Rapport de CPS
Conception par contrat d'un Dungeon Master

Amel ARKOUB 3301571
Ling-Chun SO 3414546

Descriptif du projet **3**

Manuel d'utilisation **4**

Executer le projet 4

Comment jouer 4

Spécifications **8**

Types basiques 8

Services 8

Service: Map 8

Service: EditMap refines Map 9

Service: Environment includes EditMap 10

Service: Mob 10

Service: Entity includes Mob 14

Service: Cow includes Entity 15

Service: Player includes Entity 16

Service: Engine 17

Rapport de projet **19**

Choix d'implantation et difficultés de spécification 19

Test pertinents 21

Descriptif du projet

Le but du projet est d'implanter un jeu similaire à Dungeon Master dont le cahier des charges a été partiellement décrit. Il s'agit d'implanter cette spécification selon la méthode Design-by-Contract et d'écrire une implantation du jeu.

L'archive fournit se compose de:

- Ce rapport
- Des sources du projet

Les sources du projets sont découpés dans des packages qui contiennent donc:

Dungeon.master.components → L'implantation des composants qui spécifie les contrats
Dungeon.master.components.bug → L'implantation des composants qui ne vérifie pas les contrats
Dungeon.master.contracts → L'implantation des contrats
Dungeon.master.decorators → L'implantation des decorateurs
Dungeon.master.enumerations → Les types de bases qui sont des énumérations Java
Dungeon.master.exceptions → Les exceptions des préconditions, postconditions et invariants
Dungeon.master.mbt.test → L'implantation des tests MBT
Dungeon.master.services → Les interfaces services
Dungeon.master.ui → L'implantation de l'interface graphique
Dungeon.master.ui.implementations → Des décorateurs permettant de faire le lien entre l'interface et l'implantation

Manuel d'utilisation

Executer le projet

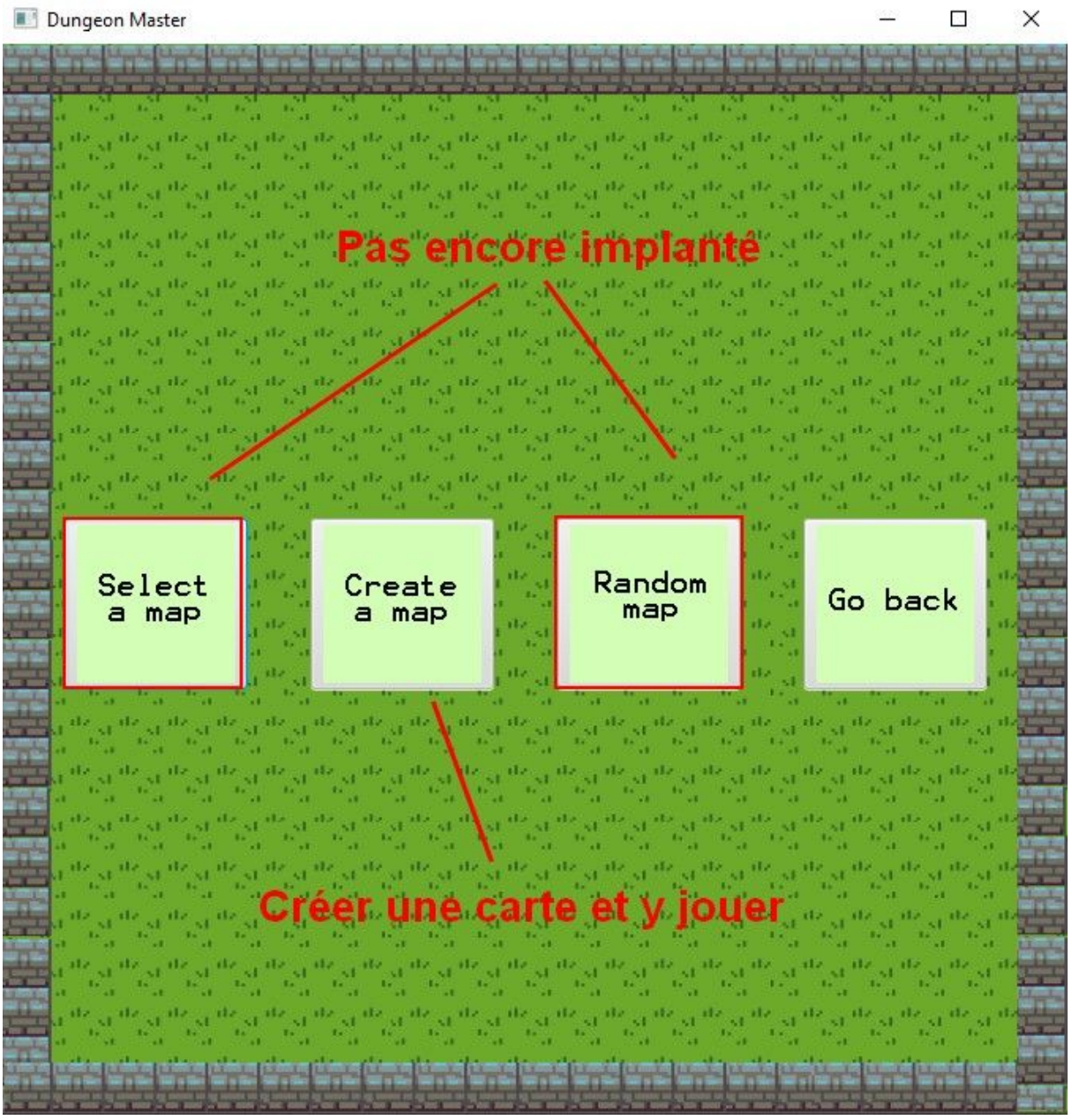
Afin de faciliter l'utilisation du projet, nous avons utilisé ant dont les différentes commandes sont les suivantes:

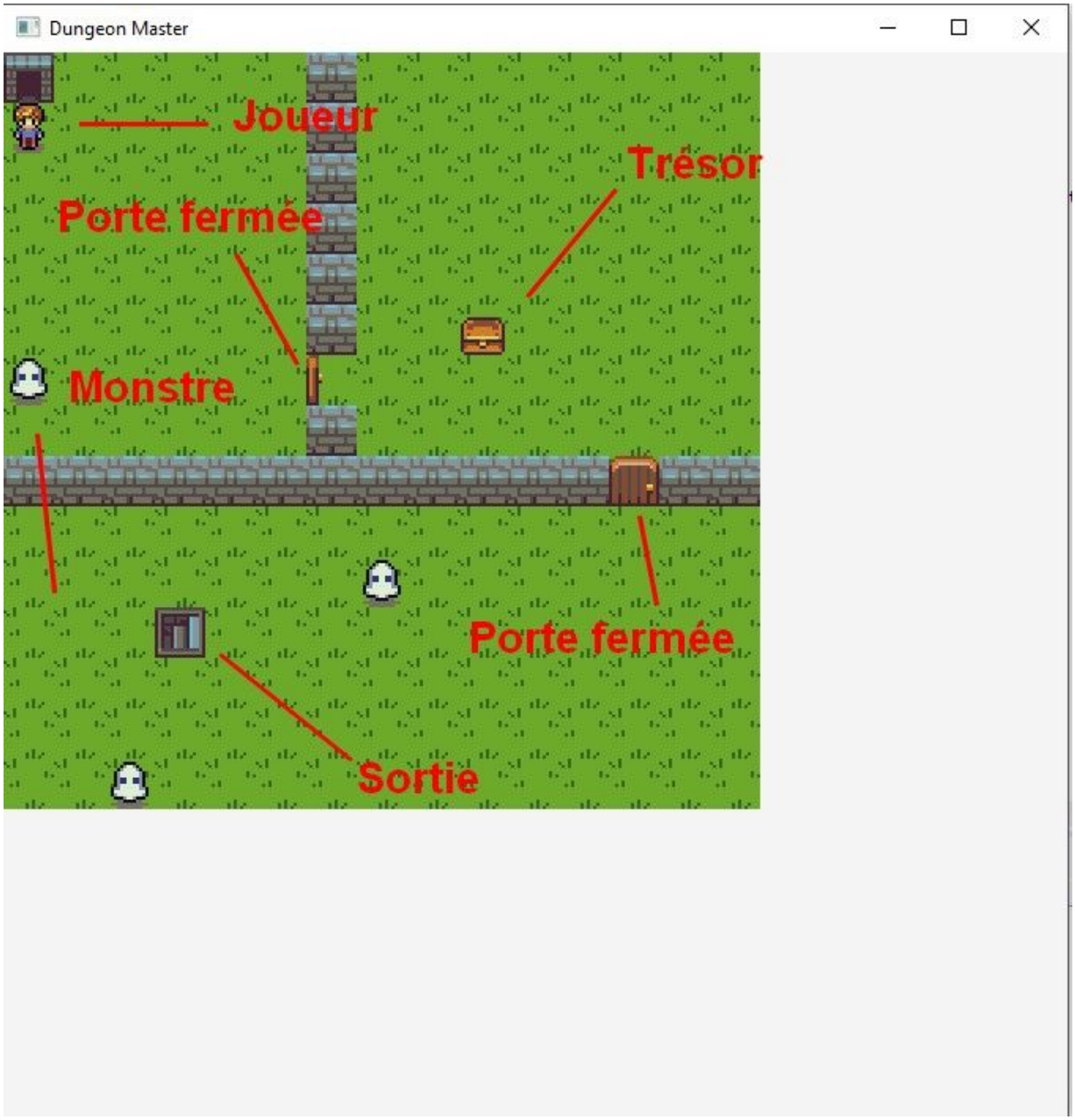
- Ant run → Lance le jeu dungeon master
- Ant dist → Génère une distribution
- Ant test → Lance tout les tests junit (Le fichier lancé est dungeon.master.mbt.test.RunAllTests)

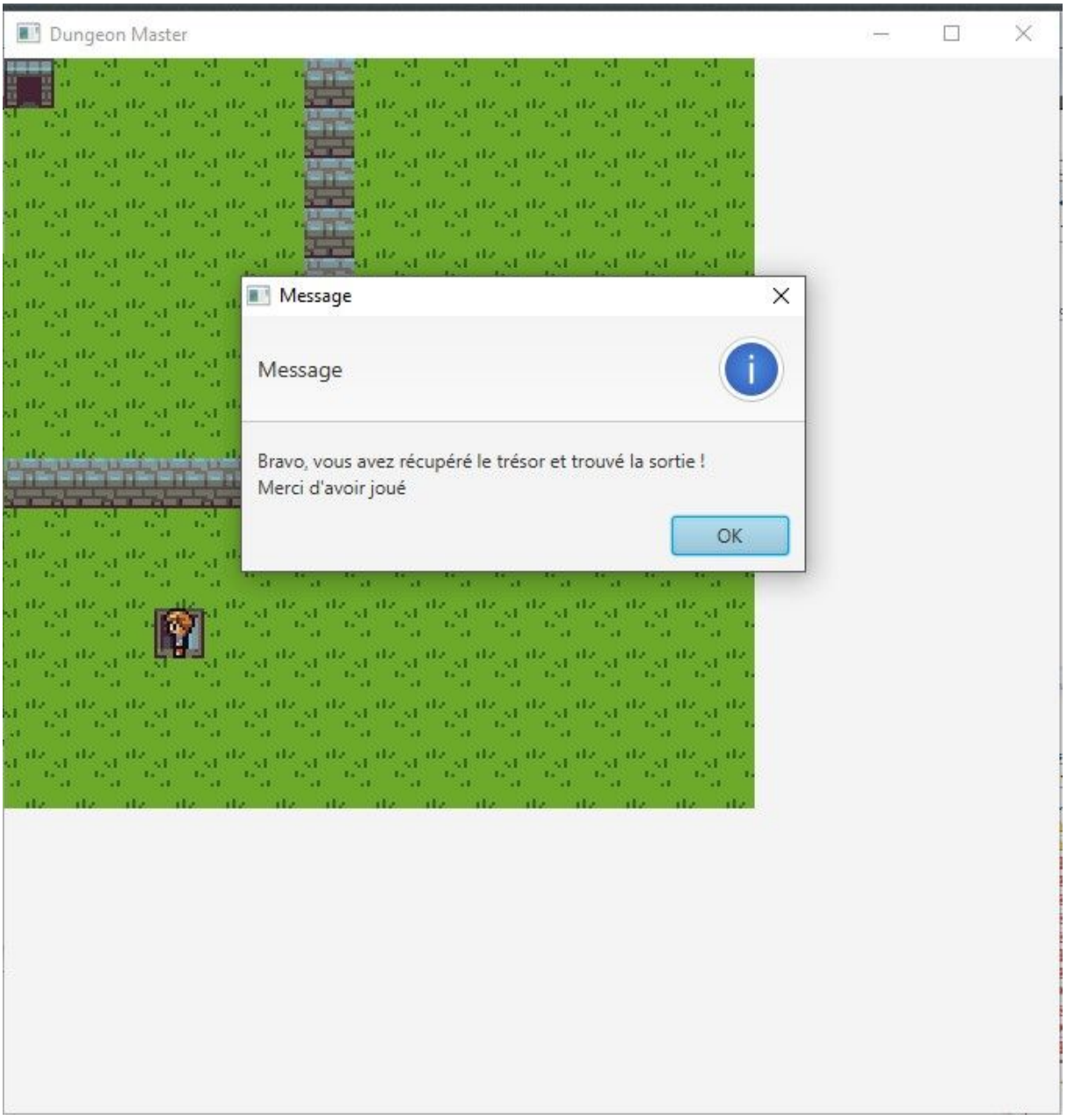
Comment jouer

Pour se déplacer, utiliser les flèches du pad. (Il est à noter, qu'à chaque changement de direction les flèches du pad sont rebinds afin d'obtenir un mouvement intuitif)
Pour tourner à gauche, appuyer sur S.
Pour tourner à droite, appuyer sur D.
Pour ouvrir une porte, appuyer sur Z.
Pour fermer une porte, appuyer sur E.
Pour attaquer un ennemi, appuyer sur ESPACE. (attaque en face du joueur)
Pour prendre le trésor, appuyer sur T. (prend le trésor en face du joueur)

Voici des images avec des légendes afin de mieux comprendre les fonctionnalités du jeu:


Dungeon Master
Pas encore implanté
Select a map
Create a map
Random map
Go back
Créer une carte et y jouer


Dungeon Master
Joueur
Trésor
Porte fermée
Monstre
Porte fermée
Sortie


Dungeon Master
Message
Message
Bravo, vous avez récupéré le trésor et trouvé la sortie !
Merci d'avoir joué
OK

Spécifications

Types basiques

Type Cell {IN, OUT, EMP, WLL, DNO, DNC, DWO, DWC, TRS}
Type Dir {N, S, W, E}
Type Command {FF, BB, RR, LL, TL, TR, NONE}
Type Option[T] {No, So(T)}
Type Set[T]
Type Array[T1,...,TN]

Services

Service: Map

Types: bool, int, Cell
Observators:
const Height: [Map] → int
const Width: [Map] → int
CellNature: [Map] x int x int → Cell
Pre CellNature(M,x,y) requires 0 ≤ x w Width(M) and 0 ≤ y < Height(M)
Constructors:
init: int x int → [Map]
Pre init(w,h) requires 0 < w and 0 < h
Operators:
OpenDoor: [Map] x int x int → [Map]
Pre OpenDoor(M,x,y) requires CellNature(M,x,y) ∈ {DNC, DWC}
CloseDoor: [Map] x int x int → [Map]
Pre CloseDoor(M,x,y) requires CellNature(M,x,y) ∈ {DNO, DWO}
Observation:
[Invariant]: T
[init]:
Height(init(h,w)) = h
Width(init(h,w)) = w
[OpenDoor]:
CellNature(M,x,y) = DWC ⇒ CellNature(OpenDoor(M,x,y),x,y) = DWO
CellNature(M,x,y) = DNC ⇒ CellNature(OpenDoor(M,x,y),x,y) = DNO

Forall u ∈ [0;Width(M)-1] forall v ∈ [0;Height(M)-1] (u ≠ x or v ≠ y) ⇒ CellNature(OpenDoor(M,x,y),u,v) = CellNature(M,u,x)

[CloseDoor]:

CellNature(M,x,y) = DWO ⇒ CellNature(OpenDoor(M,x,y),x,y) = DWC

CellNature(M,x,y) = DNO ⇒ CellNature(OpenDoor(M,x,y),x,y) = DNC

Forall u ∈ [0;Width(M)-1] forall v ∈ [0;Height(M)-1] (u ≠ x or v ≠ y) ⇒ CellNature(OpenDoor(M,x,y),u,v) = CellNature(M,u,x)

Service: EditMap refines Map

Types: bool, int, Cell

Observators:

isReachable: [EditMap] x int x int x int x int → [EditMap]

Pre isReachable(M,x1,y1,x2,y2) requires CellNature(M,x1,y1) ≠ WLL and CellNature(M,x2,y2) ≠ WLL

isReady: [EditMap] → bool

Constructors: NONE

Operators:

SetNature: [EditMap] x int x int x Cell → [EditMap]

Pre SetNature(M,x,y) requires 0 ≤ x < Width(M) and 0 ≤ y < Height(M)

Observation:

[Invariant]:

IsReachable(M,x1,y1,x2,y2) = exists P in Array[int,int], P[0] = (x1,y1) and P[size(P)-1] = (x2,y2) and forall i int [1;size(P)-1], (P[i-1]=(u,v) and P[i]=(s,t)) ⇒ (u-s)²+(v-t)² = 1 and forall i in [1;size(P)-2], P[i-1]=(u,v) ⇒ CellNature(M,u,v) ≠ WLL

isReady(M) = exists xi,yi) = IN and CellNature(M,xo,yo) = OUT
and isReachable(M,xi,yi,xo,yi)
and forall x,y in int², x≠ xi or y≠ yi ⇒ CellNature(M,x,y) ≠ IN
and forall x,y in int², x≠ xo or y≠ yo ⇒ CellNature(M,x,y) ≠ OUT
forall x,y in int, CellNature(M,x,y) ∈ {DNO,DNC} ⇒
CellNature(M,x+1,y) = CellNature(M,x-1,y) = EMP and
CellNature(M,x,y-1) = CellNature(M,x,y+1) = WLL
forall x,y in int, CellNature(M,x,y) ∈ {DWO,DWC} ⇒
CellNature(M,x+1,y) = CellNature(M,x-1,y) = EMP and
CellNature(M,x,y-1) = CellNature(M,x,y+1) = WLL

[SetNature]:

CellNature(SetNature(M,x,y,Na),x,y) = Na
forall u,v in int², u≠ x or v≠ y ⇒ CellNature(SetNature(M,x,y),u,v) = CellNature(M,u,v)

Service: Environment includes EditMap

Types: bool, int, Cell, Entity
Observators: CellContent: int x int → Option[Entity]
Operators: CloseDoor: [Environment] x int x int → [Environment]
Pre CloseDoor(M,x,y) requires CellContent(M,x,y) = No

Service: Mob

Types: bool, int, Cell
Observators:
Env: [Mob] → Environment
Col: [Mob] → int
Row: [Mob] → int
Face: [Mob] → Dir
Constructors:
Init: Environment x int x int x Dir → [Mob]
Pre init(E,x,y,D) requires 0 ≤ x < Environment::Width(E) and 0 ≤ y < Environment::Height(E)
Operators:
Forward: [Mob] → [Mob]
Backward: [Mob] → [Mob]
TurnL: [Mob] → [Mob]
TurnR: [Mob] → [Mob]
StrafeL: [Mob] → [Mob]
StrafeR: [Mob] → [Mob]

[Invariant]:
0 ≤ Col(M) < Environment::Width(Envi(M))
0 ≤ Row(M) < Environment::Height(Envi(M))
Environment::CellNature(Envi(M),Col(M),Row(M)) ∈ {WLL,FNC,DWC}
[init]:
Col(init(E,x,y,D)) = x
Row((init(E,x,y,D)) = y
Face((init(E,x,y,D)) = D
Envi(init(E,x,y,D)) = E
Observation:
[Forward]:
Aucun changement dans Face(M)
Face(M) = N ⇒
Environment::CellNature(Envi(M),Col(M),Row(M)+1) ∈ {EMP,DNO}

and Row(M)+1 < Environment::Height(Envi(M))
and Environment::CellContent(Envi(M),Col(M),Row(M)+1) = No
⇒ Row(Forward(M)) = Row(M)+1 and Col(Forward(M)) = Col(M)

Face(M) = N ⇒
Environment::CellNature(Envi(M),Col(M),Row(M)+1) ∉ {EMP,DNO}
or Row(M)+1 ≥ Environment::Height(Envi(M))
or Environment::CellContent(Envi(M),Col(M),Row(M)+1) ≠ No
⇒ Row(Forward(M)) = Row(M) and Col(Forward(M)) = Col(M)

Face(M) = S ⇒
Environment::CellNature(Envi(M),Col(M),Row(M)-1) ∈ {EMP,DNO}
and Row(M)-1 ≥ 0
and Environment::CellContent(Envi(M),Col(M),Row(M)-1) = No
⇒ Row(Forward(M)) = Row(M)-1 and Col(Forward(M)) = Col(M)

Face(M) = S ⇒
Environment::CellNature(Envi(M),Col(M),Row(M)-1) ∉ {EMP,DNO}
or Row(M)-1 < 0
or Environment::CellContent(Envi(M),Col(M),Row(M)+1) ≠ No
⇒ Row(Forward(M)) = Row(M) and Col(Forward(M)) = Col(M)

Face(M) = E ⇒
Environment::CellNature(Envi(M),Col(M)+1,Row(M)) ∈ {EMP,DWO}
and Col(M)+1 < Environment::Width(Envi(M))
and Environment::CellContent(Envi(M),Col(M)+1,Row(M)) = No
⇒ Row(Forward(M)) = Row(M) and Col(Forward(M)) = Col(M)+1

Face(M) = E ⇒
Environment::CellNature(Envi(M),Col(M)+1,Row(M)) ∉ {EMP,DWO}
or Col(M)+1 ≥ Environment::Width(Envi(M))
or Environment::CellContent(Envi(M),Col(M)+1,Row(M)) ≠ No
⇒ Row(Forward(M)) = Row(M) and Col(Forward(M)) = Col(M)

Face(M) = W ⇒
Environment::CellNature(Envi(M),Col(M)-1,Row(M)) ∈ {EMP,DWO}
and Col(M)-1 ≥ 0
and Environment::CellContent(Envi(M),Col(M)-1,Row(M)) = No
⇒ Row(Forward(M)) = Row(M) and Col(Forward(M)) = Col(M)-1

Face(M) = W ⇒
Environment::CellNature(Envi(M),Col(M)-1,Row(M)) ∉ {EMP,DWO}
or Col(M)-1 < 0
or Environment::CellContent(Envi(M),Col(M)-1,Row(M)) ≠ No
⇒ Row(Forward(M)) = Row(M) and Col(Forward(M)) = Col(M)

[Backward]:

Aucun changement dans Face(M)

Face(M) = S ⇒
Environment::CellNature(Envi(M),Col(M),Row(M)+1) ∈ {EMP,DNO}
and Row(M)+1 < Environment::Height(Envi(M))
and Environment::CellContent(Envi(M),Col(M),Row(M)+1) = No
⇒ Row(Backward(M)) = Row(M)+1 and Col(Backward(M)) = Col(M)

Face(M) = S ⇒

Environment::CellNature(Envi(M),Col(M),Row(M)+1) ∉ {EMP,DNO}
or Row(M)+1 ≥ Environment::Height(Envi(M))
or Environment::CellContent(Envi(M),Col(M),Row(M)+1) ≠ No
⇒ Row(Backward(M)) = Row(M) and Col(Backward(M)) = Col(M)

Face(M) = N ⇒
Environment::CellNature(Envi(M),Col(M),Row(M)-1) ∈ {EMP,DNO}
and Row(M)-1 ≥ 0
and Environment::CellContent(Envi(M),Col(M),Row(M)-1) = No
⇒ Row(Backward(M)) = Row(M)-1 and Col(Backward(M)) = Col(M)

Face(M) = N ⇒
Environment::CellNature(Envi(M),Col(M),Row(M)-1) ∉ {EMP,DNO}
or Row(M)-1 < 0
or Environment::CellContent(Envi(M),Col(M),Row(M)+1) ≠ No
⇒ Row(Backward(M)) = Row(M) and Col(Backward(M)) = Col(M)

Face(M) = W ⇒
Environment::CellNature(Envi(M),Col(M)+1,Row(M)) ∈ {EMP,DWO}
and Col(M)+1 < Environment::Width(Envi(M))
and Environment::CellContent(Envi(M),Col(M)+1,Row(M)) = No
⇒ Row(Backward(M)) = Row(M) and Col(Backward(M)) = Col(M)+1

Face(M) = W ⇒
Environment::CellNature(Envi(M),Col(M)+1,Row(M)) ∉ {EMP,DWO}
or Col(M)+1 ≥ Environment::Width(Envi(M))
or Environment::CellContent(Envi(M),Col(M)+1,Row(M)) ≠ No
⇒ Row(Backward(M)) = Row(M) and Col(Backward(M)) = Col(M)

Face(M) = E ⇒
Environment::CellNature(Envi(M),Col(M)-1,Row(M)) ∈ {EMP,DWO}
and Col(M)-1 ≥ 0
and Environment::CellContent(Envi(M),Col(M)-1,Row(M)) = No
⇒ Row(Backward(M)) = Row(M) and Col(Backward(M)) = Col(M)-1

Face(M) = E ⇒
Environment::CellNature(Envi(M),Col(M)-1,Row(M)) ∉ {EMP,DWO}
or Col(M)-1 < 0
or Environment::CellContent(Envi(M),Col(M)-1,Row(M)) ≠ No
⇒ Row(Backward(M)) = Row(M) and Col(Backward(M)) = Col(M)

[StrafeL]:

Aucun changement dans Face(M)

Face(M) = E ⇒
Environment::CellNature(Envi(M),Col(M),Row(M)+1) ∈ {EMP,DNO}
and Row(M)+1 < Environment::Height(Envi(M))
and Environment::CellContent(Envi(M),Col(M),Row(M)+1) = No
⇒ Row(StrafeL(M)) = Row(M)+1 and Col(StrafeL(M)) = Col(M)

Face(M) = E ⇒
Environment::CellNature(Envi(M),Col(M),Row(M)+1) ∉ {EMP,DNO}
or Row(M)+1 ≥ Environment::Height(Envi(M))
or Environment::CellContent(Envi(M),Col(M),Row(M)+1) ≠ No
⇒ Row(StrafeL(M)) = Row(M) and Col(StrafeL(M)) = Col(M)

Face(M) = W ⇒
Environment::CellNature(Envi(M),Col(M),Row(M)-1) ∈ {EMP,DNO}
and Row(M)-1 ≥ 0
and Environment::CellContent(Envi(M),Col(M),Row(M)-1) = No
⇒ Row(StrafeL(M)) = Row(M)-1 and Col(StrafeL(M)) = Col(M)
Face(M) = W ⇒
Environment::CellNature(Envi(M),Col(M),Row(M)-1) ∉ {EMP,DNO}
or Row(M)-1 < 0
or Environment::CellContent(Envi(M),Col(M),Row(M)+1) ≠ No
⇒ Row(StrafeL(M)) = Row(M) and Col(StrafeL(M)) = Col(M)
Face(M) = S ⇒
Environment::CellNature(Envi(M),Col(M)+1,Row(M)) ∈ {EMP,DWO}
and Col(M)+1 < Environment::Width(Envi(M))
and Environment::CellContent(Envi(M),Col(M)+1,Row(M)) = No
⇒ Row(StrafeL(M)) = Row(M) and Col(StrafeL(M)) = Col(M)+1
Face(M) = S ⇒
Environment::CellNature(Envi(M),Col(M)+1,Row(M)) ∉ {EMP,DWO}
or Col(M)+1 ≥ Environment::Width(Envi(M))
or Environment::CellContent(Envi(M),Col(M)+1,Row(M)) ≠ No
⇒ Row(StrafeL(M)) = Row(M) and Col(StrafeL(M)) = Col(M)
Face(M) = N ⇒
Environment::CellNature(Envi(M),Col(M)-1,Row(M)) ∈ {EMP,DWO}
and Col(M)-1 ≥ 0
and Environment::CellContent(Envi(M),Col(M)-1,Row(M)) = No
⇒ Row(StrafeL(M)) = Row(M) and Col(StrafeL(M)) = Col(M)-1
Face(M) = N ⇒
Environment::CellNature(Envi(M),Col(M)-1,Row(M)) ∉ {EMP,DWO}
or Col(M)-1 < 0
or Environment::CellContent(Envi(M),Col(M)-1,Row(M)) ≠ No
⇒ Row(StrafeL(M)) = Row(M) and Col(StrafeL(M)) = Col(M)

[StrafeR]:
Aucun changement dans Face(M)
Face(M) = W ⇒
Environment::CellNature(Envi(M),Col(M),Row(M)+1) ∈ {EMP,DNO}
and Row(M)+1 < Environment::Height(Envi(M))
and Environment::CellContent(Envi(M),Col(M),Row(M)+1) = No
⇒ Row(StrafeR(M)) = Row(M)+1 and Col(StrafeR(M)) = Col(M)
Face(M) = W ⇒
Environment::CellNature(Envi(M),Col(M),Row(M)+1) ∉ {EMP,DNO}
or Row(M)+1 ≥ Environment::Height(Envi(M))
or Environment::CellContent(Envi(M),Col(M),Row(M)+1) ≠ No
⇒ Row(StrafeR(M)) = Row(M) and Col(StrafeR(M)) = Col(M)
Face(M) = E ⇒
Environment::CellNature(Envi(M),Col(M),Row(M)-1) ∈ {EMP,DNO}
and Row(M)-1 ≥ 0
and Environment::CellContent(Envi(M),Col(M),Row(M)-1) = No

⇒ Row(StrafeR(M)) = Row(M)-1 and Col(StrafeR(M)) = Col(M)

Face(M) = E ⇒
Environment::CellNature(Envi(M),Col(M),Row(M)-1) ∉ {EMP,DNO}
or Row(M)-1 < 0
or Environment::CellContent(Envi(M),Col(M),Row(M)+1) ≠ No
⇒ Row(StrafeR(M)) = Row(M) and Col(StrafeR(M)) = Col(M)

Face(M) = N ⇒
Environment::CellNature(Envi(M),Col(M)+1,Row(M)) ∈ {EMP,DWO}
and Col(M)+1 < Environment::Width(Envi(M))
and Environment::CellContent(Envi(M),Col(M)+1,Row(M)) = No
⇒ Row(StrafeR(M)) = Row(M) and Col(StrafeR(M)) = Col(M)+1

Face(M) = N ⇒
Environment::CellNature(Envi(M),Col(M)+1,Row(M)) ∉ {EMP,DWO}
or Col(M)+1 ≥ Environment::Width(Envi(M))
or Environment::CellContent(Envi(M),Col(M)+1,Row(M)) ≠ No
⇒ Row(StrafeR(M)) = Row(M) and Col(StrafeR(M)) = Col(M)

Face(M) = S ⇒
Environment::CellNature(Envi(M),Col(M)-1,Row(M)) ∈ {EMP,DWO}
and Col(M)-1 ≥ 0
and Environment::CellContent(Envi(M),Col(M)-1,Row(M)) = No
⇒ Row(StrafeR(M)) = Row(M) and Col(StrafeR(M)) = Col(M)-1

Face(M) = S ⇒
Environment::CellNature(Envi(M),Col(M)-1,Row(M)) ∉ {EMP,DWO}
or Col(M)-1 < 0
or Environment::CellContent(Envi(M),Col(M)-1,Row(M)) ≠ No
⇒ Row(StrafeR(M)) = Row(M) and Col(StrafeR(M)) = Col(M)

[TurnL]:
Face(M)=N ⇒ Face(TurnL(M)) = W
Face(M)=W ⇒ Face(TurnL(M)) = S
Face(M)=S ⇒ Face(TurnL(M)) = E
Face(M)=E ⇒ Face(TurnL(M)) = N

[TurnR]:
Face(M)=N ⇒ Face(TurnR(M)) = E
Face(M)=E ⇒ Face(TurnR(M)) = S
Face(M)=S ⇒ Face(TurnR(M)) = W
Face(M)=W ⇒ Face(TurnR(M)) = N

Service: Entity includes Mob

Observator:

Hp: [Entity] → int

Damage: [Entity] → int

Constructors:

Init: Environment x int x int x Dir x int x int → [Entity]

Pre init(E,x,y,D,h) requires h > 0 and h ≥ 0 and 0 ≤ x < Environment::Width(E) and 0 ≤ y < Environment::Height(E)

Operators:

Step: [Entity] → [Entity]

Attack: [Entity] → [Entity]

SetHp: [Entity] x int → [Entity]

Pre SetHp(E,h) requires h ≥ 0

Observations:

[init]:

Hp(init(E,x,y,D,h,d)) = h

Damage(init,E,x,y,D,h,d)) = d

[attack]:

Face(E) = N and Row(E)+1 < Height(Env(E)) and CellContent(Env(E),Row(E)+1,Col(E)) ≠ No ⇒ Hp(CellContent(Env(Attack(E)),Col(Attack(E)),Row(Attack(E))+1)) = Hp(CellContent(Env(E),Col(E),Row(E)+1)) - Damage(E)

Face(E) = S and Row(E)-1 ≥ 0 and CellContent(Env(E),Col(E),Row(E)-1) ≠ No ⇒ Hp(CellContent(Env(Attack(E)),Col(Attack(E)),Row(Attack(E))-1)) = Hp(CellContent(Env(E),Col(E),Row(E)-1)) - Damage(E)

Face(E) = W and Col(E)-1 ≥ 0 and CellContent(Env(E),Col(E)-1,Row(E)) ≠ No ⇒ Hp(CellContent(Env(Attack(E)),Col(Attack(E))-1,Row(Attack(E)))) = Hp(CellContent(Env(E),Col(E)-1,Row(E))) - Damage(E)

Face(E) = E and Col(E)+1 < Width(Env(E)) and CellContent(Env(E),Col(E)+1,Row(E)) ≠ No ⇒ Hp(CellContent(Env(Attack(E)),Col(Attack(E))+1,Row(Attack(E)))) = Hp(CellContent(Env(E),Col(E)+1,Row(E))) - Damage(E)

Service: Cow includes Entity

Constructor: init: Environment x int x int x Dir x int x int → [Entity]

Pre init(E,x,y,D,h) requires 4 ≥ h ≥ 3

Observations:

[Step]:

Col(M) -1 ≤ Col(step(M)) ≤ Col(M)+1

Row(M) -1 ≤ Row(step(M)) ≤ Row(M)+1

[Attack]:

Face(E) = N and Row(E)+1 < Height(Env(E)) and CellContent(Env(E),Row(E)+1,Col(E)) ≠ No ⇒

Hp(CellContent(Env(Attack(E)),Col(Attack(E)),Row(Attack(E))+1)) = Hp(CellContent(Env(E),Col(E),Row(E)+1))

Face(E) = S and Row(E)-1 ≥ 0 and CellContent(Env(E),Col(E),Row(E)-1) ≠ No ⇒ Hp(CellContent(Env(Attack(E)),Col(Attack(E)),Row(Attack(E))-1)) = Hp(CellContent(Env(E),Col(E),Row(E)-1))

Face(E) = W and Col(E)-1 ≥ 0 and CellContent(Env(E),Col(E)-1,Row(E)) ≠ No ⇒ Hp(CellContent(Env(Attack(E)),Col(Attack(E))-1,Row(Attack(E)))) = Hp(CellContent(Env(E),Col(E)-1,Row(E)))

Face(E) = E and Col(E)+1 < Width(Env(E)) and CellContent(Env(E),Col(E)+1,Row(E)) ≠ No ⇒ Hp(CellContent(Env(Attack(E)),Col(Attack(E))+1,Row(Attack(E)))) = Hp(CellContent(Env(E),Col(E)+1,Row(E)))

Service: Player includes Entity

Observator:

LastCom: [Player] → Option[Command]

Content: [Player] x int x int → Option[Entity]

Pre Content(P,x,y) requires x ∈ {-1, 0, 1} and y ∈ {-1, +3}

Nature: [Player] x int x int → Cell

Pre Nature(P,x,y) requires x ∈ {-1, 0, 1} and y ∈ {-1, +3}

Viewable: [Player] x int x int → Cell

Pre Viewable(P,x,y) requires x ∈ {-1, 0, 1} and y ∈ {-1, +3}

foundTreasure: Player → boolean

Observations:

[Invariant]:

Il faut vérifier que Col(P)+-u, Col(P)+-v, Row(P)+-u et Row(P)+-v ne dépasse pas la définition des dimensions

Face(P) = N ⇒ Content(P,u,v) = Environment::CellContent(Envi(P),Col(P)+u,Row(P)+v)

Face(P) = N ⇒ Nature(P,u,v) = Environment::CellNature(Envi(P),Col(P)+u,Row(P)+v)

Face(P) = S ⇒ Content(P,u,v) = Environment::CellContent(Envi(P),Col(P)-u,Row(P)-v)

Face(P) = S ⇒ Nature(P,u,v) = Environment::CellNature(Envi(P),Col(P)-u,Row(P)-v)

Face(P) = E ⇒ Content(P,u,v) = Environment::CellContent(Envi(P),Col(P)+v,Row(P)-u)

Face(P) = E ⇒ Nature(P,u,v) = Environment::CellNature(Envi(P),Col(P)+v,Row(P)-u)

Face(P) = W ⇒ Content(P,u,v) = Environment::CellContent(Envi(P),Col(P)-v,Row(P)+u)

Face(P) = W ⇒ Nature(P,u,v) = Environment::CellNature(Envi(P),Col(P)-v,Row(P)+u)

[step]:

LastCom(P)=FF ⇒ step(P)=Forward(P)
LastCom(P)=BB ⇒ step(P)=Backward(P)
LastCom(P)=RR ⇒ step(P)=StrifeR(P)
LastCom(P)=LL ⇒ step(P)=StrifeL(P)
LastCom(P)=TR ⇒ step(P)=TurnR(P)
LastCom(P)=TL ⇒ step(P)=TurnL(P)

Service: Engine

Observator:

Envi: [Engine] → Environment
Entities: [Engine] → Array[Entity]
getEntity: [Engine] x int → Entity
getPlayer: [Engine] → Player
isOut: [Engine] → boolean
isFinished: [Engine] → boolean
isLost: [Engine] → boolean

Constructor:

Init: Environment x Player → [Engine]

Operator:

removeEntity: [Engine] x int → [Engine]
Pre removeEntity(E,i) requires 0 ≤ i < size(Entities(E))
addEntity: [Engine] x Entity → [Engine]
step: [Engine] → [Engine]
Pre step() requires forall i in [0;size(Entities(E))-1],Entity::Hp(getEntity(E,i))>0
clean: [Engine] → [Engine]

Observations:

[Invariant]:
forall i in [0;size(Entities(E))-1], Entity::Envi(getEntity(E,i))=Envi(E)
forall i in [0;size(Entities(E))-1], Entity::Col(getEntity(E,i))=x and
Entity::Row(getEntity(E,i))=y
⇒ Environment::CellContent(Envi(E),x,y)=getEntity(E,i)
(Def) isOut = Environment::CellNature(Envi(E),i,j) == OUT, i == Player::Col(getPlayer(E)) and j == Player::Row(getPlayer(E))
(Def) isFinished = foundTreasure(getPlayer(E)) and isOut(E) and isLost(E)
(Def) isLost = Player::getHp(getPlayer(E)) ≤ 0

[removeEntity]:

size(Entities(removeEntity(E,i))) = size(Entities(E))-1

forall k in [0,i-1], getEntity(removeEntity(E,i),k)) = getEntity(E,k)

forall k in [i,size(Entities(E))-2], getEntity(removeEntity(E,i),k)) = getEntity(E,k+1)

[addEntity]:

size(Entities(addEntity(E,e))) = size(Entities(E))+1

forall k in [0,size(Entities(E))-1], getEntity(addEntity(E,e),k)) = getEntity(E,k)

getEntity(addEntity(E,e),size(Entities(E))) = e

[clean]:

forall i in [0;size(Entities(E))-1],Entity::Hp(getEntity(E,i))>0

Rapport de projet

Choix d'implantation et difficultés de spécification

Tout d'abord, nous avons écrit les différents contrats en suivant la spécification donnée, cependant nous nous sommes rendus compte que cette spécification était parfois incohérente ou incomplète ; étant donnée qu'elle n'est pas issue de notre travail, nous avons pris un temps non négligeable à corriger et à compléter. En effet, nous avons dû corriger, entre autres, forward qui vérifie pour le Nord la row+1 et Width alors qu'il faudrait que cela soit Height d'après la spécification plus haut. Nous avons aussi dû vérifier dans les invariants de Player que Col()+-u et Col()+-v ne dépasse pas la définition du donjon. L'environnement, tel qu'il est décrit dans la spécification, ne permet pas la modification de cellule. Or elle nous a indispensable tant pour effectuer des tests que pour poser les Cellules etc...

Bien que nous ayons pris un certains temps pour compléter la spécification, nous y sommes parvenus. Cependant, nous avons appris assez tard que le viewable de Player était spécifier uniquement lorsqu'il regardait le Nord, ainsi en raison du temps qu'il nous restait nous avons donc décidé de le laisser ainsi. Cependant, nous aurions pu simplement écrire les autres cas si nous avions eu assez de temps.

D'un autre côté, certaines spécifications n'étaient pas simple à retranscrire en code, par exemple, isReachable, qui vérifie que deux points sont accessibles. Nous avons réussi à écrire une fonction récursive qui effectue un parcours en profondeur jusqu'à arriver au point désiré. C'est un algorithme récursif pour les quatre directions possibles qui garde en mémoire le chemin qui sera vérifié par la suite dans le contrat.

Une fois la spécification implantée, nous devions l'enrichir de certaines fonctionnalités. Nous avons ajouté le trésor en tant que type Cell.TRS et un observateur sur Player pour indiquer si le joueur a récupéré le trésor via une commande (pick il ramasse le trésor devant lui s'il existe).

En ce qui concerne les conditions de victoire et défaite, nous avons décidé qu'il y a défaite lorsque le joueur n'a plus de vie, c'est-à-dire getHp() ≤ 0. Nous avons donc ajouté un observateur indiquant s'il est mort dans Player, d'un autre côté nous avons ajouté dans engine un observateur isOut indiquant si le joueur est dans la Cellule de sortie et isFinished qui vérifie que le joueur est vivant, qu'il a récupéré le trésor et qu'il est dans la sortie. Il s'agit là juste d'un "and" des observateurs. De plus, le joueur étant une entité particulière nous avons décidé d'ajouter dans EngineService un observateur permettant de récupérer le Player.

En ce qui concerne les combats, nous avons choisi d'ajouter dans le service Entity la méthode "attack". En effet, nous avons d'abord pensé à l'ajouter dans MobService, cependant d'un point de vu sémantique, il s'agit que d'un objet mouvant. De plus, attack nécessite d'avoir des points de vie, et nous sommes arrivés à ajouter cette méthode dans EntityService. Ceci a induit un autre changement, tous les Option<MobService> ont été convertis en Option<EntityService> car sans cela il nous aurait été impossible d'utiliser la méthode attack et de voir les points de vies des entités à moins de les caster, ce que nous voulions à tout prix éviter. Ainsi, attack pour un Player attack la cible devant lui s'il existe, par contre pour un monstre il attack de tous les côtés sauf en diagonales.

Ces ajouts et modifications nous ont pris un certain temps à ajouter puisqu'il faut modifier en cascade les fonctionnalités ajoutées, à partir des interfaces, nous devons remonter aux décorateurs puis modifier les contrats pour enfin compléter l'implantation. Sans compter que devons refaire tourner les tests afin de vérifier la non régression et la validité des contrats ainsi modifiés.

Concernant l’interface graphique, nous avons décidé de prendre une vue du dessus, à la troisième personne. Nous avons utilisé des sprites pour le graphisme. Finalement, par manque de temps, nous ne proposons aucune map pré-conçue : pour jouer, l’utilisateur doit créer sa map. Une fois la map créée et validée, l’utilisateur peut jouer. Le jeu fonctionne correctement. La barre de vie du joueur n’est pas affichée. Il part avec 20 points de vie, les ennemis lui font un mal de 1 point. Les ennemis ont 4 points de vie et le joueur fait un mal de 2 points. La map est visible toute entière, car isVisible n’est écrit que pour la direction Nord.

Test pertinents

Nous avons écrit des tests suivant le Model-base testing (MBT). Cependant, pour des raisons de temps, d'efficacité et du nombre de tests à écrire, nous avons pris la décision d'indiquer uniquement les tests de préconditions et les tests de transitions, qui nous semblent ici les plus importants.

Nous avons écrit des tests unitaires qui vérifient que les préconditions et les transitions sont correctes, c’est-à-dire qu’une exception de précondition est levée lorsque le contrat n’est pas respecté ou que l’implantation ne respecte pas les valeurs attendues.

```java
@Test
public void setCellContentTransitionTest_1() {
    //cas positif
    try {
        env.init(10, 20);
        EntityService m = new Cow();
        m.init(env, 5, 7, Dir.N, 2, 10);
        @SuppressWarnings("unchecked")
        Option<EntityService> map[][] = new Option[10][20];
        for(int i = 0; i < map.length; i++){
            for(int j = 0; j < map[0].length; j++){
                map[i][j] = env.getCellContent(i, j);
            }
        }
        env.setCellContent(5, 5, m);

        assertTrue(env.getCellContent(5, 5).getValue() == m);
        for(int i = 0; i < 10; i++) {
            for(int j = 0; j < 20; j++) {
                if(i != 5 || j != 5) {
                    assertTrue(map[i][j].getValue() == env.getCellContent(i, j).getValue());
                }
            }
        }
    }catch(InvariantError ie) {
        fail();
    }
}
```

Nous avons aussi des tests qui permettent de vérifier des points plus élaboré tel que isReachable.

```java
@Test
public void isReachableTransitionTest_1() {
    //cas positif
    try {
        ems.init(10, 20);
        ems.setNature(0, 0, Cell.IN);
        ems.setNature(0, 19, Cell.OUT);
        assertTrue(ems.isReachable(0, 0, 0, 19));
    }catch(PreconditionError ex) {
        fail();
    }catch(InvariantError ie) {
        fail();
    }

}

@Test
public void isReachableTransitionTest_2() {
    //cas negatif
    try {
        ems.init(10, 20);
        ems.setNature(0, 0, Cell.IN);
        ems.setNature(1, 0, Cell.WLL);
        ems.setNature(1, 1, Cell.WLL);
        ems.setNature(0, 1, Cell.WLL);
        ems.setNature(9, 19, Cell.OUT);
        assertTrue(!ems.isReachable(0, 0, 9, 19));
    }catch(PreconditionError ex) {
        fail();
    }catch(InvariantError ie) {
        fail();
    }

}
```

Nous vérifions dans le cas positif que la position est bien atteignable, dans l'autre cas, nous enfermons le point dans des murs et vérifions qu'il n'est pas possible de passer.

Il y a aussi un cas de scénario de jeu existant qui cherche le trésor et va à la sortie, nous vérifions que les conditions de victoire sont correctes.

```java
@Test
public void GameTransTest_1() {
    //cas positif
    try {
        PlayerService player = new Player();
        player.init(env, 7, 2, Dir.N, 100, 10);
        es.init(env, player);
        player.strafeL();
        player.strafeL();
        player.forward();
        player.forward();
        player.forward();
        player.forward();
        player.pickItem();
        player.forward();
        player.forward();
        player.forward();
        player.forward();
        player.forward();
        player.forward();
        player.forward();
        player.forward();
        player.strafeR();
        player.strafeR();
        player.strafeR();
        player.strafeR();
        assertTrue(es.isFinished());
    }catch(PreconditionError pe) {
        fail();
    }catch(InvariantError e) {
        fail();
    }
```

Finalement le fichier RunningGameTest, nous a permis de faire quelques tests informels, puisqu'il s'agit d'une instanciation du jeu en ASCII avec les contrats et qui attend une commande à effectuer à chaque tour.