackrel{\text{def}}{=} init(Engine::CurrentEnvironnement(engine), 3, 2, engine)$
Opérations	$P_2 \stackrel{\text{def}}{=} Step(P_1)$
Oracle	$Character :: LastAction(P_2) = UP$

Cas de test 4	Player::testStepPositif4
Objectif du test	S'assurer que si l'engine demande au Player de faire une commande LEFT, alors il doit l'exécuter.
Condition initiales	$engine \stackrel{\text{def}}{=} Engine :: SetNextCommand(Engine :: init(30, 23, 3, 2), LEFT)$ $P_1 \stackrel{\text{def}}{=} init(Engine::CurrentEnvironnement(engine), 3, 2, engine)$
Opérations	$P_2 \stackrel{\text{def}}{=} Step(P_1)$
Oracle	$Character :: LastAction(P_2) = LEFT$

Cas de test 5	Player::testStepPositif5
Objectif du test	S'assurer que si l'engine demande au Player de faire une commande RIGHT, alors il doit l'exécuter.
Condition initiales	$engine \stackrel{\text{def}}{=} Engine :: SetNextCommand(Engine :: init(30, 23, 3, 2), RIGHT)$ $P_1 \stackrel{\text{def}}{=} init(Engine::CurrentEnvironnement(engine), 3, 2, engine)$
Opérations	$P_2 \stackrel{\text{def}}{=} Step(P_1)$
Oracle	$Character :: LastAction(P_2) = RIGHT$

Cas de test 6	Player::testStepPositif6
Objectif du test	S'assurer que si l'engine demande au Player de faire une commande DIGL, alors il doit l'exécuter.
Condition initiales	$engine \stackrel{\text{def}}{=} Engine :: SetNextCommand(Engine :: init(30, 23, 3, 2), DIGL)$ $P_1 \stackrel{\text{def}}{=} init(Engine::CurrentEnvironnement(engine), 3, 2, engine)$
Opérations	$P_2 \stackrel{\text{def}}{=} Step(P_1)$
Oracle	$Character :: LastAction(P_2) = DIGL$

Cas de test 7	Player::testStepPositif7
Objectif du test	S'assurer que si l'engine demande au Player de faire une commande DIGR, alors il doit l'exécuter.

Condition initiales	$engine \stackrel{\text{def}}{=} Engine :: SetNextCommand(Engine :: init(30, 23, 3, 2), DIGR)$ $P_1 \stackrel{\text{def}}{=} init(Engine::CurrentEnvironnement(engine), 3, 2, engine)$
Opérations	$P_2 \stackrel{\text{def}}{=} Step(P_1)$
Oracle	$Character :: LastAction(P_2) = DIGR$

Cas de test 8	Player::testStepPositif8
Objectif du test	S'assurer que si le joueur creuse un trou et va au dessus du trou, il tombe.
Condition initiales	$engine \stackrel{\text{def}}{=} Engine :: SetNextCommand(Engine :: init(30, 23, 3, 2), DIGL)$ $P_1 \stackrel{\text{def}}{=} init(Engine::CurrentEnvironnement(engine), 3, 2, engine)$
Opérations	$P_2 \stackrel{\text{def}}{=} Step(P_1)$ $engine \stackrel{\text{def}}{=} Engine :: SetNextCommand(engine, LEFT)$ $P_3 \stackrel{\text{def}}{=} Step(P_2)$ $P_4 \stackrel{\text{def}}{=} Step(P_3)$
Oracle	$Character :: LastAction(P_4) = DOWN$

Description du test du Step du service Player

3.6 Guard

Pour les tests du service Guard, le terrain est de la forme suivante :

$$cells \stackrel{\text{def}}{=} \begin{bmatrix} EMP & EMP & LAD & EMP & EMP & EMP & EMP & EMP & EMP \\ HDR & HDR & LAD & HDR & HDR & EMP & EMP & EMP & EMP \\ HOL & EMP & LAD & EMP & HDR & EMP & EMP & EMP & EMP \\ PLT & PLT & LAD & PLT & HDR & HDR & PLT & HOL & LAD \\ EMP & EMP & LAD & EMP & EMP & EMP & EMP & EMP & EMP \\ MLT & MLT & MLT & MLT & MLT & MLT & MLT & MTL & MTL \end{bmatrix}$$

Dans l'ensemble de ces tests, la variable R_0 représente un Resolver avant l'appel à *init*. E_0 et G_0 représente quant à eux respectivement un environnement et un garde avant leurs appels respectifs à *init*.

Objectif 1 : Test de init(environnement, x, y, id, resolver)

Cas de test 1	Guard::testInitPositif
Objectif du test	S'assurer que le init ne renvoie pas d'erreur.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: Create(G_0)$ $treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.6

	$envi \stackrel{\text{def}}{=} Environnement ::$ $init(9, 6, player, guards, treasures, cells)$ $resolver \stackrel{\text{def}}{=} Resolver :: init(envi, player);$
Opérations	$G_1 \stackrel{\text{def}}{=} init(envi, 1, 3, 1, resolver)$
Oracle	Pas d'exception levée.

Description du test de Init du service Guard

Objectif 2 : Tests de ClimbLeft(G)

Cas de test 1	Guard::testClimbLeftPositif1
Objectif du test	S'assurer que le ClimbLeft depuis un trou ne renvoie pas d'erreur.
Condition initiales	$G_1 \stackrel{\text{def}}{=} init(E_0, 0, 3, 1, R_0)$ $guards \stackrel{\text{def}}{=} List :: Create(G_1)$ $treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.6 $envi \stackrel{\text{def}}{=} Environnement ::$ $init(9, 6, player, guards, treasures, cells)$ $resolver \stackrel{\text{def}}{=} Resolver :: init(envi, player);$
Opérations	$ClimbLeft(G_1)$
Oracle	Pas d'exception levée.

Cas de test 2	Guard::testClimbLeftPositif2
Objectif du test	S'assurer que le ClimbLeft depuis un trou fait sortir le garde du trou.
Condition initiales	$G_1 \stackrel{\text{def}}{=} init(E_0, 7, 2, 1, R_0)$ $guards \stackrel{\text{def}}{=} List :: Create(G_1)$ $treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.6 $envi \stackrel{\text{def}}{=} Environnement ::$ $init(9, 6, player, guards, treasures, cells)$ $resolver \stackrel{\text{def}}{=} Resolver :: init(envi, player);$
Opérations	$G_2 \stackrel{\text{def}}{=} ClimbLeft(G_1)$
Oracle	$Character :: Wdt(G_2) = 6$ **and** $Character :: Hgt(G_2) = 3$

Cas de test 3	Guard::testClimbLeftNegatif

Objectif du test	S'assurer que le ClimbLeft d'un garde pas dans un trou renvoie une erreur.
Condition initiales	$G_1 \stackrel{\text{def}}{=} init(E_0, 1, 3, 1, R_0)$ $guards \stackrel{\text{def}}{=} List :: Create(G_1)$ $treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.6 $envi \stackrel{\text{def}}{=} Environnement ::$ $init(9, 6, player, guards, treasures, cells)$ $resolver \stackrel{\text{def}}{=} Resolver :: init(envi, player);$
Opérations	$ClimbLeft(G_1)$
Oracle	Exception levée.

Description des tests de ClimbLeft du service Guard

Objectif 3 : Tests de ClimbRight(G)

Cas de test 1	Guard::testClimbRightPositif1
Objectif du test	S'assurer que le ClimbRight depuis un trou ne renvoie pas d'erreur.
Condition initiales	$G_1 \stackrel{\text{def}}{=} init(E_0, 0, 3, 1, R_0)$ $guards \stackrel{\text{def}}{=} List :: Create(G_1)$ $treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.6 $envi \stackrel{\text{def}}{=} Environnement ::$ $init(9, 6, player, guards, treasures, cells)$ $resolver \stackrel{\text{def}}{=} Resolver :: init(envi, player);$
Opérations	$ClimbRight(G_1)$
Oracle	Pas d'exception levée.

Cas de test 2	Guard::testClimbRightPositif2
Objectif du test	S'assurer que le ClimbLeft depuis un trou fait sortir le garde du trou.
Condition initiales	$G_1 \stackrel{\text{def}}{=} init(E_0, 7, 2, 1, R_0)$ $guards \stackrel{\text{def}}{=} List :: Create(G_1)$ $treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.6

	$envi \stackrel{\text{def}}{=} Environnement ::$ $init(9, 6, player, guards, treasures, cells)$ $resolver \stackrel{\text{def}}{=} Resolver :: init(envi, player);$
Opérations	$G_2 \stackrel{\text{def}}{=} ClimbRight(G_1)$
Oracle	$Character :: Wdt(G_2) = 8$ **and** $Character :: Hgt(G_2) = 3$

Cas de test 3	Guard::testClimbRightNegatif
Objectif du test	S'assurer que le ClimbRight d'un garde pas dans un trou renvoie une erreur.
Condition initiales	$G_1 \stackrel{\text{def}}{=} init(E_0, 1, 3, 1, R_0)$ $guards \stackrel{\text{def}}{=} List :: Create(G_1)$ $treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.6 $envi \stackrel{\text{def}}{=} Environnement ::$ $init(9, 6, player, guards, treasures, cells)$ $resolver \stackrel{\text{def}}{=} Resolver :: init(envi, player);$
Opérations	$ClimbRight(G_1)$
Oracle	Exception levée.

Description des tests de ClimbRight du service Guard

Objectif 4 : Tests de Step(G)

Cas de test 1	Guard::testStepPositif1
Objectif du test	S'assurer qu'un step au dessus d'un EMP fait tomber le garde.
Condition initiales	$G_1 \stackrel{\text{def}}{=} init(E_0, 8, 5, 1, R_0)$ $guards \stackrel{\text{def}}{=} List :: Create(G_1)$ $treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.6 $envi \stackrel{\text{def}}{=} Environnement ::$ $init(9, 6, player, guards, treasures, cells)$ $resolver \stackrel{\text{def}}{=} Resolver :: init(envi, player);$
Opérations	$G_2 \stackrel{\text{def}}{=} Step(G_1)$
Oracle	$Character :: Wdt(G_2) = 8$ **and** $Character :: Hgt(G_2) = 4.$

Cas de test 2	Guard::testStepPositif2
Objectif du test	S'assurer qu'un step au dessus d'un HOL fait tomber le garde.
Condition initiales	$G_1 \stackrel{\text{def}}{=} init(E_0, 7, 3, 1, R_0)$ $guards \stackrel{\text{def}}{=} List :: Create(G_1)$ $treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.6 $envi \stackrel{\text{def}}{=} Environnement ::$ $init(9, 6, player, guards, treasures, cells)$ $resolver \stackrel{\text{def}}{=} Resolver :: init(envi, player);$
Opérations	$G_2 \stackrel{\text{def}}{=} Step(G_1)$
Oracle	$Character :: Wdt(G_2) = 7$ **and** $Character :: Hgt(G_2) = 2.$

Cas de test 3	Guard::testStepPositif3
Objectif du test	S'assurer qu'un step au dans un HOL augmente le TimeInHole du garde.
Condition initiales	$G_1 \stackrel{\text{def}}{=} init(E_0, 7, 2, 1, R_0)$ $guards \stackrel{\text{def}}{=} List :: Create(G_1)$ $treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.6 $envi \stackrel{\text{def}}{=} Environnement ::$ $init(9, 6, player, guards, treasures, cells)$ $resolver \stackrel{\text{def}}{=} Resolver :: init(envi, player);$
Opérations	$G_2 \stackrel{\text{def}}{=} Step(G_1)$
Oracle	$TimeInHole(G_2) = 1.$

Cas de test 4	Guard::testStepPositif4
Objectif du test	S'assurer qu'après deux Step dans un HOL le TimeInHole est incrémenté correctement, et le garde sort du HOL (au troisième Step).
Condition initiales	$G_1 \stackrel{\text{def}}{=} init(E_0, 7, 2, 1, R_0)$ $guards \stackrel{\text{def}}{=} List :: Create(G_1)$ $treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.6

	$envi \overset{\text{def}}{=} Environnement ::$ $init(9, 6, player, guards, treasures, cells)$ $resolver \overset{\text{def}}{=} Resolver :: init(envi, player);$
Opérations	$G_2 \overset{\text{def}}{=} Step(Step(G_1))$ $G_3 \overset{\text{def}}{=} Step(G_2)$
Oracle	$TimeInHole(G_2) = 2$ **and** $Character :: Wdt(G_3) = 6$ **and** $Character :: Hgt(G_3) = 3.$

Cas de test 5	Guard::testStepPositif5
Objectif du test	S'assurer que si le garde est déjà sur le *player*, il effectue une commande NEUTRAL et donc ne bouge pas.
Condition initiales	$G_1 \overset{\text{def}}{=} init(E_0, 1, 1, 1, R_0)$ $guards \overset{\text{def}}{=} List :: Create(G_1)$ $treasures \overset{\text{def}}{=} List :: CreateEmpty()$ $player \overset{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.6
	$envi \overset{\text{def}}{=} Environnement ::$ $init(9, 6, player, guards, treasures, cells)$ $resolver \overset{\text{def}}{=} Resolver :: init(envi, player);$
Opérations	$G_2 \overset{\text{def}}{=} Step(G_1)$
Oracle	$LastAction(G_2) = NEUTRAL$ **and** $Character :: Wdt(G_3) = 1$ **and** $Character :: Hgt(G_3) = 1.$

Description des tests de Step du service Guard

3.7 PathResolver

Pour les tests du service Guard, le terrain est de la forme suivante :

$$cells \overset{\text{def}}{=} \begin{bmatrix} EMP & EMP & LAD & EMP & EMP & EMP & EMP & PLT & EMP \\ HDR & HDR & LAD & HDR & HDR & EMP & EMP & PLT & PLT \\ HOL & EMP & LAD & EMP & HDR & EMP & EMP & EMP & EMP \\ PLT & PLT & LAD & PLT & HDR & HDR & PLT & HOL & LAD \\ EMP & EMP & LAD & EMP & EMP & EMP & EMP & EMP & EMP \\ MLT & MLT & MLT & MLT & MLT & MLT & MLT & MTL & MTL \end{bmatrix}$$

Dans l'ensemble de ces tests, la variable E_0 représente un environnement avant son appel à *init*.

Objectif 1 : Test de init(environnement, player)

Cas de test 1	PathResolver::testInitPositif
Objectif du test	S'assurer que le init ne renvoie pas d'erreur.

Condition initiales	$guards \stackrel{\text{def}}{=} List :: Create(Guard :: init(E_0, 4, 1), Guard :: init(E_0, 1, 3))$ $treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.7 $envi \stackrel{\text{def}}{=} Environnement :: init(9, 6, player, guards, treasures, cells)$
Opérations	$R_1 \stackrel{\text{def}}{=} init(envi, player)$
Oracle	Pas d'exception levée.

Description du test de Init du service PathResolver

Objectif 2 : Tests de la conversion du plateau en un graphe

Cas de test 1	PathResolver::testconversionPLTMTL
Objectif du test	S'assurer que la conversion du plateau en un graphe est correcte pour les cases de type PLT et MTL.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: Create(Guard :: init(E_0, 4, 1), Guard :: init(E_0, 1, 3))$ $treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.7 $envi \stackrel{\text{def}}{=} Environnement :: init(9, 6, player, guards, treasures, cells)$
Opérations	$R_1 \stackrel{\text{def}}{=} init(envi, player)$
Oracle	**forall** *node* in [0..Array::Length(Graphe(R_1)] if (NatureOfNode(*node*) ∈ {PLT, MTL}) then List::IsEmpty(Graphe(R_1)[*node*]).

Cas de test 2	PathResolver::testconversionLAD1
Objectif du test	S'assurer que la conversion du plateau en un graphe est correcte pour les cases de type LAD.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: Create(Guard :: init(E_0, 4, 1), Guard :: init(E_0, 1, 3))$ $treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.7 $envi \stackrel{\text{def}}{=} Environnement :: init(9, 6, player, guards, treasures, cells)$
Opérations	$R_1 \stackrel{\text{def}}{=} init(envi, player)$ $expected \stackrel{\text{def}}{=}$ {NodeIdOfPosition(R_1, 1, 1), NodeIdOfPosition(R_1, 3, 1), NodeIdOfPosition(R_1, 2, 2)}

Oracle	$Graphe(R_1)[NodeIdOfPosition(R_1, 2, 1))] = expected.$

Cas de test 3	PathResolver::testconversionLAD2
Objectif du test	S'assurer que la conversion du plateau en un graphe est correcte pour les cases de type LAD.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: Create(Guard :: init(E_0, 4, 1), Guard :: init(E_0, 1, 3))$ $treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.7 $envi \stackrel{\text{def}}{=} Environnement :: init(9, 6, player, guards, treasures, cells)$
Opérations	$R_1 \stackrel{\text{def}}{=} init(envi, player)$ $expected \stackrel{\text{def}}{=} \{\text{NodeIdOfPosition}(R_1, 2, 1), \text{NodeIdOfPosition}(R_1, 2, 3)\}$
Oracle	$Graphe(R_1)[NodeIdOfPosition(R_1, 2, 2))] = expected.$

Cas de test 4	PathResolver::testconversionLAD3
Objectif du test	S'assurer que la conversion du plateau en un graphe est correcte pour les cases de type LAD.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: Create(Guard :: init(E_0, 4, 1), Guard :: init(E_0, 1, 3))$ $treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.7 $envi \stackrel{\text{def}}{=} Environnement :: init(9, 6, player, guards, treasures, cells)$
Opérations	$R_1 \stackrel{\text{def}}{=} init(envi, player)$ $expected \stackrel{\text{def}}{=} \{\text{NodeIdOfPosition}(R_1, 1, 3), \text{NodeIdOfPosition}(R_1, 3, 3), \text{NodeIdOfPosition}(R_1, 2, 2)\,; \text{NodeIdOfPosition}(R_1, 2, 4)\}$
Oracle	$Graphe(R_1)[NodeIdOfPosition(R_1, 2, 3))] = expected.$

Cas de test 5	PathResolver::testconversionLAD4
Objectif du test	S'assurer que la conversion du plateau en un graphe est correcte pour les cases de type LAD.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: Create(Guard :: init(E_0, 4, 1), Guard :: init(E_0, 1, 3))$ $treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.7

Opérations	$envi \stackrel{\text{def}}{=} Environnement :: init(9, 6, player, guards, treasures, cells)$ $R_1 \stackrel{\text{def}}{=} init(envi, player)$ $expected \stackrel{\text{def}}{=} \{\text{NodeIdOfPosition}(R_1,\ 1,\ 4),$ NodeIdOfPosition(R_1, 3, 4), NodeIdOfPosition(R_1, 2, 3) ; NodeIdOfPosition(R_1, 2, 5)$\}$
Oracle	$Graphe(R_1)[NodeIdOfPosition(R_1, 2, 4))] = expected.$

Cas de test 6	PathResolver::testconversionLAD5
Objectif du test	S'assurer que la conversion du plateau en un graphe est correcte pour les cases de type LAD.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: Create(Guard :: init(E_0, 4, 1), Guard :: init(E_0, 1, 3))$ $treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.7
Opérations	$envi \stackrel{\text{def}}{=} Environnement :: init(9, 6, player, guards, treasures, cells)$ $R_1 \stackrel{\text{def}}{=} init(envi, player)$ $expected \stackrel{\text{def}}{=} \{\text{NodeIdOfPosition}(R_1,\ 2,\ 4),$ NodeIdOfPosition(R_1, 1, 5) ; NodeIdOfPosition(R_1, 3, 5)$\}$
Oracle	$Graphe(R_1)[NodeIdOfPosition(R_1, 2, 5))] = expected.$

Cas de test 7	PathResolver::testconversionHDR1
Objectif du test	S'assurer que la conversion du plateau en un graphe est correcte pour les cases de type HDR.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: Create(Guard :: init(E_0, 4, 1), Guard :: init(E_0, 1, 3))$ $treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.7
Opérations	$envi \stackrel{\text{def}}{=} Environnement :: init(9, 6, player, guards, treasures, cells)$ $R_1 \stackrel{\text{def}}{=} init(envi, player)$ $expected \stackrel{\text{def}}{=} \{\text{NodeIdOfPosition}(R_1,\ 1,\ 3)\}$
Oracle	$Graphe(R_1)[NodeIdOfPosition(R_1, 0, 3))] = expected.$

Cas de test 8	PathResolver::testconversionHDR2
Objectif du test	S'assurer que la conversion du plateau en un graphe est correcte pour les cases de type HDR.

Condition initiales	$guards \stackrel{\text{def}}{=} List :: Create(Guard :: init(E_0, 4, 1), Guard :: init(E_0, 1, 3))$ $treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.7 $envi \stackrel{\text{def}}{=} Environnement :: init(9, 6, player, guards, treasures, cells)$
Opérations	$R_1 \stackrel{\text{def}}{=} init(envi, player)$ $expected \stackrel{\text{def}}{=}$ {NodeIdOfPosition(R_1, 1, 4), NodeIdOfPosition(R_1, 0, 3)}
Oracle	$Graphe(R_1)[NodeIdOfPosition(R_1, 0, 4))] = expected.$

Cas de test 9	PathResolver::testconversionHDR3
Objectif du test	S'assurer que la conversion du plateau en un graphe est correcte pour les cases de type HDR.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: Create(Guard :: init(E_0, 4, 1), Guard :: init(E_0, 1, 3))$ $treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.7 $envi \stackrel{\text{def}}{=} Environnement :: init(9, 6, player, guards, treasures, cells)$
Opérations	$R_1 \stackrel{\text{def}}{=} init(envi, player)$ $expected \stackrel{\text{def}}{=}$ {NodeIdOfPosition(R_1, 0, 4), NodeIdOfPosition(R_1, 2, 4), NodeIdOfPosition(R_1, 1, 3)}
Oracle	$Graphe(R_1)[NodeIdOfPosition(R_1, 1, 4))] = expected.$

Cas de test 10	PathResolver::testconversionHDR4
Objectif du test	S'assurer que la conversion du plateau en un graphe est correcte pour les cases de type HDR.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: Create(Guard :: init(E_0, 4, 1), Guard :: init(E_0, 1, 3))$ $treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.7 $envi \stackrel{\text{def}}{=} Environnement :: init(9, 6, player, guards, treasures, cells)$
Opérations	$R_1 \stackrel{\text{def}}{=} init(envi, player)$ $expected \stackrel{\text{def}}{=}$ {NodeIdOfPosition(R_1, 3, 3), NodeIdOfPosition(R_1, 5, 2), NodeIdOfPosition(R_1, 4, 1)}

Oracle	$Graphe(R_1)[NodeIdOfPosition(R_1, 4, 2))] = expected.$

Cas de test 11	PathResolver::testconversionHDR5
Objectif du test	S'assurer que la conversion du plateau en un graphe est correcte pour les cases de type HDR.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: Create(Guard :: init(E_0, 4, 1), Guard :: init(E_0, 1, 3))$ $treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.7 $envi \stackrel{\text{def}}{=} Environnement :: init(9, 6, player, guards, treasures, cells)$
Opérations	$R_1 \stackrel{\text{def}}{=} init(envi, player)$ $expected \stackrel{\text{def}}{=}$ {NodeIdOfPosition(R_1, 6, 3), NodeIdOfPosition(R_1, 4, 2), NodeIdOfPosition(R_1, 5, 1)}
Oracle	$Graphe(R_1)[NodeIdOfPosition(R_1, 5, 2))] = expected.$

Cas de test 12	PathResolver::testconversionHOL1
Objectif du test	S'assurer que la conversion du plateau en un graphe est correcte pour les cases de type HOL.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: Create(Guard :: init(E_0, 4, 1), Guard :: init(E_0, 1, 3))$ $treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.7 $envi \stackrel{\text{def}}{=} Environnement :: init(9, 6, player, guards, treasures, cells)$
Opérations	$R_1 \stackrel{\text{def}}{=} init(envi, player)$ $expected \stackrel{\text{def}}{=}$ {NodeIdOfPosition(R_1, 5, 2), NodeIdOfPosition(R_1, 7, 3)}
Oracle	$Graphe(R_1)[NodeIdOfPosition(R_1, 6, 3))] = expected.$

Cas de test 13	PathResolver::testconversionHOL2
Objectif du test	S'assurer que la conversion du plateau en un graphe est correcte pour les cases de type HOL.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: Create(Guard :: init(E_0, 4, 1), Guard :: init(E_0, 1, 3))$ $treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.7

	$envi \stackrel{\text{def}}{=} Environnement :: init(9, 6, player, guards, treasures, cells)$
Opérations	$R_1 \stackrel{\text{def}}{=} init(envi, player)$ $expected \stackrel{\text{def}}{=} \{\text{NodeIdOfPosition}(R_1,\ 6,\ 3), \text{NodeIdOfPosition}(R_1,\ 8,\ 2)\}$
Oracle	$Graphe(R_1)[NodeIdOfPosition(R_1, 7, 2))] = expected.$

Cas de test 14	PathResolver::testconversionHOL3
Objectif du test	S'assurer que la conversion du plateau en un graphe est correcte pour les cases de type HOL.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: Create(Guard :: init(E_0, 4, 1), Guard :: init(E_0, 1, 3))$ $treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.7 $envi \stackrel{\text{def}}{=} Environnement :: init(9, 6, player, guards, treasures, cells)$
Opérations	$R_1 \stackrel{\text{def}}{=} init(envi, player)$ $expected \stackrel{\text{def}}{=} \{\text{NodeIdOfPosition}(R_1,\ 7,\ 2), \text{NodeIdOfPosition}(R_1,\ 8,\ 3), \text{NodeIdOfPosition}(R_1,\ 8,\ 1)\}$
Oracle	$Graphe(R_1)[NodeIdOfPosition(R_1, 8, 2))] = expected.$

Cas de test 15	PathResolver::testconversionBordure1
Objectif du test	S'assurer que la conversion du plateau en un graphe est correcte pour les bordures.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: Create(Guard :: init(E_0, 4, 1), Guard :: init(E_0, 1, 3))$ $treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.7 $envi \stackrel{\text{def}}{=} Environnement :: init(9, 6, player, guards, treasures, cells)$
Opérations	$R_1 \stackrel{\text{def}}{=} init(envi, player)$ $expected \stackrel{\text{def}}{=} \{\text{NodeIdOfPosition}(R_1,\ 7,\ 1)\}$
Oracle	$Graphe(R_1)[NodeIdOfPosition(R_1, 8, 1))] = expected.$

Cas de test 16	PathResolver::testconversionBordure2
Objectif du test	S'assurer que la conversion du plateau en un graphe est correcte pour les bordures.

Condition initiales	$guards \stackrel{\text{def}}{=} List :: Create(Guard :: init(E_0, 4, 1), Guard :: init(E_0, 1, 3))$ $treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.7 $envi \stackrel{\text{def}}{=} Environnement :: init(9, 6, player, guards, treasures, cells)$
Opérations	$R_1 \stackrel{\text{def}}{=} init(envi, player)$ $expected \stackrel{\text{def}}{=} \{\text{NodeIdOfPosition}(R_1,\ 1,\ 1)\}$
Oracle	$Graphe(R_1)[NodeIdOfPosition(R_1, 0, 1))] = expected.$

Cas de test 17	PathResolver::testconversionBordure3
Objectif du test	S'assurer que la conversion du plateau en un graphe est correcte pour les bordures.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: Create(Guard :: init(E_0, 4, 1), Guard :: init(E_0, 1, 3))$ $treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.7 $envi \stackrel{\text{def}}{=} Environnement :: init(9, 6, player, guards, treasures, cells)$
Opérations	$R_1 \stackrel{\text{def}}{=} init(envi, player)$ $expected \stackrel{\text{def}}{=} \{\text{NodeIdOfPosition}(R_1,\ 0,\ 4),$ NodeIdOfPosition$(R_1,\ 1,\ 4)\}$
Oracle	$Graphe(R_1)[NodeIdOfPosition(R_1, 0, 5))] = expected.$

Description des tests de la conversion du plateau en un graphe du service PathResolver

Objectif 3 : Tests de ShortestPathToPlayer(P, g)

Cas de test 1	PathResolver::ShortestPathToPlayer1
Objectif du test	S'assurer que la suite de noeud renvoyé correspond bien au plus court chemin pour atteindre le joueur.
Condition initiales	$g1 \stackrel{\text{def}}{=} Guard :: init(E_0, 4, 1)$ $g2 \stackrel{\text{def}}{=} Guard :: init(E_0, 1, 3)$ $guards \stackrel{\text{def}}{=} List :: Create(g1, g2)$ $treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.7

	$envi \stackrel{\text{def}}{=} Environnement ::$ $init(9, 6, player, guards, treasures, cells)$
	$R_1 \stackrel{\text{def}}{=} init(envi, player)$
	$expected \stackrel{\text{def}}{=}$ [NodeIdOfPosition(R_1, 1, 1), NodeIdOfPosition(R_1, 2, 1), NodeIdOfPosition(R_1, 3, 1), NodeIdOfPosition(R_1, 4, 1)]
Opérations	$currentId \stackrel{\text{def}}{=}$NodeIdOfPosition($R_1$, Character::Wdt($player$), Character::Hgt($player$))
	$guardId \stackrel{\text{def}}{=}$NodeIdOfPosition($R_1$, Character::Wdt($g1$), Character::Hgt($g1$))
	$i = 0$
	$pred \stackrel{\text{def}}{=} ShortestPathToPlayer(R_1, g1)$
Oracle	**while** $currentId \neq guardId$
	$expected$[i] = $currentId$
	$currentId \stackrel{\text{def}}{=} pred[currentId]$
	$i \stackrel{\text{def}}{=} i + 1$

Cas de test 2	PathResolver::ShortestPathToPlayer1
Objectif du test	S'assurer que la suite de noeud renvoyé correspond bien au plus court chemin pour atteindre le joueur.
Condition initiales	$g1 \stackrel{\text{def}}{=} Guard :: init(E_0, 4, 1)$
	$g2 \stackrel{\text{def}}{=} Guard :: init(E_0, 1, 3)$
	$guards \stackrel{\text{def}}{=} List :: Create(g1, g2)$
	$treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$
	$player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$
	voir valeur *cells* 3.7
	$envi \stackrel{\text{def}}{=} Environnement ::$ $init(9, 6, player, guards, treasures, cells)$
	$R_1 \stackrel{\text{def}}{=} init(envi, player)$
	$expected \stackrel{\text{def}}{=}$ [NodeIdOfPosition(R_1, 1, 1), NodeIdOfPosition(R_1, 2, 1), NodeIdOfPosition(R_1, 2, 2), NodeIdOfPosition(R_1, 2, 3), NodeIdOfPosition(R_1, 1, 3)]
Opérations	$currentId \stackrel{\text{def}}{=}$NodeIdOfPosition($R_1$, Character::Wdt($player$), Character::Hgt($player$))
	$guardId \stackrel{\text{def}}{=}$NodeIdOfPosition($R_1$, Character::Wdt($g2$), Character::Hgt($g2$))
	$i = 0$
	$pred \stackrel{\text{def}}{=} ShortestPathToPlayer(R_1, g2)$
Oracle	**while** $currentId \neq guardId$
	$expected$[i] = $currentId$
	$currentId \stackrel{\text{def}}{=} pred[currentId]$

$i \overset{\text{def}}{=} i + 1$

Description des tests de ShortestPathToPlayer du service PathResolver

Objectif 4 : Tests de NextMoveToreachPlayer(P, g)

Cas de test 1	PathResolver::ShortestPathToPlayer1
Objectif du test	S'assurer que les instructions renvoyées par NextMoveToReachPlayer permettent bien au garde d'aller à la position du joueur.
Condition initiales	$g1 \overset{\text{def}}{=} Guard :: init(E_0, 4, 1)$ $g2 \overset{\text{def}}{=} Guard :: init(E_0, 1, 3)$ $guards \overset{\text{def}}{=} List :: Create(g1, g2)$ $treasures \overset{\text{def}}{=} List :: CreateEmpty()$ $player \overset{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.7 $envi \overset{\text{def}}{=} Environnement ::$ $init(9, 6, player, guards, treasures, cells)$ $R_1 \overset{\text{def}}{=} init(envi, player)$
Opérations	$res1 \overset{\text{def}}{=}$ NextMoveToReachPlayer(R_1, $g1$) $g1 \overset{\text{def}}{=}$ Character::GoLeft($g1$) $res2 \overset{\text{def}}{=}$ NextMoveToReachPlayer(R_1, $g1$) $g1 \overset{\text{def}}{=}$ Character::GoLeft($g1$) $res3 \overset{\text{def}}{=}$ NextMoveToReachPlayer(R_1, $g1$) $g1 \overset{\text{def}}{=}$ Character::GoLeft($g1$) $res4 \overset{\text{def}}{=}$ NextMoveToReachPlayer(R_1, $g1$)
Oracle	$res1$ = LEFT **and** $res2$ = LEFT **and** $res3$ = LEFT **and** $res4$ = NEUTRAL

Cas de test 2	PathResolver::ShortestPathToPlayer2
Objectif du test	S'assurer que les instructions renvoyées par NextMoveToReachPlayer permettent bien au garde d'aller à la position du joueur.
Condition initiales	$g1 \overset{\text{def}}{=} Guard :: init(E_0, 4, 1)$ $g2 \overset{\text{def}}{=} Guard :: init(E_0, 1, 3)$ $guards \overset{\text{def}}{=} List :: Create(g1, g2)$ $treasures \overset{\text{def}}{=} List :: CreateEmpty()$ $player \overset{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.7 $envi \overset{\text{def}}{=} Environnement ::$ $init(9, 6, player, guards, treasures, cells)$

	$R_1 \stackrel{\text{def}}{=} init(envi, player)$
Opérations	$res1 \stackrel{\text{def}}{=}$ NextMoveToReachPlayer(R_1, $g2$)
	$g2 \stackrel{\text{def}}{=}$ Character::GoRight($g2$)
	$res2 \stackrel{\text{def}}{=}$ NextMoveToReachPlayer(R_1, $g2$)
	$g2 \stackrel{\text{def}}{=}$ Character::GoDown($g2$)
	$res3 \stackrel{\text{def}}{=}$ NextMoveToReachPlayer(R_1, $g2$)
	$g2 \stackrel{\text{def}}{=}$ Character::GoDown($g2$)
	$res4 \stackrel{\text{def}}{=}$ NextMoveToReachPlayer(R_1, $g2$)
	$g2 \stackrel{\text{def}}{=}$ Character::GoLeft($g2$)
	$res5 \stackrel{\text{def}}{=}$ NextMoveToReachPlayer(R_1, $g2$)
Oracle	$res1$ = RIGHT **and** $res2$ = DOWN **and** $res3$ = DOWN **and** $res4$ = LEFT **and** $res5$ = NEUTRAL

Cas de test 3	PathResolver::ShortestPathToPlayer3
Objectif du test	S'assurer que si le garde ne peut atteindre le joueur, NEUTRAL est renvoyé.
Condition initiales	$g1 \stackrel{\text{def}}{=} Guard :: init(E_0, 4, 1)$
	$g2 \stackrel{\text{def}}{=} Guard :: init(E_0, 8, 5)$
	$guards \stackrel{\text{def}}{=} List :: Create(g1, g2)$
	$treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$
	$player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$
	voir valeur *cells* 3.7
	$envi \stackrel{\text{def}}{=} Environnement ::$ $init(9, 6, player, guards, treasures, cells)$
	$R_1 \stackrel{\text{def}}{=} init(envi, player)$
Opérations	$res \stackrel{\text{def}}{=}$ NextMoveToReachPlayer(R_1, $g2$)
Oracle	res = NEUTRAL

Description des tests de NextMoveToreachPlayer du service PathResolver

Objectif 5 : Tests de PlayerCanReachPos(P, x, y)

Cas de test 1	PathResolver::PlayerCanReachPosPositif1
Objectif du test	S'assurer le resolver indique bien que le joueur peut atteindre cette position.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: Create(Guard ::$ $init(E_0, 4, 1), Guard :: init(E_0, 1, 3))$
	$treasures \stackrel{\text{def}}{=} List :: CreateEmpty()$
	$player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$
	voir valeur *cells* 3.7

	$envi \overset{\text{def}}{=} Environnement :: init(9, 6, player, guards, treasures, cells)$ $R_1 \overset{\text{def}}{=} init(envi, player)$
Opérations	$res1 \overset{\text{def}}{=} \text{PlayerCanReachPos}(R_1,\ 3,\ 5)$
Oracle	$res1 = true$

Cas de test 2	PathResolver::PlayerCanReachPosPositif2
Objectif du test	S'assurer le resolver indique bien que le joueur peut atteindre cette position.
Condition initiales	$guards \overset{\text{def}}{=} List :: Create(Guard :: init(E_0, 4, 1), Guard :: init(E_0, 1, 3))$ $treasures \overset{\text{def}}{=} List :: CreateEmpty()$ $player \overset{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.7 $envi \overset{\text{def}}{=} Environnement :: init(9, 6, player, guards, treasures, cells)$ $R_1 \overset{\text{def}}{=} init(envi, player)$
Opérations	$res1 \overset{\text{def}}{=} \text{PlayerCanReachPos}(R_1,\ 1,\ 3)$
Oracle	$res1 = true$

Cas de test 3	PathResolver::PlayerCanReachPosNegatif1
Objectif du test	S'assurer le resolver indique bien que le joueur ne peut pas atteindre cette position.
Condition initiales	$guards \overset{\text{def}}{=} List :: Create(Guard :: init(E_0, 4, 1), Guard :: init(E_0, 1, 3))$ $treasures \overset{\text{def}}{=} List :: CreateEmpty()$ $player \overset{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.7 $envi \overset{\text{def}}{=} Environnement :: init(9, 6, player, guards, treasures, cells)$ $R_1 \overset{\text{def}}{=} init(envi, player)$
Opérations	$res1 \overset{\text{def}}{=} \text{PlayerCanReachPos}(R_1,\ 8,\ 5)$
Oracle	$res1 = false$

Cas de test 4	PathResolver::PlayerCanReachPosNegatif2
Objectif du test	S'assurer le resolver indique bien que le joueur ne peut pas atteindre cette position.
Condition initiales	$guards \overset{\text{def}}{=} List :: Create(Guard :: init(E_0, 4, 1), Guard :: init(E_0, 1, 3))$ $treasures \overset{\text{def}}{=} List :: CreateEmpty()$ $player \overset{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.7

	$envi \stackrel{\text{def}}{=} Environnement :: init(9, 6, player, guards, treasures, cells)$ $R_1 \stackrel{\text{def}}{=} init(envi, player)$
Opérations	$res1 \stackrel{\text{def}}{=}$ PlayerCanReachPos(R_1, 1, 0)
Oracle	$res1 = false$

Description des tests de PlayerCanReachPos du service PathResolver

Objectif 6 : Tests de PlayerCanReachAllTreasuresAndPortal(P, treasures, portalX, portalY)

Cas de test 1	PathResolver::PlayerCanReachAllTreasuresAndPortalPositif1
Objectif du test	S'assurer le resolver indique bien que le joueur peut atteindre toute les positions si elles sont toutes atteignable.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: CreateEmpty()$ $treasures \stackrel{\text{def}}{=} List :: Create(Treasure :: init(E_0, 0, 1), Treasure :: init(E_0, 8, 1), Treasure :: init(E_0, 6, 3))$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.7 $envi \stackrel{\text{def}}{=} Environnement :: init(9, 6, player, guards, treasures, cells)$ $R_1 \stackrel{\text{def}}{=} init(envi, player)$
Opérations	$res1 \stackrel{\text{def}}{=}$ PlayerCanReachAllTreasuresAndPortal(R_1, treasures, 3, 5)
Oracle	$res1 = true$

Cas de test 2	PathResolver::PlayerCanReachAllTreasuresAndPortalPositif2
Objectif du test	S'assurer le resolver indique bien que le joueur peut atteindre toute les positions si elles sont toutes atteignable.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: CreateEmpty()$ $treasures \stackrel{\text{def}}{=} List :: Create(Treasure :: init(E_0, 1, 3), Treasure :: init(E_0, 4, 1), Treasure :: init(E_0, 0, 3))$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.7 $envi \stackrel{\text{def}}{=} Environnement :: init(9, 6, player, guards, treasures, cells)$ $R_1 \stackrel{\text{def}}{=} init(envi, player)$
Opérations	$res1 \stackrel{\text{def}}{=}$ PlayerCanReachAllTreasuresAndPortal(R_1, treasures, 2, 3)
Oracle	$res1 = true$

Cas de test 3	PathResolver::PlayerCanReachAllTreasuresAndPortalNegatif1
Objectif du test	S'assurer le resolver retourne bien faux si un trésor est inatteignable.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: CreateEmpty()$ $treasures \stackrel{\text{def}}{=} List :: Create(Treasure :: init(E_0, 7, 1), Treasure :: init(E_0, 1, 5), Treasure :: init(E_0, 0, 1))$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.7 $E_1 \stackrel{\text{def}}{=} Environnement :: init(9, 6, player, guards, treasures, cells)$ $envi \stackrel{\text{def}}{=} SetNature(SetNature(E_1, 6, 1, PLT), 8, 2PLT)$ $R_1 \stackrel{\text{def}}{=} init(envi, player)$
Opérations	$res1 \stackrel{\text{def}}{=}$ PlayerCanReachAllTreasuresAndPortal(R_1, treasures, 2, 3)
Oracle	$res1 = false$

Cas de test 4	PathResolver::PlayerCanReachAllTreasuresAndPortalNegatif2
Objectif du test	S'assurer le resolver retourne bien faux si le portail est inatteignable.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: CreateEmpty()$ $treasures \stackrel{\text{def}}{=} List :: Create(Treasure :: init(E_0, 7, 1), Treasure :: init(E_0, 1, 5), Treasure :: init(E_0, 0, 1))$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.7 $envi \stackrel{\text{def}}{=} Environnement :: init(9, 6, player, guards, treasures, cells)$ $R_1 \stackrel{\text{def}}{=} init(envi, player)$
Opérations	$res1 \stackrel{\text{def}}{=}$ PlayerCanReachAllTreasuresAndPortal(R_1, treasures, 0, 5)
Oracle	$res1 = false$

Cas de test 5	PathResolver::PlayerCanReachAllTreasuresAndPortalNegatif3
Objectif du test	S'assurer le resolver retourne bien faux si le joueur ne peut rien atteindre.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: CreateEmpty()$ $treasures \stackrel{\text{def}}{=} List :: Create(Treasure :: init(E_0, 3, 4), Treasure :: init(E_0, 6, 4), Treasure :: init(E_0, 0, 2))$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$

	voir valeur *cells* 3.7 $E_1 \stackrel{\text{def}}{=} Environnement ::$ $init(9, 6, player, guards, treasures, cells)$ $envi \stackrel{\text{def}}{=} SetNature(E_1, 2, 1, MTL)$ $R_1 \stackrel{\text{def}}{=} init(envi, player)$
Opérations	$res1 \stackrel{\text{def}}{=}$ PlayerCanReachAllTreasuresAndPortal(R_1, treasures, 1, 3)
Oracle	$res1 = false$

Cas de test 6	PathResolver::PlayerCanReachAllTreasuresAndPortalNegatif4
Objectif du test	S'assurer le resolver retourne bien faux si le joueur ne peut atteindre que le portail.
Condition initiales	$guards \stackrel{\text{def}}{=} List :: CreateEmpty()$ $treasures \stackrel{\text{def}}{=} List :: Create(Treasure :: init(E_0, 3, 4), Treasure :: init(E_0, 6, 4), Treasure :: init(E_0, 0, 2))$ $player \stackrel{\text{def}}{=} Player :: init(E_0, 1, 1)$ voir valeur *cells* 3.7 $E_1 \stackrel{\text{def}}{=} Environnement ::$ $init(9, 6, player, guards, treasures, cells)$ $envi \stackrel{\text{def}}{=} SetNature(E_1, 2, 1, MTL)$ $R_1 \stackrel{\text{def}}{=} init(envi, player)$
Opérations	$res1 \stackrel{\text{def}}{=}$ PlayerCanReachAllTreasuresAndPortal(R_1, treasures, 0, 1)
Oracle	$res1 = false$

Description des tests de PlayerCanReachAllTreasuresAndPortal du service PathResolver

3.8 GameObject

Objectif 1 : Tests de Init(s, x, y)

Cas de test 1	GameObject::testInitPositif1
Objectif du test	S'assurer que le init avec des valeurs positives ne renvoie pas d'erreur.
Condition initiales	$screen \stackrel{\text{def}}{=} Screen :: init(7, 6)$
Opérations	$GO_1 \stackrel{\text{def}}{=} init(1, 1);$
Oracle	Pas d'exception levée

Cas de test 2	GameObject::testInitPositif2
Objectif du test	S'assurer que le init avec les valeurs positives minimales ne renvoie pas d'erreur.
Condition initiales	$screen \stackrel{\text{def}}{=} Screen :: init(7, 6)$

Opérations	$GO_1 \stackrel{\text{def}}{=} init(6,5)$;
Oracle	Pas d'exception levée

Cas de test 3	GameObject::testInitNegatif1
Objectif du test	S'assurer que le init avec un x négatif renvoie une erreur.
Condition initiales	$screen \stackrel{\text{def}}{=} Screen :: init(7,6)$
Opérations	$GO_1 \stackrel{\text{def}}{=} init(-1,0)$;
Oracle	Exception levée

Cas de test 4	GameObject::testInitNegatif2
Objectif du test	S'assurer que le init avec une un y négatif renvoie une erreur.
Condition initiales	$screen \stackrel{\text{def}}{=} Screen :: init(7,6)$
Opérations	$GO_1 \stackrel{\text{def}}{=} init(0,-1)$;
Oracle	Exception levée

Description des tests du Init du service GameObject

3.9 Attack

Objectif 1 : Tests de Init(s, x, y, d)

Cas de test 1	Attack::testInitPositif1
Objectif du test	S'assurer que le init avec une direction égale à LEFT ne renvoie pas d'erreur.
Condition initiales	$screen \stackrel{\text{def}}{=} Screen :: init(7,6)$
Opérations	$A_1 \stackrel{\text{def}}{=} init(1,1,LEFT)$;
Oracle	Pas d'exception levée

Cas de test 2	Attack::testInitPositif2
Objectif du test	S'assurer que le init avec une direction égale à RIGHT ne renvoie pas d'erreur.
Condition initiales	$screen \stackrel{\text{def}}{=} Screen :: init(7,6)$
Opérations	$A_1 \stackrel{\text{def}}{=} init(1,1,RIGHT)$;
Oracle	Pas d'exception levée

Description des tests du Init du service Attack

Objectif 2 : Tests de Step(A)

Cas de test 1	Attack::testStepPositif1
Objectif du test	S'assurer qu'une attaque avec une direction LEFT diminura son Wdt en faisant Step.

Condition initiales	$screen \stackrel{\text{def}}{=} Screen :: init(7, 6)$
Opérations	$A_1 \stackrel{\text{def}}{=} init(1, 1, LEFT)$
	$A_2 = Step(A_1)$
Oracle	GameObject::Wdt$(A_1) - 1 =$ GameObject::Wdt(A_2)
	and GameObject::Hgt$(A_1) =$ GameObject::Hgt(A_2)

Cas de test 2	Attack::testStepPositif2
Objectif du test	S'assurer qu'une attaque avec une direction RIGHT augmentera son Wdt en faisant Step.
Condition initiales	$screen \stackrel{\text{def}}{=} Screen :: init(7, 6)$
Opérations	$A_1 \stackrel{\text{def}}{=} init(1, 1, RIGHT)$
	$A_2 = Step(A_1)$
Oracle	GameObject::Wdt$(A_1) + 1 =$ GameObject::Wdt(A_2)
	and GameObject::Hgt$(A_1) =$ GameObject::Hgt(A_2)

Description des tests du Step du service Attack

3.10 Engine

L'ensemble des tests du Service Engine se déroule dans l'environnement indiqué par la figure 12.

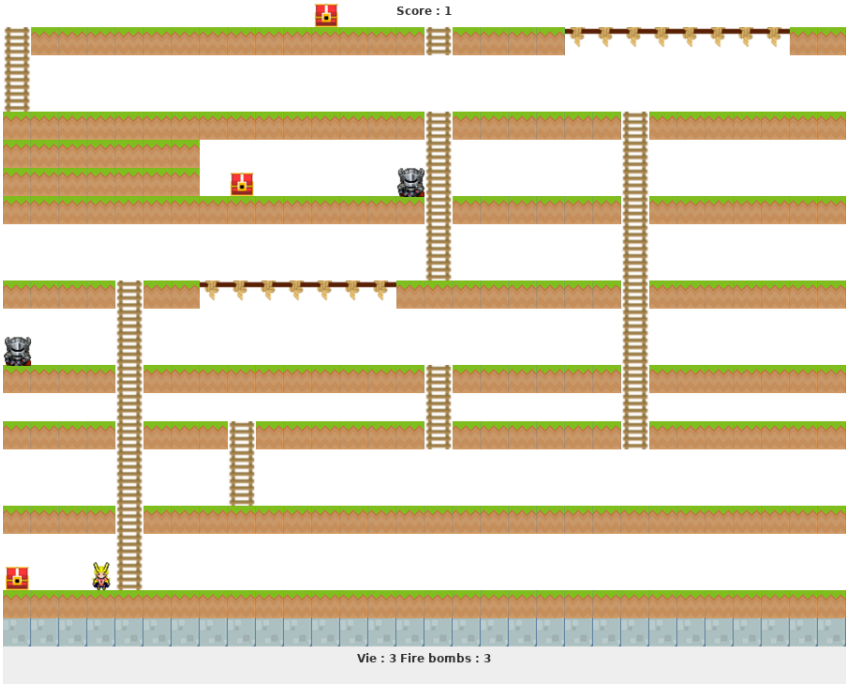

FIGURE 12 – Terrain des tests de Engine.

Objectif 1 : Tests de Init(wdtScreen, hgtScreen, playerX, playerY)

Cas de test 1	Engine::testInitPositif
Objectif du test	S'assurer que le init ne renvoie pas d'erreur.
Condition initiales	aucune
Opérations	$engine \stackrel{\text{def}}{=} Engine :: init(30, 23, 3, 2)$
Oracle	Pas d'exception levée

Cas de test 2	Engine::testInitNegatif1
Objectif du test	S'assurer que le init avec une $wdtScreen$ nulle renvoie une erreur.
Condition initiales	aucune
Opérations	$engine \stackrel{\text{def}}{=} Engine :: init(0, 23, 3, 2)$
Oracle	Exception levée

Cas de test 3	Engine::testInitNegatif2
Objectif du test	S'assurer que le init avec une $hgtScreen$ négative renvoie une erreur.
Condition initiales	aucune
Opérations	$engine \stackrel{\text{def}}{=} Engine :: init(30, -5, 3, 2)$
Oracle	Exception levée

Cas de test 4	Engine::testInitNegatif3
Objectif du test	S'assurer que le init avec un $playerX$ supérieur à la $wdtScreen$ renvoie une erreur.
Condition initiales	aucune
Opérations	$engine \stackrel{\text{def}}{=} Engine :: init(30, 23, 31, 2)$
Oracle	Exception levée

Cas de test 4	Engine::testInitNegatif3
Objectif du test	S'assurer que le init avec un $playerY$ égal à la $hgtScreen$ renvoie une erreur.
Condition initiales	aucune
Opérations	$engine \stackrel{\text{def}}{=} Engine :: init(30, 23, 31, 23)$
Oracle	Exception levée

Description des tests du Init du service Engine

Objectif 2 : Tests de AddHole(E, x, y)

Cas de test 1	Engine::testAddHolePositif1
Objectif du test	S'assurer que le AddHole ne renvoie pas d'erreur.
Condition initiales	$E_1 \stackrel{\text{def}}{=} Engine :: init(30, 23, 3, 2)$
Opérations	$AddHole(E_1, 4, 1)$

Oracle	Pas d'exception levée

Cas de test 2	Engine::testAddHolePositif2
Objectif du test	S'assurer que le AddHole ne renvoie pas d'erreur.
Condition initiales	$engine \stackrel{\text{def}}{=} Engine :: init(30, 23, 3, 2)$
Opérations	$AddHole(E_1, 2, 1)$
Oracle	Pas d'exception levée

Cas de test 3	Engine::testAddHolePositif3
Objectif du test	S'assurer qu'un AddHole sur une case non PLT ne lève pas d'erreur mais ne modifie pas le type de la case.
Condition initiales	$E_1 \stackrel{\text{def}}{=} Engine :: init(30, 23, 3, 2)$
Opérations	$E_2 \stackrel{\text{def}}{=} AddHole(E_1, 0, 0)$
Oracle	$Screen :: CellNature(CurrentEnvironnement(E_2), 0, 0) = MTL$

Cas de test 4	Engine::testAddHoleNegatif1
Objectif du test	S'assurer qu'un AddHole sur une case contenant déjà un trou renvoie une erreur.
Condition initiales	$E_1 \stackrel{\text{def}}{=} Engine :: init(30, 23, 3, 2)$
Opérations	$E_2 \stackrel{\text{def}}{=} AddHole(AddHole(E_1, 0, 0), 0, 0)$
Oracle	Exception levée.

Cas de test 5	Engine::testAddHoleNegatif2
Objectif du test	S'assurer qu'un AddHole avec un x négatif renvoie une erreur.
Condition initiales	$E_1 \stackrel{\text{def}}{=} Engine :: init(30, 23, 3, 2)$
Opérations	$E_2 \stackrel{\text{def}}{=} AddHole(E_1, -5, 22)$
Oracle	Exception levée.

Cas de test 6	Engine::testAddHoleNegatif3
Objectif du test	S'assurer qu'un AddHole avec un y égal à la hauteur de l'écran renvoie une erreur.
Condition initiales	$E_1 \stackrel{\text{def}}{=} Engine :: init(30, 23, 3, 2)$
Opérations	$E_2 \stackrel{\text{def}}{=} AddHole(E_1, 2, 23)$
Oracle	Exception levée.

Description des tests du AddHole du service Engine

Objectif 3 : Tests de AddAttack(E, x, y, d)

Cas de test 1	Engine::testAddAttackPositif1

Objectif du test	S'assurer que le AddAttack ne renvoie pas d'erreur.
Condition initiales	$E_1 \stackrel{\text{def}}{=} Engine :: init(30, 23, 3, 2)$
Opérations	$AddAttack(E_1, 4, 2, LEFT)$
Oracle	Pas d'exception levée

Cas de test 2	Engine::testAddAttackPositif2
Objectif du test	S'assurer qu'un AddAttack en position 0 :0 ne renvoie pas d'erreur.
Condition initiales	$E_1 \stackrel{\text{def}}{=} Engine :: init(30, 23, 3, 2)$
Opérations	$AddAttack(E_1, 0, 0, RIGHT)$
Oracle	Pas d'exception levée

Cas de test 3	Engine::testAddAttackNegatif1
Objectif du test	S'assurer qu'une attaque ne peut être ajoutée la ou une attaque similaire est déjà présente.
Condition initiales	$E_1 \stackrel{\text{def}}{=} Engine :: init(30, 23, 3, 2)$
Opérations	$AddAttack(AddAttack(E_1, 4, 2, RIGHT), 4, 2)$
Oracle	Exception levée

Cas de test 4	Engine::testAddAttackNegatif2
Objectif du test	S'assurer que l'ajout d'une attaque avec un x négatif renvoie une erreur.
Condition initiales	$E_1 \stackrel{\text{def}}{=} Engine :: init(30, 23, 3, 2)$
Opérations	$AddAttack(E_1, -5, 1, LEFT)$
Oracle	Exception levée

Cas de test 5	Engine::testAddAttackNegatif3
Objectif du test	S'assurer que l'ajout d'une attaque avec un y égal à la hauteur du terrain renvoie une erreur.
Condition initiales	$E_1 \stackrel{\text{def}}{=} Engine :: init(30, 23, 3, 2)$
Opérations	$AddAttack(E_1, 2, 23, RIGHT)$
Oracle	Exception levée

Description des tests du AddAttack du service Engine

Objectif 4 : Tests de SetNextCommand(E, c)

Cas de test 1	Engine::testSetNextCommandPositif
Objectif du test	S'assurer que le SetNextCommand ne renvoie pas d'erreur et assigne bien la commande.
Condition initiales	$E_1 \stackrel{\text{def}}{=} Engine :: init(30, 23, 3, 2)$
Opérations	$E_2 \stackrel{\text{def}}{=} SetNextCommand(E_1, RIGHT)$ $E_3 \stackrel{\text{def}}{=} SetNextCommand(E_2, NEUTRAL)$

Oracle	$NextCommand(E_2) = RIGHT$ **and** $NextCommand(E_3) = NEUTRAL$

Description du test du SetNextCommand du service Engine

Objectif 5 : Tests de CreateEnvironnement(E, wdt, hgt)

Cas de test 1	Engine::testCreateEnvironnementPositif
Objectif du test	S'assurer que le CreateEnvironnement ne renvoie pas d'erreur.
Condition initiales	$E_1 \stackrel{\text{def}}{=} Engine :: init(30, 23, 3, 2)$
Opérations	$CreateEnvironnement(E_1, 30, 23)$
Oracle	Pas d'exception levée.

Description du test du CreateEnvironnement du service Engine

Objectif 6 : Tests de CreateEnvironnementWithContrat(E, wdt, hgt)

Cas de test 1	Engine::testCreateEnvironnementPositif
Objectif du test	S'assurer que le CreateEnvironnementWithContrat ne renvoie pas d'erreur.
Condition initiales	$E_1 \stackrel{\text{def}}{=} Engine :: init(30, 23, 3, 2)$
Opérations	$CreateEnvironnementWithContrat(E_1, 30, 23)$
Oracle	Pas d'exception levée.

Description du test du CreateEnvironnementWithContrat du service Engine

Objectif 7 : Tests de NextLevel(E)

Cas de test 1	Engine::testNextLevel
Objectif du test	S'assurer que le NextLevel ne renvoie pas d'erreur. Ici il n'y a pas de prochain niveau.
Condition initiales	$E_1 \stackrel{\text{def}}{=} Engine :: init(30, 23, 3, 2)$
Opérations	$E_2 \stackrel{\text{def}}{=} NextLevel(E_1)$
Oracle	$CurrentLevel(E_2) = \infty$ **and** $Status(E_2) = WIN$.

Description du test du NextLevel du service Engine

Objectif 8 : Tests de Step(E)

Cas de test 1	Engine::testStepPositif1

Objectif du test	S'assurer que Step ne renvoie pas d'erreur et incrémente NbStep.
Condition initiales	$E_1 \stackrel{\text{def}}{=} Engine :: init(30, 23, 3, 2)$
Opérations	$E_2 \stackrel{\text{def}}{=} Step(E_1)$
Oracle	$NbStep(E_2) = 1$

Cas de test 2	Engine::testStepPositif2
Objectif du test	S'assurer que Step incrémente le paramètre t des trous et gère les mouvement des attaques (ici deux attaques sortent de l'écran et doivent être supprimées).
Condition initiales	$E_1 \stackrel{\text{def}}{=} Engine :: init(30, 23, 3, 2)$
Opérations	$E_2 \stackrel{\text{def}}{=}$ $AddHole(AddHole(AddHole(E_1, 4, 21), 23, 18), 17, 12)$ $E_3 \stackrel{\text{def}}{=}$ $AddAttack(AddAttack(AddAttack(E_2, 10, 8, LEFT), 29, 5, RIGHT), 0$ $E_4 \stackrel{\text{def}}{=} Step(Step(E_3))$
Oracle	**forall** $< h, t >$ in Holes(E_4) $t = 2$ **and** $List :: Size(Attacks(E_4)) = 2$ **and** $GameObject :: Wdt(List :: get(Attacks(E_4), 0)) = 8$ **and** $GameObject :: Hgt(List :: get(Attacks(E_4), 0)) = 8$

Cas de test 3	Engine::testStepPositif3
Objectif du test	Un trésor est présent trois cases à gauche du Player. Ici on s'assure que le Step fera disparaître les trésor si le joueur est dessus.
Condition initiales	$E_1 \stackrel{\text{def}}{=} Engine :: init(30, 23, 3, 2)$
Opérations	$E_2 \stackrel{\text{def}}{=} SetNextCommand(E_1, LEFT)$ $E_3 \stackrel{\text{def}}{=} Step(E_2)$ $E_4 \stackrel{\text{def}}{=} SetNextCommand(E_3, LEFT)$ $E_5 \stackrel{\text{def}}{=} Step(E_4)$ $E_6 \stackrel{\text{def}}{=} SetNextCommand(E_5, LEFT)$ $E_7 \stackrel{\text{def}}{=} Step(E_6)$
Oracle	$List :: Size(Treasures(E_1)) - 1 = List ::$ $Size(Treasures(E_7))$ **and** $Score(E_7) = 1$

Cas de test 4	Engine::testStepPositif4
Objectif du test	S'assurer que si le joueur meurt, le statut de Engine passe à LOSS.
Condition initiales	$E_1 \stackrel{\text{def}}{=} Engine :: init(30, 23, 3, 2)$

Opérations	**tant que** $Vie(Player(E_1)) > 0$
	$E_1 \overset{\text{def}}{=} Step(E_1)$
Oracle	$Status(E_1) = LOSS$

Description des tests du Step du service Engine

4 Rapport de projet

4.1 Présentations succinctes des services

Ces sections vous présentent de manière brève, les ajouts ou modifications aux services existants. Nous présenterons également les services ajoutés au projet.
Pour plus de précisions sur ces services, nous vous invitons à consulter leurs *spécifications formelles* à la partie 2).

4.1.1 Screen

Le *"Screen"* est un service offrant une représentation du terrain de jeu.
Il s'initialise grâce à des entiers, représentant le nombre de cellules en largeur et le nombre de cellules en hauteur.
Il offre également un moyen de connaître la nature d'une cellule grâce à ses coordonnées, et de pouvoir transformer une case de nature PLT en HOL, et inversement grâce aux opérations *"Dig"* et *"Fill"*.

4.1.2 EditableScreen

Un *écran éditable*, est un service qui inclut le service *"Screen"* et qui permet de modifier la nature des cellules de l'environnement, et de générer des cellules sous la forme : Cells[][], contenant les modifications apportées. Ce service est uniquement utilisé par l'Engine au moment de son initialisation pour permettre la modification des cases en fonction des fichiers définissant les niveaux (voir partie "*Les niveaux*" dans la section 4.1.6).
Son opération *"getPlayableEnvi"*, renvoie ce double tableau de cellules, et permet d'initialiser un Environnement avec ces cellules.

4.1.3 Environnement

Un environnement, est un *"Screen"* permettant de savoir le contenu des cellules, c'est-à-dire s'il y a un "Character" ou un "Item" à des coordonnées fournies.

Son initialisation : Il possède plusieurs constructeurs, découlant du constructeur de *"Screen"*, le sevice qu'il *include*, et d'un constructeur prenant en plus un tableau à double dimension (Cell[][]) représentant les cellules et leurs natures dans l'environnement.

Les gardes et les trésors : Ses observateurs, *"Guards"* et *"Treasures"* stockent respectivement la liste des gardes et des trésors présents dans cet environnement. Ces listes sont maintenues à jour par l'Engine en fonction de l'évolution de l'état du jeu après chaque *"Step"*.

4.1.4 Character

Le service *"Character"* a le même rôle que le service *"Character"* décrit dans le projet. Il offre notamment des opérations pour se déplacer dans le terrain. Nous avons cependant apporté quelques modifications à ses déplacements. Ainsi, si un personnage est dans un trou, celui-ci peut sortir de celui-ci en faisant GoLeft ou GoRight (si la case visée est libre). Il est aussi possible pour un personnage dans un trou de se déplacer sur la case juste à sa droite ou à sa gauche si celle-ci est un LAD ou un HDR.
Nous avons aussi ajouté l'opération *"BackInitialPosition"*. Cette opération permet de faire revenir un personnage à sa position initiale. Elle est notamment utilisée lorsqu'un personnage est dans un trou et que celui-ci se rebouche. Un observator *"LastAction"* a aussi été ajouté. Celui-ci permet d'obtenir la dernière action faite par le personnage concerné. Il nous a notamment permis de tester que le personnage exécuté bien l'action que nous attendions.

4.1.5 Player

"Player" est un service représentant le joueur pendant une partie. Il inclut le service *"Character"*, et y ajoute des observateurs sur : les points de vie et le nombre d'attaques possible. Un opérateur *"Step"* a également été ajouté pour savoir quelle action effectuer à chaque tour de jeu.

Des opérateurs pour décrémenter les points de vie et le nombre d'attaques ont également été ajoutés. Ils seront appelés par son observateur *"Engine"* si cela est nécessaire. À *l'initialisation*, la valeur des observateurs *"Vie"* et *"nbAttack"* est 3.

4.1.6 Engine

Le service *"Engine"* est le coeur du jeu. C'est lui qui coordonne l'avancement du jeu notamment grâce à son opération *"Step"*, faisant bouger le joueur, les gardes et les attaques.

Avancement des différents acteurs : Dans l'opération *"Step"*, il lancera le *"Step"* du joueur, ainsi que celui des gardes et des attaques. Les actions que cela engendrera sont définies dans les services respectifs de ceux-ci.

Les contacts : Les différents contacts possibles suite à un *"Step"* des acteurs sont les suivants :

- Joueur et garde : Diminue les points de vie du joueur.
- Joueur et trésor : Augmente le score du joueur et fais disparaître le trésor.
- Attaque et garde : Renvoi le garde à sa position initiale.

Les trous : Engine possède un observateur sur les trous. Cet observateur est une table d'association entre un trou et un entier. L'entier représente le nombre de tours depuis lequel le trou est présent dans l'environnement. À chaque *"Step"*, cet entier est incrémenté, une fois qu'il est égal à 15 le trou se refermera, ce qui se traduit par la suppression du trou dans la liste.
Si le joueur se trouve dans un trou qui se referme, il perdra 1 point de vie et retournera à sa position initiale. Si c'est un garde qui se trouve dans le trou, il retournera également à sa position initiale.

Les niveaux : Lors d'un *"Step"*, Engine vérifiera si tout les trésors on été récupérés, si c'est le cas, le changement de niveau sera effectué. Pour se faire, nous avons besoin de l'opération *"NextLevel"*. En effet, dans notre application, le changement de niveau se traduit par un changement d'environnement.
L'observateur *"Levels"*, associe pour chaque niveau lu dans les fichiers, le niveau et son environnement. Une fois le nouvel environnement récupéré, on le transmet à tous les services interagissant avec celui-ci, notamment *"Player"*, et *"Board"*. Une nouvelle liste de gardes et une nouvelle liste de trésors sont également récupérées par le biais du nouvel environnement maintenant des observateurs sur ces différentes listes à son initialisation (voir 4.1.3).
Chaque environnement possède un portail, ce portail est caché par défaut car lors de l'affichage des cellules, le service *"Board"*, ignore le portail. Il sera affiché au joueur que s'il remplit la condition, c'est-à-dire qu'il a récupéré tous les trésors du niveau.

Les commandes utilisateur : Le service *"Board"* est chargé de récupérer les entrées clavier et les transmettre à l'Engine. C'est l'opération *"SetNextCommand"*, présente dans Engine qui permet cela en modifiant l'observateur *nextCommand*. Cette commande sera utilisée par le service *"Player"* lors de son *"Step"* pour savoir quelle action l'utilisateur souhaite effectuer.

4.1.7 Guard

Le service *"Guard"* représente les gardes du terrain. Ce sont les ennemies qui se chargeront d'infliger des dégâts au joueur afin de lui faire perdre la partie.
Ce service inclut le service *"Character"* car celui-ci a besoin de pouvoir se déplacer dans le terrain. Il offre aussi un operator *"Step"* qui provoquera le calcul du prochain mouvement du garde et le fera se déplacer en fonction du résultat de ce calcul. Les operators *"ClimbLeft"* et *"ClimbRight"* sont quant à eux utilisés pour permettre à un garde de sortir d'un trou.
La principale modification par rapport à la spécification de base du projet est que le garde ne s'occupe plus du calcul de la prochaine action à réaliser pour atteindre le joueur. Cette modification est justifiée par notre volonté d'offrir au garde une intelligence artificielle améliorée. Ainsi, le calcul de la prochaine action à réaliser est délégué à une instance du service *PathResolver*. Service que nous allons décrire dans la prochaine partie de cette section.

4.1.8 PathResolver

PathResolver permet le calcul de plus court chemin dans le terrain. Il est notamment utilisé par les gardes pour le calcul de leur plus court chemin respectif. Il est aussi utilisé pour savoir si un terrain est jouable ou non. Vous trouverez plus d'informations sur ce service dans la partie 4.1.8.

4.2 Les extensions développées

4.2.1 Intelligence améliorée des gardes

L'intelligence artificielle des gardes est limitée dans la spécification initiale du projet. Celle-ci ne permet pas aux gardes d'atteindre à chaque fois le joueur lorsque cela est possible. Afin d'offrir aux gardes une intelligence artificielle améliorée, le service PathResolver a été réalisé. Ce service permet aux gardes d'obtenir un plus court chemin entre leurs positions actuelles et la position du joueur. Avec la réalisation de ce nouveau service, tous les calculs pour l'obtention de la prochaine action à réaliser par un garde sont délégués au *PathResolver* lui-même. Le *PathResolver* calculera le graphe du terrain ainsi que le plus court chemin dans ce graphe pour aller de la position du garde à celle du joueur.
Ainsi, les gardes prendront toujours un chemin optimal pour rejoindre le joueur. Si aucun chemin ne permet d'atteindre le joueur, les gardes resteront immobiles

4.2.2 Changement de niveaux, et lecture des environnements de jeu dans des fichiers

Dans le cadre d'une *extension* permettant de changer de niveaux dans le jeu, nous avons réfléchi à une solution pour stocker les différents niveaux et pouvoir les *"charger"* au lancement du jeu.
La solution la plus pertinente fut la lecture de fichiers. En effet, les niveaux sont stockés dans des fichiers et définissent :

- Le nombre de trésors à récupérer
- Le nombre de gardes
- Les différentes natures des cases

Une fois le fichier lu, nous obtenons un niveau complet.
Une ligne d'un fichier est composée de chaque case séparée par des virgules ",". Nous avons utilisé des abréviations pour définir les cellules des trésors : *"TRS"*, les cellules initiales des gardes, *"GDR"*, et celles des portails permettant l'accès aux autres niveaux : *"PRT" (à noté que les portails n'apparaissent qu'une fois que tout les trésors du niveau courant ont été récupérés).*
Voici un exemple des lignes dans nos fichiers :

EMP,EMP,EMP,EMP,EMP,EMP,EMP,EMP,**GRD**,EMP
HDR,HDR,HDR,HDR,HDR,HDR,HDR,HDR,PLT,PLT
...

Dans nos spécifications, nous nous assurons que la CellNature(col, line), d'une cellule donnée, correspond bien à celle décrite dans le fichier à la ligne "line", et la colonne "col". Ainsi pour vérifier la bonne correspondance des cellules, nous spécifions comme ceci (exemple de lecture de PLT dans le fichier) :

Content(File, cell, line) = "PLT" $\implies$
Environnement::CellNature(Map::Get(Levels(CreateEnvironnements(E, wdt, hgt)), nbNiveau), cell, line) = "PLT"

Nous avons dû modifier la spécification fournie en correction pour le service *"Engine"*. Désormais nous n'avons plus besoin d'une liste de gardes et de trésors, car nous les obtenons à partir des fichiers.
De plus une opération "*NextLevel*" a également été ajoutée et spécifiée, permettant le changement de niveau et la propagation du nouvel environnement à tous les services ayant un observateur sur l'environnement courant.
Ce mécanisme de fichier permet efficacement l'édition de niveaux proposée en extensions dans le sujet du projet. Les niveaux créés par un utilisateur doivent être jouables et sont donc soumis à une vérification sur leur jouabilité ou non (voir 4.3.3).

4.2.3 Attaque à distance

Une autre extension développée est *l'attaque à distance*. Nous avons créé un service *"Attack"* (voir 2.10) qui offre des observateurs sur la position de l'attaque et sa direction.
Dans nos spécifications, nous nous assurons qu'à chaque *"Step"*, la coordonnée x de l'attaque évolue en fonction de sa direction ($LEFT$ ou $RIGHT$).
De plus nous avons spécifié notre service de sorte à ce que l'attaque disparaisse une fois qu'elle touche un garde, un PLT, un MTL, ou qu'elle sort des limites de l'environnement dans lequel elle évolue. Nous avons donc dû apporter des modifications au service *"Player"* car il a fallu borner l'utilisation d'attaque par niveaux du jeu. Ainsi le joueur peut utiliser aux maximum 3 attaques par niveau. Une opération pour diminuer son nombre d'attaques restantes a également été ajoutée. À chaque passage de niveau, nous avons spécifié que son compteur est remis à 3.
Des modifications de la spécification de Engine ont également été faites. Nous avons ajouté un observateur, une liste, contenant les attaques lancées, et une opération permettant d'ajouter une attaque à cet observateur. Cette liste d'attaques est mise à jour à chaque *step* de l'Engine en fonction de si l'attaque touche un garde ou un PLT/MTL.

4.2.4 Gestion des scores et de la vie

Pour un bon fonctionnement du jeu, nous avons ajouté des spécifications sur le nombre de vies que possède le joueur ainsi que son score. Les services qui ont subi des modifications son *"Player"* et *"Engine"*.
Dans le service *"Player"*, nous avons ajouté un *observateur* sur la vie, du joueur et une *opération* pour diminuer de 1 ses points de vie.
Engine a subi des modifications pour gérer le score. Il s'incrémente à chaque fois qu'un trésor est récupéré par le joueur. Un *observateur "Score"* a donc été ajouté et spécifié.

4.3 Quelques choix de spécifications

Afin de ne pas surspécifier, et de rendre la spécification à la fois lisible et compréhensible, certains choix ont été opérés. Dans cette partie nous décrirons de manière détaillée ces différents choix. Nous parlerons notamment des références à des services annexes qui ont été faite dans la spécification. Nous décrirons ensuite plus en détail la notion de *ShortestPath* utilisé dans le projet, ainsi que notre spécification définissant si un terrain est jouable ou non.

4.3.1 Des références à des services annexes

Durant la réalisation de la spécification du projet, nous avons fait le choix d'omettre la spécification de certains services supposant que ces services existaient déjà et que les décrire n'aurait fait qu'alourdir la spécification.
Les principaux services pour lesquels nous avons émis l'hypothèse de leur existence font référence à des structures de données. Ainsi, nous avons dans la spécification utilisé les services suivants sans pour autant décrire formellement leurs spécifications respectives :

- Array : Description du service d'un tableau, permet d'accéder à des éléments de manière indicée avec la notation *"crochet"*. La taille du tableau est aussi accessible avec l'observator Length(Array).
- List : Description du service de liste, permet d'accéder à des éléments de manière indicée avec l'observator Get(List, indice). Sa taille est aussi accessible avec l'observator Length(List). Une liste permet aussi de savoir si un élément est présent dans celle-ci avec l'observator Contains(List, element).
- Map : Le service Map représente une table d'association. Parmi ses observators, nous pouvons citer Keys(Map) qui renvoie l'ensemble des clés stockées dans la table et Get(Map, key) qui renvoie la valeur associée à la clé dans la table.
- File : Ce service est le seul service non-spécifié que nous avons utilisé ne représentant pas une structure de données, il représente un service offrant une manipulation de fichiers. Parmi ces observators, nous pouvons citer Lines(File), qui renvoie une liste contenant les différentes lignes du fichier.

L'hypothèse de ces différents services a pu être effectuée car ceux-ci sont inclus dans la bibliothèque de Java.

4.3.2 PathResolver et la notion de plus court chemin

Un des services ayant demandé le plus de réflexion dans la réalisation de sa spécification est le service *PathResolver*. Comme décrit plus haut, ce service est un service permettant de calculer des plus courts chemins dans un environnement donné.
La notion de plus court chemin est une notion mathématique assez complexe. Ici le plus court chemin doit correspondre à celui dans un graphe orienté non-pondéré.
Afin de définir s'il existe un plus court chemin dans notre terrain entre le joueur et un garde donné en paramètre, nous avons choisi de réaliser la spécification suivante :
$\exists$ ShortestPath($guardNode$, $playerNode$) $\in$ Graphe(P) $\iff$
Soit g, le garde représenté par $guardNode$ dans le graphe
Soit $currentNode \stackrel{\text{def}}{=} playerNode$
tant que $currentNode \neq guardNode$
$currentNode \stackrel{\text{def}}{=}$ ShortestPathToPlayer(P, g)[$currentNode$]

Ici, nous cherchons à décrire que l'existence d'un plus court chemin dans notre graphe, revient à dire qu'il existe dans le résultat de "ShortestPathToPlayer(P, g)", un "chemin" allant du joueur jusqu'au garde concerné. Ici, "ShortestPathToPlayer(P, g)" nous renvoie donc un tableau indiquant le prédécesseur d'un noeud dans le chemin retourné. Ainsi, si en allant de prédécesseur en prédécesseur à partir du joueur nous arrivons à atteindre le garde, alors cela signifie qu'il existe un plus court chemin entre le garde et le joueur.
Dans la suite des spécifications de *PathResolver*, si nous avions besoin de décrire si un plus court chemin existait entre deux éléments de notre terrain, nous avons choisi d'utiliser la formule suivante pour plus de lisibilité :
$\exists$ ShortestPath($srcNode$, $dstNode$) $\in$ Graphe(P).

4.3.3 Définir si un terrain est jouable ou non

Notre spécification d'un terrain jouable est celle donnée ci-dessous. Comme décrit à la fin de la partie 4.3.2, nous nous permettons ici d'utiliser la notion de *ShortestPath*.

PlayerCanReachAllTreasuresAndPortal($treasures$, $portalX$, $portalY$) $\iff$
Soit $playerNode \stackrel{\text{def}}{=}$ NodeIdOfPosition(P, Character::Wdt(Player(P)), Character::Hgt(Player(P)))
Soit $portalNode \stackrel{\text{def}}{=}$ NodeIdOfPosition(P, $portalX$, $portalY$)
$\exists$ ShortestPath($playerNode$, $portalNode$) $\in$ Graphe(P)

and forall t in $treasures$
Soit $treasureNode \stackrel{\text{def}}{=}$ NodeIdOfPosition(P, GameObject::Wdt(t), GameObject::Hgt(t))
$\exists$ ShortestPath($playerNode$, $treasureNode$) $\in$ Graphe(P)
and $\exists$ ShortestPath(portalNode, treasureNode) $\in$ Graphe(P)
and $\exists$ ShortestPath(treasureNode, portalNode) $\in$ Graphe(P)

and forall t in $treasures$
Soit $srcNode \stackrel{\text{def}}{=}$ NodeIdOfPosition(P, GameObject::Wdt(t), GameObject::Hgt(t))
forall $t2$ in $treasures$
Soit $dstNode \stackrel{\text{def}}{=}$ NodeIdOfPosition(P, GameObject::Wdt($t2$), GameObject::Hgt($t2$))
$\exists$ ShortestPath($srcNode$, $dstNode$) $\in$ Graphe(P)

Avec cette spécification, nous pouvons voir qu'un terrain jouable se doit de remplir beaucoup de condition en matière de chemin réalisable entre les différents éléments du terrain.

Premièrement, pour qu'un terrain soit jouable, il faut que le joueur puisse atteindre le portail du terrain, sinon il lui est impossible de passer au niveau suivant.
Il faut aussi que le joueur puisse atteindre chaque trésor du niveau, et que chacun de ces trésors permet un accès au portail à leur tour.
Et enfin, il est aussi nécessaire que chaque trésor permet l'accès aux autres trésors du terrain, ainsi un trésor ne sera pas un "puits" bloquant l'accès aux autres éléments du niveau.

Nous allons désormais analyser certains scénarios de tests que nous avons jugés intéressants.

4.4 Des tests intéressants

4.4.1 Les tests du service Character

La réalisation des tests du service *"Character"* a été l'une de celles demandant le plus de temps. En effet nous avons essayé d'offrir pour chaque test une version avec un environnement vide (sans garde), et une autre avec des gardes.
L'une des principales motivations de cette subdivision des tests était de nous assurer du bon fonctionnement du service *"Character"* dans un environnement vide avant de prendre en compte son comportement dans un environnement contenant des gardes. Afin de décrire plus en détail cette démarche, nous allons parler des tests 22 et 28. Ces tests concernent tous deux l'operator GoRight du service *"Character"*. Ainsi, le rôle du premier test sera d'assurer que le personnage peut bien se déplacer sur des HDRs si rien ne semble bloquer son chemin, alors que dans le deuxième test, le personnage sera bloqué par un garde déjà présent sur un des HDRs.

4.4.2 Les tests sur les terrains jouables ou non du service PathResolver

La fonctionnalité indiquant si un terrain est jouable ou non est une fonctionnalité essentielle du projet, dans cette partie nous tâcherons de nous concentrer sur certains des tests qui ont été réalisés sur cette fonctionnalité.
Les tests qui nous ont semblés être les plus intéressants concernent notamment les tests lorsque le terrain n'est pas jouable, pour ce faire nous avons réalisé quatre scénarios de tests différents(Voir tests 43, 43, 43 et 43), avec pour chacun des motifs d'injouabilités différents. Le test 43 se déroule sur un terrain dans lequel seul un trésor est injoignable par le joueur. Dans le test 43, ce n'est pas un trésor qui est inatteignable, mais le portail de fin de niveau. Dans le test 43, le joueur est entièrement bloqué et ne peut atteindre aucun des éléments du terrain. Le dernier test quant à lui (le 43) se situe sur un terrain où seul le portail est accessible par le joueur.

4.4.3 Un test du Step du service engine

L'operator *"Step"* du service Engine est une opération extrêmement importante, c'est en partie elle qui s'occupera du déroulement d'une partie et permettra au joueur de bouger son personnage. Le test dont nous allons parler ici est le test numéro 54. Ce test est assez spécial, son objectif est de s'assurer que si le joueur n'a plus de vie, alors la partie se termine sur une défaite. Pour ce faire, nous avons mis en place une boucle *tant que* appelant le *"Step"* de Engine. Ceci a pour objectif de faire s'approcher les gardes du joueur et ainsi faire en sorte que ces derniers lui infligent des dégâts.
Une fois que le joueur a perdu tous ces points de vie, nous nous assurons que le statut de la partie est bien passé à LOSS.

Dans la prochaine partie, nous ferons une courte présentation des implantations buggées réalisées dans ce projet.

4.5 Les implantations de services buggés

L'objectif de ces implantations buggées est de tester nos contrats et de voir s'ils lèvent bien des exceptions lorsque l'implantation viole le contrat du service associé.

Character : Lors d'un *BackToInitialPosition*, le x devient la hauteur initiale au lieu de la largeur initiale.

Player : Dans le step de Player, s'il est au-dessus d'un trou normalement il devrait tomber dans ce trou. L'implantation buggée elle ne le fait pas tomber dans le trou.

Guard : Lorsque le garde doit sortir d'un trou par la gauche, dans notre implantation buggée il sortira par la droite.

Engine : Si un niveau n'est pas jouable, nous ne sommes pas censés comptabiliser le nombre de trésors présents dans ce niveau comme objectif de fin de partie. L'implantation buggé compte justement les trésors des niveaux non jouables.

Attack : Si la direction de l'attaque est *"LEFT"*, l'implantation buggée fait *augmenter le "wdt"* au lieu de le réduire.

Nous allons désormais conclure ce rapport avec une présentation des principales difficultés rencontrées durant le projet.

4.6 Difficultés rencontrées

4.6.1 Spécifier/tester du graphisme

Spécifier l'utilisation des outils graphiques a été particulièrement délicat. L'outil utilisé pour la partie graphique est `Java.Swing`.
Nous avons créé un *"LodeRunnerFrame"* pour notre application. Ce dernier étend la classe *"JFrame"*. Nous avons aussi réalisé un service *"Board"* (voir 2.11) étendant JPanel pour dessiner les cellules à partir d'images. Ce service se chargera aussi de récupérer les entrées claviers de l'utilisateur. Les appels au *"Step"* de Engine se font par l'intermédiaire d'un *Timer*. Ce *Timer* est un thread chargé d'effectuer des actions dans un intervalle de temps défini. Toutes ces classes Java possèdent des dépendances entre elles, il est difficile voire impossible de les substituer par des instances de contrats.
En effet, le cas du *JFrame* auquel on peut ajouter des *JPanel* et d'autres "components" Java.Swing, est le cas le plus problématique, car dans une approche de développement par contrat des services graphiques, nous récupérons des erreurs de cast type : "*BoardContract* ne peut pas être casté en Component".
Nous avons néanmoins essayé de spécifier le service Board, notamment avec ses observateurs *"Engine"*, et *"Environnement"* et leur utilisation.
En ce qui concerne les opérations, elles ne font que de l'affichage et de la récupération d'entrée clavier, nous n'avons donc pas réussi à y ajouter des post-conditions satisfaisantes, ni des invariants pertinents.
La partie test elle aussi fut compliquée, ce qui explique le manque de tests de notre service Board.
Nous n'avons pas trouvé comment utiliser *JUnit* pour tester des résultats graphiques. Nous avons préféré nous pencher sur des services plus importants de l'application, aussi bien en terme de spécifications que de tests.

4.6.2 Le maintien de la cohérence des spécifications

Une autre difficulté durant ce projet a été le maintien de la cohérence des spécifications. Lors d'une modification dans les spécifications d'un service, nous nous devions de modifier l'ensemble des éléments du service. C'est-à-dire le contrat du service, son implantation et les scénarios de tests du service.
Cette difficulté nous a poussé à mûrement réfléchir les spécifications de nos services avant de commencer tout autres tâches (contrat, test, implantation).

4.6.3 Les bugs introduits par les contrats

Durant nos tests du projet, nous avons privilégié l'utilisation des contrats de nos services, ce choix nous a permis une plus grande rigueur et une meilleure surveillance du bon déroulement de la partie. Cependant, il arrivait que les bugs que nous trouvions durant ces tests soient provoqués par des erreurs d'implantation des contrats du service, et non des erreurs dans l'implantation du service à proprement parlé.
Ainsi, à l'instar de la réalisation des spécifications d'un service, l'implantation des contrats devait être très rigoureuse. Il était aussi important lors de la découverte d'un bug de bien relire les contrats pour nous assurer que l'erreur n'était pas introduite par le contrat.

4.6.4 Spécifier certaines propriétés complexes

Une autre difficulté majeure durant le projet a été la spécification de certaines propriétés que l'on pourrait qualifier de complexes. Nous pensons notamment à la spécification des plus courts chemins dans le service *PathResolver*. Afin d'avoir plus d'informations sur notre solution face à cette difficulté, vous pouvez vous rendre à la section numéro 4.3.2 de ce rapport.