Rapport CPS : LodeRunner

Ning Guo
Kelly Seng

8 mai 2019

Manuel d'utilisation

Le but du jeu est de récupérer les trésors présents dans le niveau tout en évitant de se faire attraper par des gardes.

Notre personnage est représenté par "@" dans ce projet. Les gardes normaux sont représentés par " !", les gardes spéciaux par "" et les trésors par " ?". Les gardes spéciaux ne peuvent pas tomber dans les trous.

Un niveau est composé de plusieurs cases qui peuvent posséder les natures suivantes :

- ' ' : plateforme
- '-' : métal
- '^' : rail
- '+' : échelle
- ' ' : vide ou trou

Le joueur dispose de 6 actions :

- 'q' : se déplacer à gauche
- 's' : se déplacer vers le bas
- 'd' : se déplacer à droite
- 'z' : se déplacer vers le haut
- 'o' : creuser un trou en bas à gauche du personnage
- 'p' : creuser un trou en bas à droite du personnage

Pour compiler le projet, il faut faire "ant compile" à la racine du projet et "ant jar" pour créer les exécutables de l'implémentation correcte et de l'implémentation buggée qui nous serviront à lancer le jeu. L'implémentation buggée fait en sorte que le jeu ne se termine jamais même lorsque les conditions de défaite sont remplies.

Le jeu possède pour l'instant 3 niveaux que l'on retrouve dans le package loderunner.map : **lvl1**, **mapTestEngine** et **mapTestPlayer**. Nous lançons le jeu avec lvl1 en faisant :

java -jar loderunner.jar lvl1.txt (version sans bug)

java -jar loderunnerbug.jar lvl1.txt (version avec bug)

Il est également possible d'importer ce projet sous eclipse et de paramétrer les arguments lancés lors de l'exécution des classes MainClass.java et MainBugClass.java

Nous pouvons lancer les tests avec : ant testNom .Les tests qui ont été réalisés sont :

- Service
- EditableScreen
- Environment
- Character
- Player
- Guard
- Engine

Par exemple, pour tester la spécification du service Screen, on fait : ant testScreen

Spécification formelle complète

Ecran

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** $0 <$ `h` **and** $0 <$ `w`
Operators : `Dig` : [Screen] × int × int → [Screen]
pre `Dig(S,x,y)` **requires** `CellNature(S,x,y)` = **PLT**
`Fill` : [Screen] × int × int → [Screen]
pre `Dig(S,x,y)` **requires** `CellNature(S,x,y)` = **HOL**
Observations :
`[init]` : `Height(init(h,w))` = `h`
`Width(init(h,w))` = `w`
forall (u, v) **in** `[0;Width(S)[`× `[0;Height(S)[`, `CellNature(init(h,w),x,y)` = **EMP**
`[Dig]` : `CellNature(Dig(S,x,y)),x,y` = **HOL**
forall (u, v) **in** `[0;Width(S)[` × `[0;Height(S)[`,
(`x` ≠ `u` **or** `y` ≠ `v`) **implies** `CellNature(Dig(S,x,y)),u,v)` = `CellNature(u,v)`
`[Fill]` : `CellNature(Fill(S,x,y),x,y)` = **PLT**
forall (u, v) **in** `[0;Width(S)[`× `[0;Height(S)[`,
(`x` ≠ `u` **or** `y` ≠ `v`) **implies** `CellNature(Fill(S,x,y)),u,v)` = `CellNature(u,v)`

Ecran editable

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, y) **in** `[0;Width(S)[`× `[0;Height(S)[`, `CellNature(S,x,y)` ≠ **HOL**
and forall x **in** `[0;Width(S)[`, `CellNature(S,x,0)` = **MTL**
`[init]` : **forall** u **in** `[0;Width(S)[`, CellNature(S,u,0) = **MTL**
`[SetNature]` : `CellNature(SetNature(S,x,y,C)),x,y` = C
forall (u, v) **in** `[0;Width(S)[`× `[0;Height(S)[`,
(`x` ≠ `u` **or** `y` ≠ `v`) **implies** `CellNature(SetNature(S,x,y,C)),u,v)` = `CellNature(u,v)`

Environnement

Service : Environment **includes** Screen

Observators : `CellContent` : int × int → Set{Character + Item}
pre `CellContent(E,x,y)` **requires** $0 \leq$ `y` $<$ `Height(E)` **and** $0 \leq$ `x` $<$ `Width(E)`

Constructors : `init` : EditableScreen → Environment

Observations :

`[invariant]` : **forall** (`x`, `y`) **in** `[0;Width(E)[`× `[0;Height(E)[`,
forall Character `c1`, `c2` **in** `CellContent(E,x,y)`^{2}, `c1` = `c2`
forall (`x`, `y`) **in** `[0;Width(E)[`× `[0;Height(E)[`,
`CellNature(E,x,y)` **in** {**MTL**, **PLR**} **implies** `CellContent(x,y)` = ∅
forall (`x`, `y`) **in** `[0;Width(E)[`× `[0;Height(E)[`,
exists Treasure `t` **in** `CellContent(E,x,y)`
implies (`CellNature(E,x,y)` = **EMP and** `CellNature(E,x,y-1)` **in** {**PLT**, **MTL**})

`[init]` : **forall** (`x`, `y`) **in** `[0;Width(E)[`× `[0;Height(E)[`,
`CellNature(init(S),x,y)` = EditableScreen : :`CellNature(S,x,y)`

Personnage

Service : Character

Observators : **const** `Envi` : [Character] → Environment
`Hgt` : [Character] → int
`Wdt` : [Character] → int

Operators : `init` : Screen × int × int → [Character]
pre init(S,x,y) **requires** Environment : :CellNature(S,x,y) = **EMP**
`GoLeft` : [Character] → [Character]
`GoRight` : [Character] → [Character]
`GoUp` : [Character] → [Character]
`GoDown` : [Character] → [Character]

Observations :

`[invariant]` : Environment : :CellNature(Envi(C),Wdt(C),Hgt(C)) **in** {**EMP**, **HOL**, **LAD**, **HDR**}
exists Character `x` **in** Environment : :CellContent(Envi(C),Wdt(C),Hgt(C)) **implies** `x` = `C`

`[init]` : `Hgt(init(s,h,w))` = `h`
`Wdt(init(s,h,w))` = `w`
`Envi(init(s,h,w))` = `s`

`[GoLeft]` : Hgt(GoLeft(C)) = Hgt(C)
Wdt(C) = 0 **implies** Wdt(GoLeft(C)) = Wdt(C)
Environment : :CellNature(Envi(C),Wdt(C)-1,Hgt(C)) **in** {**MTL**, **PLT** }
implies Wdt(GoLeft(C)) = Wdt(C)
Environment : :CellNature(Envi(C),Wdt(C),Hgt(C)) **not in** {**LAD**, **HDR**}
and Environment : :CellNature(Envi(C),Wdt(C),Hgt(C)-1) **not in** {**PLT**, **MTL**, **LAD** }
and not exists Character c **in** Environment : :CellContent(Envi(C),Wdt(C),Hgt(C)-1)
implies Wdt(GoLeft(C)) = Wdt(C)
exists Character c **in** Environment : :CellContent(Envi(C),Wdt(C)-1,Hgt(C))
implies Wdt(GoLeft(C)) = Wdt(C)
(Wdt(C) ≠ 0) **and** Environment : :CellNature(Envi(C),Wdt(C)-1,Hgt(C)) **not in** {**MTL**, **PLT** }
and (Environment : :CellNature(Envi(C),Wdt(C),Hgt(C)) **in** {**LAD**, **HDR**}
or Environment : :CellNature(Envi(C),Wdt(C),Hgt(C)-1) **in** {**PLT**, **MTL**, **LAD**}
or exists Character c **in** Environment : :CellContent(Envi(C),Wdt(C),Hgt(C)-1))
and not (**exists** Character c **in** Environment : :CellContent(Envi(C),Wdt(C)-1,Hgt(C)))
implies Wdt(GoLeft(C)) = Wdt(C)-1

`[GoRight]` : Hgt(GoRight(C)) = Hgt(C)
Wdt(C) = Environement : :Width(Envi(C)) **implies** Wdt(GoRight(C)) = Wdt(C)
Environment : :CellNature(Envi(C),Wdt(C)+1,Hgt(C)) **in** {**MTL**, **PLT** }
implies Wdt(GoRight(C)) = Wdt(C)
Environment : :CellNature(Envi(C),Wdt(C),Hgt(C)) **not in** {**LAD**, **HDR**}
and Environment : :CellNature(Envi(C),Wdt(C),Hgt(C)-1) **not in** {**PLT**, **MTL**, **LAD** }
and not exists Character c **in** Environment : :CellContent(Envi(C),Wdt(C),Hgt(C)-1)
implies Wdt(GoRight(C)) = Wdt(C)
exists Character c **in** Environment : :CellContent(Envi(C),Wdt(C)+1,Hgt(C))
implies Wdt(GoRight(C)) = Wdt(C)
(Wdt(C) ≠ Environement : :Height(Envi(C))
and Environment : :CellNature(Envi(C),Wdt(C)+1,Hgt(C)) **not in** {**MTL**, **PLT** }
and (Environment : :CellNature(Envi(C),Wdt(C),Hgt(C)) **in** {**LAD**, **HDR**}
or Environment : :CellNature(Envi(C),Wdt(C),Hgt(C)-1) **in** {**PLT**, **MTL**, **LAD**}
or exists Character c **in** Environment : :CellContent(Envi(C),Wdt(C),Hgt(C)-1))
and not (**exists** Character c **in** Environment : :CellContent(Envi(C),Wdt(C)+1,Hgt(C)))
implies Wdt(GoRight(C)) = Wdt(C)+1

`[GoUp]` : Wdt(GoUp(C)) = Wdt(C)
Hgt(C) = Environment : :Height(Envi(C)) **implies** Hgt(GoUp(C)) = Hgt(C)
Environment : :CellNature(Envi(C),Wdt(C),Hgt(C)) ≠ **LAD implies** Hgt(GoUp(C)) = Hgt(C)
Environment : :CellNature(Envi(C),Wdt(C),Hgt(C)+1) ≠ **LAD**
implies Hgt(GoUp(C)) = Hgt(C)
exists Character c **in** Environment : :CellContent(Envi(C),Wdt(C),Hgt(C)+1) **implies**
Hgt(GoUp(C)) = Hgt(C)
Hgt(C) ≠ Environment : :Height(Envi(C))
and (Environment : :CellNature(Envi(C),Wdt(C),Hgt(C)+1 = **LAD**
Environment : :CellNature(Envi(C),Wdt(C),Hgt(C)+1 **not in** **{ MTL,PLT }**
and Environment : :CellNature(Envi(C),Wdt(C),Hgt(C) = **LAD**
and not exists Character c **in** Environment : :CellNature(Envi(C),Wdt(C),Hgt(C)+1
implies Wdt(GoUp(C)) = Hgt(C)+1

`[GoDown]` : Wdt(GoDown(C)) = Wdt(C)
Hgt(C) = 0 **implies**
Hgt(GoDown(C)) = Hgt(C)
Environment : :CellNature(Envi(C),Wdt(C),Hgt(C)-1) in **{ MTL,PLT }**
implies Hgt(GoDown(C)) = Hgt(C)
exists Character c **in** Environment : :CellContent(Envi(C),Wdt(C),Hgt(C)-1)
implies Hgt(GoDown(C)) = Hgt(C)
Hgt(C) ≠ 0
and Environment : :CellNature(Envi(C),Wdt(C),Hgt(C)-1) in **{ MTL,PLT }**
and not exists Character c **not in** Environment : :CellNature(Envi(C),Wdt(C),Hgt(C)-1
implies Wdt(GoDown(C)) = Hgt(C)-1

Garde

Service : Guard **includes** Character

Observators : **const** `Id` : [Guard] → int
`Behaviour` : [Guard] → Move
`Target` : [Guard] → Character
`TimeInHole` : [Guard] → int
`IsSpecial` : [Guard] → boolean

Operators : `init` : Screen × int × int × Player× boolean → [Guard]
pre `init(S,x,y,p,b)` **requires** Environment : :CellNature(S,x,y) = **EMP and** p ≠ ∅
`ClimbLeft` : [Guard] → [Guard]
pre `ClimbLeft(G)` **requires** `Environment::CellNature(Envi(G),Hgt(G),Wdt(G))` = **HOL**
`ClimbRight` : [Guard] → [Guard]
pre `ClimbRight(G)` **requires** `Environment::CellNature(Envi(G),Hgt(G),Wdt(G))` = **HOL**
`Step` : [Guard] → [Guard]

Observations :

[invariant] : Environment : :CellNature(Envi(G),Wdt(G),Hgt(G)) = **LAD**
and Hgt(G) < Character : :Hgt(Target(G))
and (Environment : :CellNature(Envi(G),Wdt(G),Hgt(G)-1) **not in** {**PLT**, **MTL** }
or exists Character c **in** Environment : :CellContent(Envi(G),Wdt(G),Hgt(G)-1)
implies |Environment : :Hgt(Target(G)) - Hgt(G)| <
|Environment : :Wdt(Target(G)) - Wdt(G))|)
implies Behaviour(G) = **Up**
Environment : :CellNature(Envi(G),Wdt(G),Hgt(G)) = **LAD**
and Hgt(G) > Character : :Hgt(Target(G))
and (Environment : :CellNature(Envi(G),Wdt(G),Hgt(G)-1) **not in** {**PLT**, **MTL** }
or exists Character c **in** Environment : :CellContent(Envi(G),Wdt(G),Hgt(G)-1)
implies |Environment : :Hgt(Target(G)) - Hgt(G)| <
|Environment : :Wdt(Target(G)) - Wdt(G))|)
implies Behaviour(G) = **Down**
Environment : :CellNature(Envi(G),Wdt(G),Hgt(G)) = **LAD**
and Wdt(G) < Character : :Wdt(Target(G))
and (Environment : :CellNature(Envi(G),Wdt(G),Hgt(G)-1) **not in** {**PLT**, **MTL** }
or exists Character c **in** Environment : :CellContent(Envi(G),Wdt(G),Hgt(G)-1)
or (Environment : :CellNature(Envi(G),Wdt(G),Hgt(G)-1) = **HOL and** IsSpecial))
implies |Environment : :Hgt(Target(G)) - Hgt(G)| >
|Environment : :Wdt(Target(G)) - Wdt(G))|)
implies Behaviour(G) = **Left**
Environment : :CellNature(Envi(G),Wdt(G),Hgt(G)) = **LAD**
and Wdt(G) > Character : :Wdt(Target(G))
and (Environment : :CellNature(Envi(G),Wdt(G),Hgt(G)-1) **not in** {**PLT**, **MTL** }
or exists Character c **in** Environment : :CellContent(Envi(G),Wdt(G),Hgt(G)-1)
or (Environment : :CellNature(Envi(G),Wdt(G),Hgt(G)-1) = **HOL and** IsSpecial))
implies |Environment : :Hgt(Target(G)) - Hgt(G)| <
|Environment : :Wdt(Target(G)) - Wdt(G))|)
implies Behaviour(G) = **Right**
Environment : :CellNature(Envi(G),Wdt(G),Hgt(G)) **in** **{HOL,HDR} or**
Environment : :CellNature(Envi(G),Wdt(G),Hgt(G)-1) **in {MTL,PLT,LAD} or**
or exists Character c **in** Environment : :CellContent(Envi(G),Wdt(G),Hgt(G)-1))
and Wdt(G) > Character : :Wdt(Target(G))
and (Environment : :CellNature(Envi(G),Wdt(G)-1,Hgt(G)) **not in** {**PLT**, **MTL** }
or not exists Character c **in** Environment : :CellContent(Envi(G),Wdt(G)-1,Hgt(G))
implies Behaviour(G) = **Left**
Environment : :CellNature(Envi(G),Wdt(G),Hgt(G)) **in** **{HOL,HDR} or**
Environment : :CellNature(Envi(G),Wdt(G),Hgt(G)-1) **in {MTL,PLT,LAD} or**
or exists Character c **in** Environment : :CellContent(Envi(G),Wdt(G),Hgt(G)-1))
and Wdt(G) > Character : :Wdt(Target(G))
and (Environment : :CellNature(Envi(G),Wdt(G)+1,Hgt(G)) **not in** {**PLT**, **MTL** }
or not exists Character c **in** Environment : :CellContent(Envi(G),Wdt(G)+1,Hgt(G))
implies Behaviour(G) = **Right**

[invariant] : Environment : :CellNature(Envi(G),Wdt(G),Hgt(G)) **in {HOL,HDR} or**
Environment : :CellNature(Envi(G),Wdt(G),Hgt(G)-1) **in {MTL,PLT,LAD} or**
or exists Character c **in** Environment : :CellContent(Envi(G),Wdt(G),Hgt(G)-1))
and Wdt(G) = Character : :Wdt(Target(G))
implies Behaviour(G) = **Neutral**
Environment : :CellNature(Envi(G),Wdt(G),Hgt(G)) = **LAD**
and Hgt(G) < Character : :Hgt(Target(G))
implies Behaviour(G) = **Up**
Environment : :CellNature(Envi(G),Wdt(G),Hgt(G)) = **LAD**
and Hgt(G) > Character : :Hgt(Target(G))
implies Behaviour(G) = **Down**
Environment : :CellNature(Envi(G),Wdt(G),Hgt(G)) = **LAD**
and Hgt(G) = Character : :Hgt(Target(G))
implies Behaviour(G) = **Neutral**

[init] : Target(G) = p

[ClimbLeft] : Wdt(G) = 0 **implies** Wdt(ClimbLeft(G)) = Wdt(G) **and** Hgt(ClimbLeft(G)) = Hgt(G)
Environment : :CellNature(Envi(G),Wdt(G)-1,Hgt(G) +1) **in** {**MTL**, **PLT** }
implies Wdt(ClimbLeft(G)) = Wdt(G) **and** Hgt(ClimbLeft(G)) = Hgt(G)
exists Character c **in** Environment : :CellContent(Envi(G),Wdt(G)-1,Hgt(G)+1)
implies Wdt(ClimbLeft(G)) = Wdt(G) **and** Hgt(ClimbLeft(G)) = Hgt(G)
Wdt(G) ≠ 0 **and** Environment : :CellNature(Envi(G),Wdt(G)-1,Hgt(G)+1) **not in** {**MTL**, **PLT** }
and not exists Character c **in** Environment : :CellContent(Envi(G),Wdt(G)-1,Hgt(G)+1)
implies Wdt(ClimbLeft(G)) = Wdt(G)-1 **and** Hgt(ClimbLeft(G)) = Hgt(C)+1

[ClimbRight] : Wdt(G) = Environment : :Width(G)
implies Wdt(ClimbRight(G)) = Wdt(G) **and** Hgt(ClimbRight(G)) = Hgt(G)
Environment : :CellNature(Envi(G),Wdt(G)+1,Hgt(G) +1) **in** {**MTL**, **PLT** }
implies Wdt(ClimbRight(G)) = Wdt(G) **and** Hgt(ClimbRight(G)) = Hgt(G)
exists Character c **in** Environment : :CellContent(Envi(G),Wdt(G)+1,Hgt(G)+1)
implies Wdt(ClimbRight(G)) = Wdt(G) **and** Hgt(ClimbLeft(G)) = Hgt(G)
Wdt(G) ≠ Environment : :Width(G) **and**
Environment : :CellNature(Envi(G),Wdt(G)-1,Hgt(G)+1) **not in** {**MTL**, **PLT** }
and not exists Character c **in** Environment : :CellContent(Envi(G),Wdt(G)+1,Hgt(G)+1)
implies Wdt(ClimbRight(G)) = Wdt(G)+1 **and** Hgt(ClimbLeft(G)) = Hgt(G)+1

[Step] : **WillFall(G) defined by** (Environment : :CellNature(Envi(G),Wdt(G),Hgt(G)-1) **in** {**HOL**, **EMP** }
and not exists Character c **in** Environment : :CellContent(Envi(G),Wdt(G),Hgt(G)-1)
and Environment : :CellNature(Envi(G),Wdt(G),Hgt(G)) **not in** {**LAD**, **HDR** }
and not(Environment : :CellNature(Envi(G),Wdt(G),Hgt(G)-1) = **HOL and** IsSpecial))
WillIncTime(G) defined by (Environment : :CellNature(Envi(G),Wdt(G),Hgt(G)) = HOL **and**
TimeInHole(G) < 5)
WillLeft(G) defined by (Environment : :CellNature(Envi(G),Wdt(G),Hgt(G)) = HOL **and**
TimeInHole(G) = 5 **and** Behavior(G) = Left)
WillRight(G) defined by (Environment : :CellNature(Envi(G),Wdt(G),Hgt(G)) = HOL **and**
TimeInHole(G) = 5 **and** Behavior(G) = Right)
WillNeutral(G) defined by (Environment : :CellNature(Envi(G),Wdt(G),Hgt(G)) = HOL **and**
TimeInHole = 5 **and** Behavior = Neutral)
WillGoLeft(G) defined by Behavior(G) = Left **and**
(Wdt(G) ≠ 0) **and** Environment : :CellNature(Envi(G),Wdt(P)-1,Hgt(G)) **not in** {**MTL**, **PLT**}
and (Environment : :CellNature(Envi(G),Wdt(G),Hgt(G)) **in** {**LAD**, **HDR**}
or Environment : :CellNature(Envi(G),Wdt(G),Hgt(G)-1) **in** {**PLT**, **MTL**, **LAD**}
or exists Character c **in** Environment : :CellContent(Envi(G),Wdt(G),Hgt(G)-1))
and not (**exists** Character c **in** Environment : :CellContent(Envi(G),Wdt(G)-1,Hgt(G)))
WillGoRight(G) defined by Behavior(G) = Right **and**
(Wdt(P) ≠ Environement : :Height(Envi(P))
Environment : :CellNature(Envi(P),Wdt(P)+1,Hgt(P)) **not in** {**MTL**, **PLT**}
and (Environment : :CellNature(Envi(P),Wdt(P),Hgt(P)) **in** {**LAD**, **HDR**}
or Environment : :CellNature(Envi(P),Wdt(P),Hgt(P)-1) **in** {**PLT**, **MTL**, **LAD**}
or exists Character c **in** Environment : :CellContent(Envi(P),Wdt(P),Hgt(P)-1))
and not (**exists** Character c **in** Environment : :CellContent(Envi(P),Wdt(P)+1,Hgt(P)))
WillGoUp(G) defined by Behavior(G) = Up **and**
Hgt(P) ≠ Environment : :Height(Envi(P))
and (Environment : :CellNature(Envi(P),Wdt(P),Hgt(P)+1 = **LAD**
Environment : :CellNature(Envi(P),Wdt(P),Hgt(P)+1 **not in** **{ MTL,PLT }**
and Environment : :CellNature(Envi(P),Wdt(P),Hgt(P)) = **LAD**
and not exists Character c **in** Environment : :CellNature(Envi(P),Wdt(P),Hgt(P)+1)
WillGoDown(G) defined by Behavior(G) = Down
and Hgt(P) ≠ 0
and Environment : :CellNature(Envi(P),Wdt(P),Hgt(P)-1) in **{ MTL,PLT }**
and not exists Character c **not in** Environment : :CellNature(Envi(P),Wdt(P),Hgt(P)-1)
WillFall(G) implies GoDown(G)
WillTimeInc(G) implies TimeInHole(Step(G)) = TimeInHole(G) +1
WillLeft(G) implies ClimbLeft (G)
WillRight(G) implies ClimbRight (G)
WillNeutral(G) implies Hgt(Step(G)) = Hgt(G) **and** Wdt(Step(G)) = Wdt(G)
WillGoLeft(G) implies GoLeft (G)
WillGoRight(G) implies GoRight (G)
WillGoUp(G) implies GoUp (G)
WillGoDown(G) implies GoDown (G)

Joueur

Service : Player **includes** Character

Observators :
`Engine` : [Player] → Engine
`Vie` : [Player] → int
`Score` : [Player] → int

Operators :
`init` : Screen × int × int × Engine → [Player]
pre `init(S,x,y,e)` **requires** Environment : :CellNature(S,x,y) = **EMP and** e ≠ ∅
`Step` : [Player] → [Player]
`setScore` : [Player] × int → [Player]
`decreVie` : [Player] → [Player]

Observations :

[init] : **Engine(P)** = e

[setScore] : **Score(setScore(P,s))** = s

[decreVie] : **Vie(decreVie(P))** = Vie(P)-1

[Step] : **WillFall(P) defined by** (Environment : :CellNature(Envi(P),Wdt(P),Hgt(P)-1)
in {**HOL**, **EMP** }
and not exists Character c **in** Environment : :CellContent(Envi(G),Wdt(P),Hgt(P)-1)
and Environment : :CellNature(Envi(P),Wdt(P),Hgt(P)) **not in** {**LAD**, **HDR** })
WillGoLeft(P) defined by Engine : :NextCommand(Engine(P)) = Left **and**
(Wdt(P) ≠ 0) **and** Environment : :CellNature(Envi(P),Wdt(P)-1,Hgt(P)) **not in** {**MTL**, **PLT**}
and (Environment : :CellNature(Envi(P),Wdt(P),Hgt(P)) **in** {**LAD**, **HDR**}
or Environment : :CellNature(Envi(P),Wdt(P),Hgt(P)-1) **in** {**PLT**, **MTL**, **LAD**}
or exists Character c **in** Environment : :CellContent(Envi(P),Wdt(P),Hgt(P)-1))
and not (**exists** Character c **in** Environment : :CellContent(Envi(P),Wdt(P)-1,Hgt(P)))
WillGoRight(P) defined by Engine : :NextCommand(Engine(P)) = Right **and**
(Wdt(P) ≠ Environement : :Height(Envi(P))
Environment : :CellNature(Envi(P),Wdt(P)+1,Hgt(P)) **not in** {**MTL**, **PLT**}
and (Environment : :CellNature(Envi(P),Wdt(P),Hgt(P)) **in** {**LAD**, **HDR**}
or Environment : :CellNature(Envi(P),Wdt(P),Hgt(P)-1) **in** {**PLT**, **MTL**, **LAD**}
or exists Character c **in** Environment : :CellContent(Envi(P),Wdt(P),Hgt(P)-1))
and not (**exists** Character c **in** Environment : :CellContent(Envi(P),Wdt(P)+1,Hgt(P)))
WillGoUp(P) defined by Engine : :NextCommand(Engine(P)) = Up **and**
Hgt(P) ≠ Environment : :Height(Envi(P))
and (Environment : :CellNature(Envi(P),Wdt(P),Hgt(P)+1 = **LAD**
Environment : :CellNature(Envi(P),Wdt(P),Hgt(P)+1 **not in** **{ MTL,PLT }**
and Environment : :CellNature(Envi(P),Wdt(P),Hgt(P) = **LAD**
and not exists Character c **in** Environment : :CellNature(Envi(P),Wdt(P),Hgt(P)+1
WillGoDown(P) defined by Engine : :NextCommand(Engine(P)) = Down
and Hgt(P) ≠ 0
and Environment : :CellNature(Envi(P),Wdt(P),Hgt(P)-1) in **{ MTL,PLT }**
and not exists Character c **not in** Environment : :CellNature(Envi(P),Wdt(P),Hgt(P)-1
WillDigL(P) defined by Engine : :NextCommand(Engine(P)) = DigL
and (Environment : :CellNature(Envi(P),Wdt(P),Hgt(P)-1) **in** {**PLT**, **MTL**, **LAD**}
or exists Character c **in** Environment : :CellContent(Envi(P),Wdt(P),Hgt(P)-1))
and Environment : :CellNature(Envi(P),Wdt(P)-1,Hgt(P)) **not in** {**PLT**, **MTL**}
and Environment : :CellNature(Envi(P),Wdt(P)-1,Hgt(P)-1) = PLT
WillDigR(P) defined by Engine : :NextCommand(Engine(P)) = DigR
and (Environment : :CellNature(Envi(P),Wdt(P),Hgt(P)-1) **in** {**PLT**, **MTL**, **LAD**}
or exists Character c **in** Environment : :CellContent(Envi(P),Wdt(P),Hgt(P)-1))
and Environment : :CellNature(Envi(P),Wdt(P)+1,Hgt(P)) **not in** {**PLT**, **MTL**}
and Environment : :CellNature(Envi(P),Wdt(P)+1,Hgt(P)+1) = PLT
WillGoLeft(P) implies Wdt(P) = Wdt(GoLeft(P))
and Hgt(P) = Hgt(GoLeft(P))
WillGoRight(P) implies Wdt(P) = Wdt(GoRight(P))
and Hgt(P) = Hgt(GoRight(P))
WillGoUp(P) implies Wdt(P) = Wdt(GoUp(P))
and Hgt(P) = Hgt(GoUp(P))
WillGoDown(P) implies Wdt(P) = Wdt(GoDown(P))
and Hgt(P) = Hgt(GoDown(P))

[Step] : **WillDigL(P) implies** Environment : :Dig(Envi(P),Wdt(P)-1,Hgt(P)-1)
and Environment : :CellNature(Envi(P),Wdt(P)-1,Hgt(P)-1) = HOL
WillDigR(P) implies Environment : :Dig(Envi(P),Wdt(P)+1,Hgt(P)+1)
and Environment : :CellNature(Envi(P),Wdt(P)+1,Hgt(P)+1) = HOL

Engine

Service : Engine
Observators : `Env` : [Engine] → Environment
`Player` : [Engine] → Player
`Guards` : [Engine] → Set {Guard}
`Treasures` : [Engine] → Set {Item}
`Status` : [Engine] → Status
`NextCommand` : [Engine] → NextCommand
`Holes` : [Engine] → Triplet {int × int × int }
Operators : `init` : EditableScreen × int × int × List {int × int × boolean } × List{int × int } → [Engine]
pre `init(S,P,G,T)` **requires** Environment : :CellNature(S,Character : :Wdt(P),
Character : :Hgt(P)) = **EMP**
and Environment : :CellNature(S,Character : :Wdt(P), Character : :Hgt(P)-1)
in {PLT,MTL}
and forall g **in** G,
`(Environment::CellNature(S,Character::Wdt(g),Character::Hgt(g))` = **EMP**
and `Environment::CellNature(S,Character::Wdt(g),Character::Hgt(g)-1)`
in {PLT,MTL})
and forall t **in** T, `Environment::CellNature(S,Item::Wdt(t),Item::Hgt(t)))` = **EMP**
and `(Environment::CellNature(S,Item::Wdt(t),Item::Hgt(t)-1)` **in {PLT,MTL})**
and EditableScreen : :Playable(S)
`Step` : [Engine] → [Engine]
Observations :
[invariant] : Player(E) ∈ Environment : :CellContent(Env(E),Character : :Wdt(Player(E)),
Character : :Hgt(Player(E))))
forall g **in** Guards(E),
g ∈ Environment : :CellContent(Env(E),Character : :Wdt(g),Character : :Hgt(g)))
forall t **in** Treasures(E),
t ∈ Environment : :CellContent(Env(E),Character : :Wdt(t),Character : :Hgt(t)))
[init] : Status(E) = Playing
Character : :Wdt(Player(E)) = x **and** Character : :Hgt(Player(E)) = y
forall (t, ct) **in** Treasures`(E)` × T, Item : :Wdt(t) = Pair : :L(ct) **and**
Item : :Hgt(t) = Pair : :R(ct)
forall (g, cg) **in** Guards`(E)` × T, Character : :Wdt(g) = Pair : :L(ct) **and**
Character : :Hgt(g) = Pair : :R(cg)
[Step] : **exists** Treasures t **in** Environment : :CellContent(Env(E),Character : :Wdt(Player(E)),
Character : :Hgt(Player(E))) **implies** t **not exists in**
Environment : :CellContent(Env(E),Character : :Wdt(Player(E)),Character : :Hgt(Player(E)))
Treasures(E) = ∅ **implies** Status(E)=Win
Player : :Vie(Player(E)) = 0 **implies** Status(E) = Loss
Guard g ∈ (Environment : :CellContent(Env(E),Character : :Wdt(Player(E)),
Character : :Hgt(Player(E)))) **implies** Status(E) = Loss

Description formelle des tests MBT effectués

Tests Screen

On note qu'il n'est pas possible d'obtenir des tests positifs sur les préconditions de Dig et Fill car on ne peut pas modifier la nature d'une case dans ce service. Ces tests se feront dans ceux de EditableScreen.

Tests préconditions

Cas de test : Screen : :test1InitPrePositif
Description : On initialise un Screen de taille (6,6)

— Condition initiale : ∅
— Opération : S0 = init(6,-6)
— Oracle :
Pas d'exception levée

Cas de test : Screen : :test1InitPreNegatif
Description : On initialise un Screen de taille (5,-5)

— Condition initiale : ∅
— Opération : S0 = init(5,-5)
— Oracle :
Exception levée

Cas de test : Screen : :test1CellNaturePrePositif Description : On récupère la nature de la case à (0,0)

— Condition initiale : S0 = init(6,6)
— Opération : S1 = CellNature(S0,0,0)
— Oracle :
Pas d'exception levée

Cas de test : Screen : :test1CellNaturePreNegatif Description : On récupère la nature de la case à (0,8) qui n'existe pas

— Condition initiale : S0 = init(6,6)
— Opération : S1 = CellNature(S0,0,8)
— Oracle :
Exception levée

Cas de test : Screen : :testDigPreNegatif
Description : On "dig" la case à (5,0)

— Condition initiale : S0 = init(6,6)
— Opération : S1 = Dig(S0,5,0)
— Oracle :
Exception levée

Cas de test : Screen : :testFillPreNegatif
Description : On "fill" la case à (1,1)

— Condition initiale : S0 = init(6,6)

— Opération : S1 = Fill(S0,1,1)

— Oracle :
Exception levée

Test Transitions

Cas de test : Screen : :test1Init
Description : On initialise un Screen de taille (6,6)

— Condition initiale : $\emptyset$

— Opération : S0 = init(6,6)

— Oracle :
Height(S0) = 6
Width(S0) = 6
$\forall x \in [0; getWidth(S0)], \forall y \in [0; getHeight(S0)]$, getCellNature(S0,x,y) == EMP

Cas de test : Screen : :testCellNature
Description : On vérifie que la case initialisé à (0,0) est vide.

— Condition initiale : S0 = init(6,6)

— Opération : S1 = CellNature(S0,0,0)

— Oracle :
CellNature(S1,0,0) = EMP

Tests EditableScreen

Tests préconditions

Cas de test : EditableScreen : :test1SetNaturePositif
Description : On modifie la case (0,5) en LAD.

— Condition initiale : S0 = init(6,6)

— Opération : S1 = setNature(S0,0,5,LAD)

— Oracle :
Pas d'exception levée

Cas de test : EditableScreen : :test1SetNatureNegatif
Description : On modifie la case (-6,5) à LAD.

— Condition initiale : S0 = init(6,6)

— Opération : S1 = setNature(S0,-6,5,LAD)

— Oracle :
Exception levée

Cas de test : EditableScreen : :test1DigPrePositif
Description : On "dig" la case (0,1).

— Condition initiale : S0 = setNature(init(6,6),0,1,PLT)

— Opération : S1 = Dig(S0,0,1)

— Oracle :
Pas d'exception levée

Cas de test : Screen : :test1FillPrePositif
Description : On "fill" la case (0,0).

— Condition initiale : S0 = setNature(init(6,6),0,1,HOL)

— Opération : S1 = Fill(S0,0,1)

— Oracle :
Pas d'exception levée

Tests Transition

Cas de test : EditableScreen : :Playable1
Description : On initialise l'écran à la taille (6,6).

— Condition initiale : ∅

— Opération : S1 = init(6,6)

— Oracle :
Playable(S1) = true

Cas de test : EditableScreen : :Playable2
Description : On initialise l'écran à la taille (6,6) et on met la case (1,0) à HDR.

— Condition initiale : S0= init(6,6)

— Opération : S1 = setNature(init(6,6),1,0,HDR)

— Oracle :
Playable(S1) = false

Cas de test : EditableScreen : :Playable3
Description : On initialise l'écran à la taille (6,6) et on met la case (5,5) à HOL.

— Condition initiale : S0= init(6,6)

— Opération : S1 = setNature(init(6,6),5,5,HOL)

— Oracle :
Playable(S1) = false

Cas de test : EditableScreen : :test1SetNature
Description : On modifie la case (5,5) à MTL.

— Condition initiale : S0 = init(6,6)

— Opération : S1 = setNature(S0,5,5,MTL)

— Oracle :
CellNature(S1,5,5) = MTL

Tests Environment

Tests préconditions

Cas de test : Environment : :test1InitPrePositif
Description : On initialise l'environnement de hauteur et de largeur 6 avec un EditableScreen non null.

— Condition initiale : ∅

— Opération : S0 = init(6,6,EditableScreen : :init(6,6))

— Oracle :
Pas d'exception levée

Cas de test : Environment : :test1InitPreNegatif
Description : On initialise l'environnement de hauteur et de largeur 6 avec un EditableScreen null.

— Condition initiale : ∅

— Opération : S0 = init(5,2,null)

— Oracle :
Exception levée

Cas de test : Environment : :test1CellContentPrePositif
Description : On récupère le contenu de la case (5,5).

— Condition initiale : S0 = init(6,6,EditableScreen : :init(6,6))

— Opération : S1 = CellContent(S0,5,5)

— Oracle :
Pas d'exception levée

Cas de test : Environment : :test1CellContentPreNegatif
Description : On récupère le contenu de la case (5,6).

— Condition initiale : S0 = init(6,6,EditableScreen : :init(6,6))

— Opération : S1 = CellContent(S0,5,6)

— Oracle :
Exception levée

Tests Transition

Cas de test : Environment : :test1CellContent
Description : On récupère le contenu de la case (5,5).

— Condition initiale : S0 = init(6,6,EditableScreen : :init(6,6))

— Opération : S1 = CellContent(S0,5,5)

— Oracle :
| CellContent(S1,5,5) | = 0

Tests Guards

Tests Préconditions

Cas de test : Guard : :test1InitPrePositif
Description : On initialise un garde normal à la position (0,1) dans un Environment de largeur et de hauteur 5 avec une cible non null.

— Condition initiale : ∅

— Opération : S0 = init(0,1,env,target,false)

— Oracle :
Pas d'exception levée

Cas de test : Guard : :test1InitPreNegatif
Description : On initialise un garde normal à la position (0,1) dans un Environment de largeur et de hauteur 5 qui une case PLT à la position (0,1) avec une cible non null.

— Condition initiale : ∅

— Opération : S0= init(0,1,env,target,false)

— Oracle :
Exception levée

Cas de test : Guard : :test1ClimbLeftPrePositif

Description : On initialise un garde normal à la position (2,2) dans un Environment de largeur et de hauteur 5 où toutes les cases se trouvant à la hauteur 1 sont des plateformes sauf celle se trouvant à la position (1,1) qui correspond à un trou. Le garde se déplace à gauche et tombe dans ce trou et fait ClimbLeft.

— Condition initiale : S0= init(2,2,env,target,false)

— Opération : S1= climbLeft(goDown(goLeft(SO)))

— Oracle :
Pas d'exception levée

Cas de test : Guard : :test1ClimbLeftPreNegatif

Description : On initialise un garde normal à la position (1,1) dans un Environment et une cible non null. Le garde fait ClimbLeft.

— Condition initiale : S0= init(1,1,env,target,false)

— Opération : S1= climbLeft(SO)

— Oracle :
Exception levée

Cas de test : Guard : :test1ClimbRightPrePositif

Description : On initialise un garde normal à la position (0,2) dans un Environment de largeur et de hauteur 5 où toutes les cases se trouvant à la hauteur 1 sont des plateformes sauf celle se trouvant à la position (1,1) qui correspond à un trou. Le garde se déplace à droite et tombe dans ce trou et fait ClimbRight.

— Condition initiale : S0= init(0,2,env,target,false)

— Opération : S1= climbLeft(goDown(goRight(SO)))

— Oracle :
Pas d'exception levée

Cas de test : Guard : :test1ClimbRightPreNegatif

Description : On initialise un garde normal à la position (0,1) dans un Environment et une cible non null. Le garde fait ClimbRight.

— Condition initiale : S0= init(0,1,env,target,false)

— Opération : S1= climbRight(SO)

— Oracle :
Exception levée

Tests Transitions

Cas de test : Guard : :test1ClimbLeft

Description : On initialise un garde normal à la position (2,2) dans un Environment de largeur et de hauteur 5 où toutes les cases se trouvant à la hauteur 1 sont des plateformes sauf celle se trouvant à la position (1,1) qui correspond à un trou. Le garde se déplace à gauche et tombe dans ce trou et fait ClimbLeft.

— Condition initiale : S0= init(2,2,env,target,false)

— Opération : S1= climbLeft(goDown(goLeft(SO)))

— Oracle :
Hgt(S1) = Hgt(S0) Wdt(S1) = Wdt(S0)-2

Cas de test : Guard : :test1ClimbRight

Description : On initialise un garde normal à la position (0,2) dans un Environment de largeur et de hauteur 5 où toutes les cases se trouvant à la hauteur 1 sont des plateformes sauf celle se trouvant à la position (1,1) qui correspond à un trou. Le garde se déplace à gauche et tombe dans ce trou et fait ClimbLeft.

— Condition initiale : S0= init(0,2,env,target,false)

— Opération : S1= climbLeft(goDown(goRight(SO)))

— Oracle :
Hgt(S1) = Hgt(S0)
Wdt(S1) = Wdt(S0)+2

Cas de test : Guard : :test2Step

```
 Lc
 LM
 L
gL
MMMMM
```

Description : L'environnement utilisé dans ce test est décrit ci-dessus. Le garde est représenté par 'g', la cible 'c', les cases métal par 'M' et les échelles par 'L'. Le garde se déplace à droite, monte 3 fois et va à droite pour atteindre la cible.

— Condition initiale : S0= init(0,1,env,target,false)

— Opération : S1= step(step(step(step(step(SO)))))

— Oracle :
Hgt(S1) = Hgt(Target(S1))
Wdt(S1) = Wdt(Target(S1))

Tests Player

Dans les tests de Player, on a utilisé un Map comme ci-dessous :

```

    HHH
  PLPP  P
   L
   L
   L
PPPPPPPPP
MMMMMMMM
```

le map est dans loderunner/map/mapTestPlayer.txt

On note cet Environment "envi", dans la Map, P représente PLT (plateforme), M représente MTL(métal), L représente LDR(échelle) et H représente HDR (handrail)

Tests préconditions

Cas de test : Player : :testInitPositif

— Conditions initiales : envi est un Environment comme décrit dans la Map ci-dessus , e est un Engine

— Opération : P0 = init(envi,4,2,e)

— Oracle :
Engine(P0) = e
Hgt(P0)=h
Wdt(P0)=w
Envi(P0)=s

Cas de test : Player : :testInitNegatif
Description : On initialise le player avec (4,1) ou envi.getCellNature(4,1) == PLT

— Conditions initiales : envi est un Environment comme décrit dans la Map ci-dessus , e est un Engine
— Opération : P0 = init(envi,4,1,e)
— Oracle :
Exception levée

Tests Transition

Cas de test : Player : :testGoRightPLT
Description : le joueur est sur une plateforme et se déplace à droite

— Conditions initiales :P0 = init(envi,4,2,e)
e est un Engine qui ne vaut pas null
envi est un Environment comme décrit dans la Map ci-dessus
— Opération : P1 = GoRight(P0)
— Oracle :
Hgt(P1)= Hgt(P0)
Wdt(P1)= Wdt(P0)+1

Cas de test : Player : :testGoRightHDR
Description : le player est accroché à un rail et et se déplace à droite

— Condition initial :P0 = init(envi,4,2,e)
e est un Engine pas null
envi est un environment comme le map dessus
— Opération : P1 =GoRight(GoRight(GoRight(GoRight(GoUp(GoUp(GoUp(GoUp(GoLeft(P0))))))))))
— Oracle :
Hgt(P1)= Hgt(P0)+4
Wdt(P1)= Wdt(P0)+3

Cas de test : Player : :testGoLeftPLT
Description : le player est sur une plateforme et et se déplace à gauche

— Conditions initiales :P0 = init(envi,4,2,e)
envi est un Environment comme décrit dans la Map ci-dessus , e est un Engine non null
— Opération : P1 = GoLeft(P0)
— Oracle :
Hgt(P1)= Hgt(P0)
Wdt(P1)= Wdt(P0)-1

Cas de test : Player : :testGoUp
Description : le player est sur une plateforme et et se déplace à gauche et puis monte en haut à échelle

— Conditions initiales :P0 = init(envi,4,2,e)
envi est un Environment comme décrit dans la Map ci-dessus , e est un Engine non null
— Opération : P1 = GoUp(GoLeft(P0))
— Oracle :
Hgt(P1)= Hgt(P0)+1
Wdt(P1)= Wdt(P0)-1

Cas de test : Player : :testDigL
Description : Le player est sur une plateforme et il fait DigL
— Condition initial : e est un Engine non null
envi est un Environment comme décrit dans la Map ci-dessus
P0 = init(envi,8,2,e)
— Opération : P1 = DigL(P0)
— Oracle :
Hgt(P1)= Hgt(P0)
Wdt(P1)= Wdt(P0)
Environment : :CellNature(Wdt(P1)-1, Hgt(P1)-1) = HOL

Cas de test : Player : :testPlayerFallDeHaut
Description : Le player tombe de haut
— Condition initial : e est un Engine non null
envi est un Environment comme décrit dans la Map ci-dessus
P0 = init(envi,4,2,e)
— Opération : P1 = Neutral(Left(Left(Up(Up(Up(Up(Left(P0))))))))
— Oracle :
Hgt(P1)= 5
Wdt(P1)= 1

Cas de test : Player : :testPlayerFallDansHole
Description : Le player fait un DigR, et puis il va à droite et tombe dans un trou
— Condition initial : e est un Engine non null
envi est un Environment comme décrit dans la Map ci-dessus
P0 = init(envi,4,2,e)
— Opération : P1 = Neutral(Right(DigR(P0)))
— Oracle :
Hgt(P1)= Hgt(P0)-1
Wdt(P1)= Wdt(P0)+1

Tests Engine

Dans les tests de Engine, on a utilisé un Map comme ci-dessous :

```
1
2
3
4     PPLPPPP
5        L
6        L
7        L
8  PPPPPPPPPP
9  MMMMMMMM
```

le map est dans loderunner/map/mapTestEngine.txt
On note cet Environment "envi", dans la Map, P représente PLT (plateforme), M représente MTL(métal), L représente LDR(échelle) et H représente HDR (handrail)

Tests transitions

Cas de test : Engine : :testTrapperGuard
Description : Le Player est sur une plateforme en position(4,2).Il existe un seul Guard normal en position(0,2). Le Player fait DigL, et ne fait rien pendant 3 steps, le Guard tombe dans le trou creusé.

- Conditions initiales : envi est un Environment comme décrit dans la Map ci-dessus
  P0 = init(envi,4,2)
  G0 = init(envi,0,2,false)
  E0 = init(envi, P, {G}, <6,2>)
- Opération : E1 = Neutral(Neutral(Neutral(DigL(E0))))
- Oracle :
  Environment(E1) : :CellNature(3, 1) = HOL
  Height(G1)= 1
  Width(G1)= 3
  Hgt(P1)= Hgt(P0)
  Wdt(P1)= Wdt(P0)

Cas de test : Engine : :testRamasserTresors
Description : Le Player est sur une plateforme en position(4,2). Il y a trois trésors dans lescases (3,2) (6,2) (7,2), le player va récupérer tous les trésors.

- Conditions initiales : envi est un Environment comme décrit dans la Map ci-dessus
  P0 = init(envi,4,2)
  E0 = init(envi, P, {}, {(3,2)(6,2)(7,2)}
- Opération : E1 = Neutral(GoRight(GoRight(GoRight(GoRight(GoLeft(E0))))
- Oracle :
  Score(P1) = 3
  Status(E1) = Win
  Environment(E1) : :CellContent(3,2) = {}
  Environment(E1) : :CellContent(6,2) = {}
  Environment(E1) : :CellContent(7,2) = {player}

Cas de test : Engine : :testGuardClimbRight
Description : Le Player est sur une plateforme en position (4,2). Il existe un seul Garde normal en position (0,2). Le player fait DigL, puis il va droite et ne fait rien pendant 2 step, le Garde tombe dans le trou. Apres 5 step, le guard fait un ClimbRight.

- Conditions initiales : envi est un Environment comme décrit dans la Map ci-dessus
  P0 = init(envi,4,2)
  G0 = init(envi,0,2,false)
  E0 = init(envi, P, {G}, <6,2>)
- Opération : E1 =Neutral(Neutral(Neutral(Neutral(Neutral(Neutral(Neutral(Right(DigL(E0)))))))))
- Oracle :
  Environment(E1) : :CellNature(3, 1) = HOL
  Height(G1)= 2

Width(G1)= 4

Cas de test : Engine : :testExtensionContrat
Description : Le Player est sur une plateforme en position (4,2). Il existe un seul Garde normal en position (0,2). Le player fait DigL, puis il attend 3 step pour que le guard tombe dans le trou, et puis, il va à gauche et dans la cas au-dessus de guard,il attend encore 4 step pour que le TimeInHole de guard devient 5, donc le guard touche le player.

- Conditions initiales : envi est un Environment comme décrit dans la Map ci-dessus
  P0 = init(envi,4,2)
  G0 = init(envi,0,2,false)
  E0 = init(envi, P, {G}, <6,2>)
- Opération : E1 = Neutral(Neutral(Neutral(Neutral(GoLeft(Neutral(Neutral(Neutral(DigL(E0))))))))
- Oracle :
  Environment(E1) : :CellNature(3, 1) = HOL
  TimeInHole(G1) = 5
  Vie(P1) = 2
  //Le player et le guard sont revenus à leurs positions initiales :
  Height(P1)= Height(P0)
  Width(P1) = Width(P0)
  Height(G1)= Height(G0)
  Width(G1) = Width(G0)

Cas de test : Engine : :testExtensionGuardSpecial
Description : Le Player est sur une plateforme en position (4,2). Il existe un seul Garde spécial en position (1,2). Le player fait DigL, puis il attend 2 step, vue que le guard special peut passer au dessus des trous, il touche le player sans tomber dans le trou.

- Conditions initiales : envi est un Environment comme décrit dans la Map ci-dessus
  P0 = init(envi,4,2)
  G0 = init(envi,1,2,true)
  E0 = init(envi, P, {G}, <6,2>)
- Opération : E1 = Neutral(Neutral(DigL(E0)))
- Oracle :
  Environment(E1) : :CellNature(3, 1) = HOL
  Vie(P1) = 2
  Height(P1)= Height(P0) //le player et le guard sont revenu à sa position initiale
  Width(P1) = Width(P0)
  Height(G1)= Height(G0)
  Width(G1) = Width(G0)

Cas de test : Engine : :testExtensionScoreVie
Description : Le Player est sur une plateforme en position (4,2). Il existe un seul Garde normal en position (0,2). et il y a trois trésors dans cases (3,2) (6,2) (7,2), le player va à gauche et récupérer le premier trésor, le score devient 1, il attend 2 step pour que le guard le touche, player revients à sa position initiale, le score reinitialise à 0, il va à gauche jusqu'à récupérer tous les trésors, le jeu est gagné, son score est 2.

- Conditions initiales : envi est un Environment comme décrit dans la Map ci-dessus
  P0 = init(envi,4,2)
  G0 = init(envi,0,2,false)
  E0 = init(envi, P, {G}, (3,2),(6,2),(7,2)

— Opération : E1 = Neutral(GoRight(GoRight(GoRight(Neutral(Neutral(GoLeft(E0)))
— Oracle :
Vie(P1) = 2
Score(P1) = 2
Status(E1) = Win

Tests scénarios

Test scénario : Le garde revient à sa position initiale

Cas de test : Engine : :testGuardRevientPosInit

Description : Au début, le Player est en position (4,2), un seul Guard est en position(0, 2), le joueur va à gauche et fait DigL, puis il va à droite et fait un DigL, puis il va à droite et fait un DigL (il a fait 3 trous au total), apres il attend pendant 15 step. Le Guard tombe dans le premier trou et puis apres 5 step, le guard fait un ClimbRight, et puis il tombe dans le deuxième trou. Après 5 step, le Guard refait un ClimbRight et est amené dans la 3ème trou. Le 3ème trou se rebouche, et le Guard revient à sa position initiale.

— Conditions initiales : envi est un Environment comme décrit dans la Map ci-dessus
P = init(envi,3,2)
G = init(envi,0,2)
E0 = init(envi, P, {G}, <4,2>)
— Opération : P1 =Neutral(Neutral(Neutral(Neutral(Neutral(Neutral(Neutral(Neutral(Neutral(Neutral(Neutral(
Neutral(Neutral(Neutral(Neutral(DigL(Right(DigL(Right(DigL(Left(E0))))))))))))))))))))
— Oracle :
Width(G1) = 0
Height(G1) = 2

Tests Etats Remarquables

Etat remarquable : Le jeu est gagné

Cas de test : Engine : :testWin
Description :Au début, le Player est en position (3,2), un seul trésor est en position(4, 2), le Player va aller a droite et récuperer le trésor, le jeu est gagné.
— Conditions initiales : envi est un Environment comme décrit dans la Map ci-dessus
P = init(envi,3,2)
G = init(envi,0,2)
E0 = init(envi, P, {G}, <4,2>)
— Opération : E1 = GoRight(E0)
— Oracle :
Status(E1) = Win
Environment(E1).getCellContent(4, 2) = {P1}
Player(E1).getScore = 1

Etat remarquable : Le jeu est perdu car le joueur est attaqué par un guard 3 fois
Cas de test : Engine : :testLossByGuard
Description :Au début, le Player est en position (2,2) et possède trois vies, un seul guard est en position(0, 2), le Player ne bouge pas, le Guard va rattraper le joueur, et ca recommence encore 2 fois, le joueur est attrappé par le guard 3 fois le jeu est perdu.

— Conditions initiales : envi est un Environment comme décrit dans la Map ci-dessus
P0 = init(envi,2,2)
G0 = init(envi,0,2,false)
E0 = init(envi, P, {G}, <6,2>)

— Opération : E1 = Neutral(Neutral(Neutral(Neutral(Neutral(Neutral(E0))))))

— Oracle :
getStatus(E1) = Loss
Player(E1).getVie = 0

Etat remarquable : Le player perd une vie car le joueur se trouve dans un trou qui se rebouche.
Cas de test : Engine : :testPlayerPerdVieByTrouRebouche
Description :Le Player tombe dans un trou pendant 15 step et possède la totalité de ses points de vie. Le trou se rebouche.Le Player perd une vie et revient a sa position initiale.

— Condition initial : envi est un Environment comme décrit dans la Map ci-dessus
P0 = init(envi,3,2)
E0 = init(envi, P, {}, <4,2>)

— Opération : E1 = Neutral(Neutral(Neutral(Neutral(Neutral(Neutral(Neutral(Neutral(Neutral(Neutral(Neutral(Neutral(Neutral(Left(DigL(E0))))))))))))))))

— Oracle :
Environnement(E1) : :getCellNature(Wdt(P1),Hgt(P1)) = PLT Vie(P1) = Vie(P0)-1
Score(P1) = 0

Rapport de projet

Les modifications et extensions

Vie et Score

Le jeu doit maintenir un nombre de ”vies” pour le joueur, tout échec dans écran décrémente ce compteur et réinitialise l'écran. Quand ce nombre vaut 0, la partie est perdue. Chaque trésor ramassé augmente le "score" du joueur (le score gagné dans un niveau réinitialisé suite à un échec est perdu). Pour ajouter cette modification, on ajoute dans Player deux observateurs :
`Vie` : [Player] → int
`Score` : [Player] → int
et aussi deux operateurs pour modifier le score et la vie du joueur :
`setScore` : [Player] × int → [Player]
`decreVie` : [Player] →[Player]

Initialement, le player possède 3 vies. Lorsque le joueur perd une vie, on initialise les positions du Player et des Guards, on décrémente les vies du player et on remet le score du Player à 0.
Cela nous oblige à ajouter une fonction pour réinitialiser toutes les positions des personnages dans EngineImpl :

```
public void reinitialisePos() {
    player.setPos(xPlayerInit, yPlayerInit);
    for(GuardService guard : guards) {
      int index = guards.indexOf(guard);
      int guardX_init = listGuardsInitiaux.get(index).getL();
      int guardY_init = listGuardsInitiaux.get(index).getR();
      guard.setPos(guardX_init, guardY_init);
    }
    player.setScore(0);
}
```

On a stocké la position initiale du Player dans (xPlayerInit , yPlayerInit), et les positions des Guards dans listGuardsInitiaux.

Le joueur perd une vie quand il est touché par un guard ou lorsque le joueur se trouve dans un trou qui se rebouche :

```
 ......
    //quand un trou est rebouche, ou quand un garde est dans la meme case que le joueur
      player.decreVie();
      reinitialisePos() ;
      if(player.getVie()==0) {
        status = Status.Loss;
      }
......
```

On doit aussi faire des modifications dans EngineContrat (chaque fois le player est touché par un guard, il doit perdre une vie) :

```
 public void step() {
    for(GuardService g : getGuards()) {
      if(g.getHgt() == getPlayer().getHgt() && g.getWdt() == getPlayer().getWdt()) {
        if(vie_pre>0) {
          if((getPlayer().getVie() != vie_pre-1)) {
            throw new PostconditionError("player should lose a life");
          }
        }
......
```

Contact entre Guard et Player

Pour la gestion du contact entre un garde et le joueur : quand un garde s'extirpe d'un trou au-dessus duquel se trouve le joueur, il attrape le joueur, le joueur perd une vie et réinitialise l'écran.
Pour ajouter cette modification, on a modifié GuardImpl :

```
public void step(){
......
 if(getEnvi().getCellNature(x, y) == Cell.HOL && timeInHole == 5 ) {
    ......
   if(target.getHgt() == y + 1 && target.getWdt() == x){
    System.out.println("guard go up to catch player");
        goUp();
  }
}
......
}
```

Dans la fonction step(), si le garde se trouve dans un trou, et que TimeInHole vaut 5, on vérifie si il y a un player au-dessus de lui en ce moment, si oui, le guard va faire goUp pour attraper le joueur.

Et ca doit aussi aménер des modifications dans GuardContrat : (On avait défini des prédicats willGoUp et les réutiliser dans les gardes permet de rendre plus lisible)

```
 boolean willGoUp = ( // gestion contrat
      ( getBehavior() == Move.Up || ( getEnvi().getCellNature(getWdt(),getHgt()) == Cell.HOL
             && getTarget().getHgt() == getHgt() +1
             && getTarget().getWdt() == getWdt()
             && getTimeInHole() == 5)
      )
      && getHgt() < getEnvi().getHeight()
      && (getEnvi().getCellNature(getWdt(),getHgt()+1) == Cell.LAD || getEnvi().
    getCellNature(getWdt(),getHgt()+1) == Cell.EMP
        || getEnvi().getCellNature(getWdt(),getHgt()+1) == Cell.HDR || getEnvi().
    getCellNature(getWdt(),getHgt()+1) == Cell.HOL )
      && ! haveGuardEnHaut ) ;
```

Personnages spéciaux

On a créé un personnage spécial comme des gardes qui peuvent passer au dessus des trous. Pour ajouter cette extension dans l'implementation du jeu, on doit ajouter dans Guard un nouveau observateur pour définir si ce guard est un guard spécial :
`IsSpecial` : [Guard] → boolean
Aussi, on a modifié l'opérator init de Guard en ajoutant un nouveau paramètre boolean à la fin :
`init` : Screen × int × int × Player × boolean → [Guard]
Si c'est un guard normal, on donne false en initialisation, si c'est un guard spéciale, on donne true en paramètre d'initialisation.
Et ca nous oblige aussi de modifier init de Engine :
Engine : : `init` : EditableScreen × int × int × List {int × int × boolean } × List{int × int } → [Engine]

On a ajouté des modifications dans GuardImpl pour que le guard ne tombe pas quand il est au dessous d'un trou :

```
......
    if(...conditions guard tombe ...
        && !(getEnvi().getCellNature(x, y-1) == Cell.HOL
            && isSpecial())){
    ...le garde tombe...
    }
```

On a aussi ajouté des modifications pour que le guard puisse marcher à gauche ou à droite quand il est au dessous d'un trou :

```
public void step(){
......
   if(isSpecial() && getEnvi().getCellNature(x, y-1) == Cell.HOL) {
     if(getBehavior() == Move.Right
         && env.getCellNature(getWdt()+1,getHgt()) !=  Cell.MTL
         && env.getCellNature(getWdt()+1,getHgt()) !=  Cell.PLT
         && wdt!= env.getWidth()-1
         && !hasGuardRight) {
       env.getCellContent(wdt, hgt).remove(this);
       wdt = wdt+1;
       env.getCellContent(wdt, hgt).add(this);
     }
     if(getBehavior() == Move.Left
         && env.getCellNature(getWdt()-1,getHgt()) !=  Cell.MTL
         && env.getCellNature(getWdt()-1,getHgt()) !=  Cell.PLT
         && wdt!= 0
         && !hasGuardLeft) {
       env.getCellContent(wdt, hgt).remove(this);
       wdt = wdt-1;
       env.getCellContent(wdt, hgt).add(this);
     }
   }
......
```

Et on ajoute une condition pour WillFall dans GuardContrat :

```
boolean willFall = ( ....
...les condition de WillFall...
&& !(isSpecial() && getEnvi().getCellNature(getWdt(), getHgt()-1) == Cell.HOL)
);
```