Rapport du Projet CPS 2018
Spécification et implémentation du jeu Dungeon Master

Naim CHOULLIT, Mohammed Charafeddine LACHACHI

18 avril 2014

Contents

1 Introduction

Dans le cadre de l'unité d'enseignement CPS, nous été demandé réaliser le jeu de rôle Dungeon Master, L'objectif du projet est de spécifier et construire des tests pour l'ensemble des services réalisées,

2 Spécification

La spécification des services est la partie la plus importante de ce projet, pour cela nous lui avons consacré la majorité du temps. Le projet à son début contenait les six services suivants : Map, EditMap, Environnement, Mob, Entity, Cow, Player et Engine. En plus de ces services nous avons ajouté le service Ressources représentant le trésor que doit récupérer le joueur pour pouvoir s'échapper du labyrinthe et aussi le service Key qui est un simple refine du service Ressources.

2.1 Ressources Service

Service: Ressources
Observators : Col : [Ressources] → int
Row : [Ressources] → int
Row : [Ressources] → Environement
Constructors: init : Environement → [Ressources]
pre init(e) **requires** 0¡Col(S)¡EnvironnementService::Width(K,getEnv(K))
0¡Row(S)¡EnvironnementService::Height(K,getEnv(K))
Observations
Invariants: Environnement::CellNature(getEnv(K),Col(S), Row(K)) in EMP
Environemment::CellNature(Envi(S),u,v) in IN
implies Environnement::isReachable(Envi(S),u,v, Col(S), Row(S))
[init]: Envi(init(e)) = e
Environnement::CellNature(Envi(S), Col(s), Row(S) in $\{EMP, DNC, DWC\}$

Service: Key refine Ressources

2.2 Map Service

Service: Map
Types: bool, int, Cell
Observators: **const** Height: $[Map] \to int$
const Width: $[Map] \to int$
CellNature: $[Map] \times int \times int \to Cell$
pre CellNature(M, x, y) **requires** $0 \leq x < Width(M)$ and $0 \leq y < Height(M)$
Constructors: **init:** $int \times int \to [Map]$
pre init(w,h) requires 0 ¡ w and 0 ¡ h
Operators: **OpenDoor:** $[Map] \times int \times int \to [Map]$
pre OpenDoor(M, x, y) **requires** CellNature$(M, x, y) \in DNC, DWC$
CloseDoor: $[Map] \times int \times int \to [Map]$
pre $CloseDoor(M, x, y)$ **requires** CellNature$(M, x, y) \in \{DNO, DWO\}$
Observation:
[**Invariant**]: $\top$
[**Init**]: Height(init(h,w)) = h
Width(init(h,w)) = w
[**OpenDoor**]: CellNature(M,x,y) = DWC **implies** CellNature(OpenDoor(M,x,y),x,y) = DWO
CellNature(M,x,y) = DNC **implies** 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)
implies CellNature(OpenDoor(M,x,y),u,v) = CellNature(M,u,v)
[**CloseDoor**]: CellNature(M,x,y) = DWO **implies** CellNature(OpenDoor(M,x,y),x,y) = DWC
CellNature(M,x,y) = DNO **implies** 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)
implies CellNature(OpenDoor(M,x,y),u,v) = CellNature(M,u,v)

2.3 EditMap Service

Service: EditMap **refine** Map
Types: bool, int, Cell
Observators: isReachable: $[EditMap] \times int \times int \times int \times int \rightarrow bool$
pre isReachable(M,x1,y1,x2,y2) **requires** CellNature(M,x1,y1) $\neq$ WLL **and** CellNature(M,x2,y2)
isReady: [EditMap] $\rightarrow$ bool
Constructors: $\emptyset$
Operators: SetNature: [EditMap]×int×int×Cell $\rightarrow$ [EditMap]
pre SetNature(M,x,y) **requires** 0≤x¡Width(M) **and** 0≤y¡ Height(M)
Observations:
[**Invariants**]: 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 **in** [1;size(P)-1], (P[i-1]=(u,v) **and** P[i]=(s,t)) **implies** $(u-s)^2 + (v-t)^2 = 1$
and forall i in [1;size(P)-2], P[i-1]=(u,v) **implies** CellNature(M,u,v) $\neq$ WLL
isReady(M) = **exists** xi,yi,xo,yo **in** int^4 , CellNature(M,xi,yi) = IN
and CellNature(M,xi,yi) = OUT
and isReachable(M,xi,yi,xo,yo)
and forall x,y in int 2 , x $\neq$ xi or y $\neq$ yi **implies** CellNature(M,x,y) $\neq$ IN
and forall x,y in int 2 , x $\neq$ xo or y $\neq$ yo **implies** CellNature(M,x,y) $\neq$ OUT
forall x,y in int, CellNature(M,x,y) $\in \{DNO, DNC\}$
implies 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) $\in \{DWO, DWC\}$
implies CellNature(M,x+1,y) = CellNature(M,x-1,y) = WLL
and CellNature(M,x,y-1) = CellNature(M,x,y+1) = EMP
[**SetNature**]: CellNature(SetNature(M,x,y,Na),x,y) = Na
forall u,v in int^2 , u $\neq$ x or v $\neq$ y
implies CellNature(SetNature(M,x,y),u,v) = CellNature(M,u,v)

2.4 Environement Service

Service: Environement **include** EditMap
Types: bool, int, Cell, Mob
Observators: CellContent: int × int $\rightarrow$ Option[Mob]
CellRessources: int × int $\rightarrow$ Option[Ressource]
mob : int × int $\rightarrow$ Mob
ressource : int × int $\rightarrow$ Ressource
Operators: CloseDoor: [Environment] × int × int × [Environment]
pre CloseDoor(M,x,y) **requires** CellContent(M,x,y) = No

2.5 Mob Service

Service: Mob
Types: bool, int, Cell
Observators: Env: [Mob] → Environment
Col: [Mob] → int
Row: [Mob] → int
Face: [Mob] → Dir
Constructors: init: Environment × **in**t × **in**t × Dir → [Mob]
pre init(E,x,y,D) **require**s 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]
Attack: [Mob] → [Mob]
[Observations] :
[invariant] : 0 ≤ Col(M) ¡ Environment::Width(Envi(M))
0 ≤ Row(M) ¡ Environment::Height(Envi(M))
Environment::CellNature(Envi(M),Col(M),Row(M)) **in** $\{WLL, DNC, DWC\}$
[**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
[Forward]: Face(M)=S **implies**
Environment::CellNature(Envi(M),Col(M),Row(M)+1) ∈ $\{EMP, DWO\}$
and Row(M)+1 ¡ Environment::Width(Envi(M))
and Environment::CellContent(Envi(M),Col(M),Row(M)+1) = No
implies Row(Forward(M)) = Row(M) + 1
and Col(Forward(M)) = Col(M)
Face(M)=S **implies**
Environment::CellNature(Envi(M),Col(M),Row(M)+1) ∈ $\{EMP, DWO\}$
or Row(M)+1 ≥ Environment::Width(Envi(M))
or Environment::CellContent(Envi(M),Col(M),Row(M)+1) ≠ No
implies Row(Forward(M)) = Row(M) **and** Col(Forward(M)) = Col(M)
Face(M)=E **implies** Environment::CellNature(Envi(M),Col(M)+1,Row(M)) ∈ $\{EMP, DNO\}$
and Col(M)+1 ¡ Environment::Height(Envi(M))
and Environment::CellContent(Envi(M),Col(M)+1,Row(M)) = No
implies Row(Forward(M)) = Row(M) **and** Col(Forward(M)) = Col(M) + 1
Face(M)=E **implies**
Environment::CellNature(Envi(M),Col(M)+1,Row(M)) ∈ $\{EMP, DWO\}$
or Row(M) ≥ Environment::Width(Envi(M))
or Environment::CellContent(Envi(M),Col(M)+1,Row(M)) ≠ No
implies Row(Forward(M)) = Row(M) **and** Col(Forward(M)) = Col(M)

Face(M)=N **implies**
Environment::CellNature(Envi(M),Col(M),Row(M)-1) $\in \{EMP, DWO\}$
and Col(M)-1 $\geq$ 0
and Environment::CellContent(Envi(M),Col(M),Row(M)+1) = No
implies Row(Forward(M)) = Row(M) - 1 **and** Col(Forward(M)) = Col(M)
Face(M)=N **implies**
Environment::CellNature(Envi(M),Col(M),Row(M)-1) $\in \{EMP, DWO\}$ or Col(M)-1 ¡ 0
or Environment::CellContent(Envi(M),Col(M),Row(M)-1) $\neq$ No
implies Row(Forward(M)) = Row(M) **and** Col(Forward(M)) = Col(M)
Face(M)=W **implies**
Environment::CellNature(Envi(M),Col(M)-1,Row(M)) $\in \{EMP, DNO\}$
and Row(M)-1 $\geq$ 0
and Environment::CellContent(Envi(M),Col(M)-1,Row(M)) = No
implies Row(Forward(M)) = Row(M) **and** Col(Forward(M)) = Col(M) - 1
Face(M)=W **implies**
Environment::CellNature(Envi(M),Col(M)-1,Row(M)) $\in \{EMP, DNO\}$
or Row(M)-1 ¡ 0 or Environment::CellContent(Envi(M),Col(M),Row(M)-1) $\neq$ No
implies Row(Forward(M)) = Row(M) **and** Col(Forward(M)) = Col(M)

[Backward]: Face(M)=N **implies**
Environment::CellNature(Envi(M),Col(M),Row(M)+1) $\in \{EMP, DWO\}$
and Row(M)+1 ¡ Environment::Width(Envi(M))
and Environment::CellContent(Envi(M),Col(M),Row(M)+1) = No
implies Row(Backward(M)) = Row(M) + 1
and Col(Backward(M)) = Col(M)
Face(M)=N **implies**
Environment::CellNature(Envi(M),Col(M),Row(M)+1) $\in \{EMP, DWO\}$
or Row(M)+1 $\geq$ Environment::Width(Envi(M))
or Environment::CellContent(Envi(M),Col(M),Row(M)+1) $\neq$ No
implies Row(Backward(M)) = Row(M) **and** Col(Backward(M)) = Col(M)
Face(M)=W **implies** Environment::CellNature(Envi(M),Col(M)+1,Row(M)) $\in \{EMP, DNO\}$
and Col(M)+1 ¡ Environment::Height(Envi(M))
and Environment::CellContent(Envi(M),Col(M)+1,Row(M)) = No
implies Row(Backward(M)) = Row(M) **and** Col(Backward(M)) = Col(M) + 1
Face(M)=W **implies**
Environment::CellNature(Envi(M),Col(M)+1,Row(M)) $\in \{EMP, DWO\}$
or Row(M) $\geq$ Environment::Width(Envi(M))
or Environment::CellContent(Envi(M),Col(M)+1,Row(M)) $\neq$ No
implies Row(Backward(M)) = Row(M) **and** Col(Backward(M)) = Col(M)
Face(M)=S **implies**
Environment::CellNature(Envi(M),Col(M),Row(M)-1) $\in \{EMP, DWO\}$
and Col(M)-1 $\geq$ 0
and Environment::CellContent(Envi(M),Col(M),Row(M)+1) = No
implies Row(Backward(M)) = Row(M) - 1 **and** Col(Backward(M)) = Col(M)
Face(M)=S **implies**
Environment::CellNature(Envi(M),Col(M),Row(M)-1) $\in \{EMP, DWO\}$ or Col(M)-1 ¡ 0
or Environment::CellContent(Envi(M),Col(M),Row(M)-1) $\neq$ No
implies Row(Backward(M)) = Row(M) **and** Col(Backward(M)) = Col(M)
Face(M)=E **implies**
Environment::CellNature(Envi(M),Col(M)-1,Row(M)) $\in \{EMP, DNO\}$
and Row(M)-1 $\geq$ 0
and Environment::CellContent(Envi(M),Col(M)-1,Row(M)) = No
implies Row(Backward(M)) = Row(M) **and** Col(Backward(M)) = Col(M) - 1

Face(M)=E **implies**
Environment::CellNature(Envi(M),Col(M)-1,Row(M)) $\in \{EMP, DNO\}$
or Row(M)-1 ¡ 0 or Environment::CellContent(Envi(M),Col(M),Row(M)-1) $\neq$ No
implies Row(Backward(M)) = Row(M) **and** Col(Backward(M)) = Col(M)

[StrafeR]: Face(M)=N **implies**
Environment::CellNature(Envi(M),Col(M)+1,Row(M)) $\in \{EMP, DNO\}$
and Col(M)+1 ¡ Environment::Height(Envi(M))
and Environment::CellContent(Envi(M),Col(M)+1,Row(M)) = No
implies Row(StrafeR(M)) = Row(M) **and** Col(StrafeR(M)) = Col(M) + 1

Face(M)=N **implies**
Environment::CellNature(Envi(M),Col(M)+1,Row(M)) $\in \{EMP, DWO\}$
or Row(M) $\geq$ Environment::Width(Envi(M))
or Environment::CellContent(Envi(M),Col(M)+1,Row(M)) $\neq$ No
implies Row(StrafeR(M)) = Row(M) **and** Col(StrafeR(M)) = Col(M)

Face(M)=S **implies**
Environment::CellNature(Envi(M),Col(M)-1,Row(M)) $\in \{EMP, DNO\}$
and Row(M)-1 $\geq$ 0
and Environment::CellContent(Envi(M),Col(M)-1,Row(M)) = No
implies Row(StrafeR(M)) = Row(M) **and** Col(StrafeR(M)) = Col(M) - 1

Face(M)=S **implies**
Environment::CellNature(Envi(M),Col(M)-1,Row(M)) $\in \{EMP, DNO\}$
or Row(M)-1 $\geq$ 0 or Environment::CellContent(Envi(M),Col(M),Row(M)-1) $\neq$ No
implies Row(StrafeR(M)) = Row(M) **and** Col(StrafeR(M)) = Col(M)

[StrafeL]: Face(M)=S **implies**
Environment::CellNature(Envi(M),Col(M)+1,Row(M)) $\in \{EMP, DNO\}$
and Col(M)+1 ¡ Environment::Height(Envi(M))
and Environment::CellContent(Envi(M),Col(M)+1,Row(M)) = No
implies Row(StrafeL(M)) = Row(M) **and** Col(StrafeL(M)) = Col(M) + 1

Face(M)=S **implies**
Environment::CellNature(Envi(M),Col(M)+1,Row(M)) $\in \{EMP, DWO\}$
or Row(M) $\geq$ Environment::Width(Envi(M))
or Environment::CellContent(Envi(M),Col(M)+1,Row(M)) $\neq$ No
implies Row(StrafeL(M)) = Row(M) **and** Col(StrafeL(M)) = Col(M)

Face(M)=N **implies**
Environment::CellNature(Envi(M),Col(M)-1,Row(M)) $\in \{EMP, DNO\}$
and Row(M)-1 $\geq$ 0
and Environment::CellContent(Envi(M),Col(M)-1,Row(M)) = No
implies Row(StrafeL(M)) = Row(M) **and** Col(StrafeL(M)) = Col(M) - 1

Face(M)=N **implies**
Environment::CellNature(Envi(M),Col(M)-1,Row(M)) $\in \{EMP, DNO\}$
or Row(M)-1 $\geq$ 0 or Environment::CellContent(Envi(M),Col(M),Row(M)-1) $\neq$ No
implies Row(StrafeL(M)) = Row(M) **and** Col(StrafeL(M)) = Col(M)

[TurnLeft]: Face(M)=N **implies** Face(TurnLeft(M))=W
Face(M)=W **implies** Face(TurnLeft(M))=S
Face(M)=S **implies** Face(TurnLeft(M))=E
Face(M)=E **implies** Face(TurnLeft(M))=N

[TurnRight]: Face(M)=S **implies** Face(TurnRight(M))=W
Face(M)=E **implies** Face(TurnRight(M))=S
Face(M)=N **implies** Face(TurnRight(M))=E
Face(M)=W **implies** Face(TurnRight(M))=N

2.6 Entity Service

Service: Entity **include** Mob
Observators: Hp: [Entity] → int
Constructors: init: Environment × int × int × Dir × int → [Entity]
pre init(E,x,y,D,h) **requires** h ¿ 0
Operators: step: [Entity] → [Entity]
Observations:

[init]: Hp(init(E,x,y,D,h)) = h

[attack]: Face(E) = N **and** Environment::CellContent(Envi(E),Col(E),Row(E)-1) ≠ No **implies** HP(Attack(Environment::CellContent(Envi(E),Col(E),Row(E)-1)))) = HP(Environment::CellContent(Envi(E),Col(E),Row(E)-1)) - 1

Face(E) = S **and** Environment::CellContent(Envi(E),Col(E),Row(E)+1) ≠ No **implies** HP(Environment::CellContent(Envi(E),Col(E),Row(E)+1)) = HP(Environment::CellContent(Envi(E),Col(E),Row(E)+1))@pre - 1

Face(E) = E **and** Environment::CellContent(Envi(E),Col(E)+1,Row(E)) ≠ No **implies** HP(Environment::CellContent(Envi(E),Col(E)+1,Row(E))) = HP(Environment::CellContent(Envi(E),Col(E)+1,Row(E)))@pre - 1

Face(E) = W **and** Environment::CellContent(Envi(E),Col(E)-1,Row(E)) ≠ No **implies** HP(Environment::CellContent(Envi(E),Col(E)-1,Row(E))) = HP(Environment::CellContent(Envi(E),Col(E)-1,Row(E)))@pre - 1

2.7 Cow Service

Service: Cow **include** Entity
Constructors: init: Environment × int × int × Dir × int → [Entity]
pre init(E,x,y,D,h) requires 4 ≥ h ≥ 3
Opertators: Chase: [Cow] → [Cow]
Observations:

[**step:**] Col(M) - 1 ≤ Col(step(M)) ≤ Col(M) + 1
Row(M) - 1 ≤ Row(step(M)) ≤ Row(M) + 1

2.8 Player Service

Comme nous pouvons le voir dans la spécification du joueur présentée ci-dessous, nous avons ajouté les commandes C, CLOSE, et OPEN afin de permettre au joueur d'attaquer un monstre, d'ouvrir une porte et de la fermer.

Service: Player **include** Entity

Observators:
LastCom: [Player] → Option[Command]
Content: [Player] × int × int → Option[Mob]
pre Content(P,x,y) **requires** x ∈ {−1, 0, 1} and y ∈ {−1, +3}
Nature: [Player] × int × int → Cell
pre Nature(P,x,y) **requires** x ∈ {−1, 0, 1} and y ∈ {−1, +3}
Viewable: [Player] × int × int → Cell
pre Nature(P,x,y) **requires** x ∈ {−1, 0, 1} and y ∈ {−1, +3}
Ressource : [Player] → Ressource
Key: [Player] → boolean
Win: [Player] → boolean
pre Win(P) **requires** Ressource(P) = TRESOR
Dead: [Player] → boolean
pre Dead(P) **requires** Hp(P) ≤ 0

Operators:
openDoor:[Player] → [Player]
pre OpenDoor(P) **require** Key(P) = true **and**
Face(P) = N **implies** Environement::CellNature(Envi(P), Col(p), Row(p) - 1) ∈ {*DWC*}
and Environement::CellContent(Envi(P), Col(p), Row(p) - 1) = NO

Face(P) = E **implies** Environement::CellNature(Envi(P), Col(p) + 1, Row(p)) ∈ {*DNC*}
and Environement::CellContent(Envi(P), Col(p) + 1, Row(p)) = NO

Face(P) = S **implies** Environement::CellNature(Envi(P), Col(p), Row(p) + 1) ∈ {*DWC*}
and Environement::CellContent(Envi(P), Col(p), Row(p) + 1) = NO

Face(P) = W **implies** Environement::CellNature(Envi(P), Col(p) - 1, Row(p)) ∈ {*DNC*}
and Environement::CellContent(Envi(P), Col(p) - 1, Row(p)) = NO

CloseDoor:[Player] → [Player]
pre CloseDoor(P) **require**

Face(P) = N **implies** Environement::CellNature(Envi(P), Col(p), Row(p) - 1) ∈ {*DWO*}
and Environement::CellContent(Envi(P), Col(p), Row(p) - 1) = NO

Face(P) = E **implies** Environement::CellNature(Envi(P), Col(p) + 1, Row(p)) ∈ {*DNO*}
and Environement::CellContent(Envi(P), Col(p) + 1, Row(p)) = NO

Face(P) = S **implies** Environement::CellNature(Envi(P), Col(p), Row(p) + 1) ∈ {*DWC*}
and Environement::CellContent(Envi(P), Col(p), Row(p) + 1) = NO

Face(P) = W **implies** Environement::CellNature(Envi(P), Col(p) - 1, Row(p)) ∈ {*DNC*}
and Environement::CellContent(Envi(P), Col(p) - 1, Row(p)) = NO **Observations:**

[**Invariants**]: Face(P) = N
implies Content(P,u,v) = Environment:CellContent(Envi(P),Col(P)+u,Row(P)+v)
Face(P) = N
implies Nature(P,u,v) = Environment:CellNature(Envi(P),Col(P)+u,Row(P)+v)
Face(P) = S
implies Content(P,u,v) = Environment:CellContent(Envi(P),Col(P)-u,Row(P)-v)
Face(P) = S

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

Face(P) = W
implies Content(P,u,v) = Environment:CellContent(Envi(P),Col(P)-v,Row(P)+u)
Face(P) = W
implies Nature(P,u,v) = Environment:CellNature(Envi(P),Col(P)-v,Row(P)+u)
forall u,v in [-1,1] × [-1,1], not Viewable(P,u,v)
Viewable(P,-1,2)= Nature(P,-1,1) $\notin \{WALL, DWC, DNC\}$
Viewable(P,0,2)= Nature(P,0,1) $\notin \{WALL, DWC, DNC\}$
Viewable(P,1,2)= Nature(P,1,1) $\notin \{WALL, DWC, DNC\}$
Viewable(P,-1,3)= Nature(P,-1,2) $\notin \{WALL, DWC, DNC\}$ and Viewable(P,-1,2)
Viewable(P,0,3)= Nature(P,0,2) $\notin \{WALL, DWC, DNC\}$ and Viewable(P,0,2)
Viewable(P,1,3)= Nature(P,1,2) $\notin \{WALL, DWC, DNC\}$ and Viewable(P,1,2)

[**openDoor**]: Key(OpenDoor(P)) = true
Face(P) = N **implies** Environement::CellNature(Envi(P), Col(p), Row(p) - 1) $\in \{DWO\}$
Face(P) = E **implies** Environement::CellNature(Envi(P), Col(p) + 1, Row(p)) $\in \{DNO\}$
Face(P) = S **implies** Environement::CellNature(Envi(P), Col(p), Row(p) + 1) $\in \{DWO\}$
Face(P) = W **implies** Environement::CellNature(Envi(P), Col(p) - 1, Row(p)) $\in \{DNO\}$

[**ClooseDoor**]: Key(OpenDoor(P)) = Key(p)
Face(P) = N **implies** Environement::CellNature(Envi(P), Col(p), Row(p) - 1) $\in \{DWC\}$
Face(P) = E **implies** Environement::CellNature(Envi(P), Col(p) + 1, Row(p)) $\in \{DNC\}$
Face(P) = S **implies** Environement::CellNature(Envi(P), Col(p), Row(p) + 1) $\in \{DWC\}$
Face(P) = W **implies** Environement::CellNature(Envi(P), Col(p) - 1, Row(p)) $\in \{DNC\}$

[**step**]: LastCom(P)=FF **implies** step(P) = Forward(P)
LastCom(P)=BB **implies** step(P) = Backward(P)
LastCom(P)=LL **implies** step(P) = StrafeLeft(P)
LastCom(P)=RR **implies** step(P) = StrafeRight(P)
LastCom(P)=TL **implies** step(P) = TurnLeft(P)
LastCom(P)=TR **implies** step(P) = TurnRight(P)
LastCom(P)=C **implies** step(P) = Attack(P)
LastCom(P)=CLOSE **implies** step(P) = ColseDoor(P)
LastCom(P)=OPEN **implies** step(P) = OpenDoor(P)

2.9 Engine Service

Service: Engine
Observator: `Envi`: [Engine] → Environment
`Entities`: [Engine] → Array[Entity]
`getEntity`: [Engine] × int → Entity
Constructor: `init`: Environment → [Engine]
Operator: `removeEntity`: [Engine] × int → [Engine]
pre `removeEntity(E,i)` **requires** $0 \leq$ `i` $<$ `size(Entities(E))`
`addEntity`: [Engine] × Entity → [Engine]
`step`: [Engine] → [Engine]
pre `step()` **requires**
forall i **in** [0;`size(Entities(E))-1`], `Entity::Hp(getEntity(E,i))>0`
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`
implies `Environment::CellContent(Envi(E),x,y)` = `getEntity(E,i)`
[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

3 Implémentation

La partie d'implémentation n'était pas la plus dure à réaliser, mais elle comprenait quelques difficultés parmi eux, l'algorithme de l'observateur **isReachable** du service **EditMap**.

3.1 IsReachable

Cet observateur permet à la base de déterminer si un chemin existe entre l'entrée de la map et sa sortie. Pour des raisons d'optimisations algorithmiques, nous avons choisis la programmation dynamique comme solution a ce problème.
En plus de l'entrée et la sortie de la map, nous avons ajouté à cette dernière un trésor et une clef, ces deux ressources doivent être atteignable par le joueur, donc nous avons utilisé le même algorithme afin de bien les placer sur la map.
Cette méthode est indispensable pour déterminer si une map est prête ou pas à être explorée par notre héros. Le code source de cet observateur est accessible dans le package **src.components.TestReachable.java**

3.2 Génération aléatoire des cellules et des portes

Afin de permettre à l'utilisateur de jouer directement à des maps qui sont générées aléatoirement, nous avons ajouté les deux opérations **initCells** et **initDoors** dans le service EditMap.
initCells : permet selon un algorithme aléatoire de placer au début les murs et les case vide dans map, et à la fin l'entrée et la sortie qui sont respectivement placées sur les case (0,0) et (Height-1,Width-1).
initDoor : Afin de placer les portes sur la map nous avons utiliser cette opération, qui permet de générer les porte aléatoirement en prenant en considération les contraintes des cases voisines pour chaque type de porte.

Ces deux algorithme génères respectivement les cellules et les porte, d'une façon à respecter l'atteignabilité de la sortie par le joueur.

3.3 Tests

Dans tous développement de solution informatique les test est l'une des parties les plus importantes. Pour cela en plus de la spécification et l'implementation des services. nous avons lis en place des test *MBT*. Pour chaque service vous trouverez dans le package une classe **TestService** permettant de lancer le test, et une classe abstraite **AbstractServiceTest** contenant l'implémentation des tests.

Parmi les tests pertinent nous pouvons citer celui qui permet de tester la postcondition de combat entre le héros et le monstre. C'est à dire que la santé du monstre à bien diminuéé après un coup donné par le héros.

```
@Test
public void attackTestPost1Pos(){
   EngineService labyrinthe = new EngineContract(new Engine());
   Dir dir = Dir.E;
   int x = 0;
   int y = 0;
   int h = 10;
   CowService cow = new CowContract(new Cow());
   EnvironnementService env = new EnvironnementContract(new Environement());
   env.init(15, 15);
   labyrinthe.init(env);
   player.init(env, x, y, dir,h);
   labyrinthe.addEntity(player);
   int row = player.getRow();
   int col = player.getCol() + 1;
   cow.init(env, row, col, Dir.W, 4);
   labyrinthe.addEntity(cow);
   System.err.println(cow.getRow() +" " + cow.getCol());
   Optional<MobService> coww = env.getCellContent(row, col);
   System.out.println(coww);
   int hp = ((EntityService) coww.get()).getHp();
   player.attack();
   assertTrue(((EntityService) coww.get()).getHp() != hp - 1);
}
```

3.4 Fonctionnalité réalisées

3.4.1 Jeu

La version actuel du jeu permet de lancer le jeux avec une grille 15x15, cette dernière contient le joueur, un monstre et un trésor à récupérer pour s'évader du labyrinthe, une version modifier donnant la possibilité d'éditer les dimensions de la grille sera publier prochainement.

3.4.2 Monstre et combat

La grille contient un monstre qui se déplace aléatoirement dans le labyrinthe, une fois le héros est repéré par le monstre se dernier le poursuit pour se combattre.

3.4.3 Trésor

Au lancement de la partie un trésor est placer dans le labyrinthe, d'une façon qu'il soit atteignable par le héros.

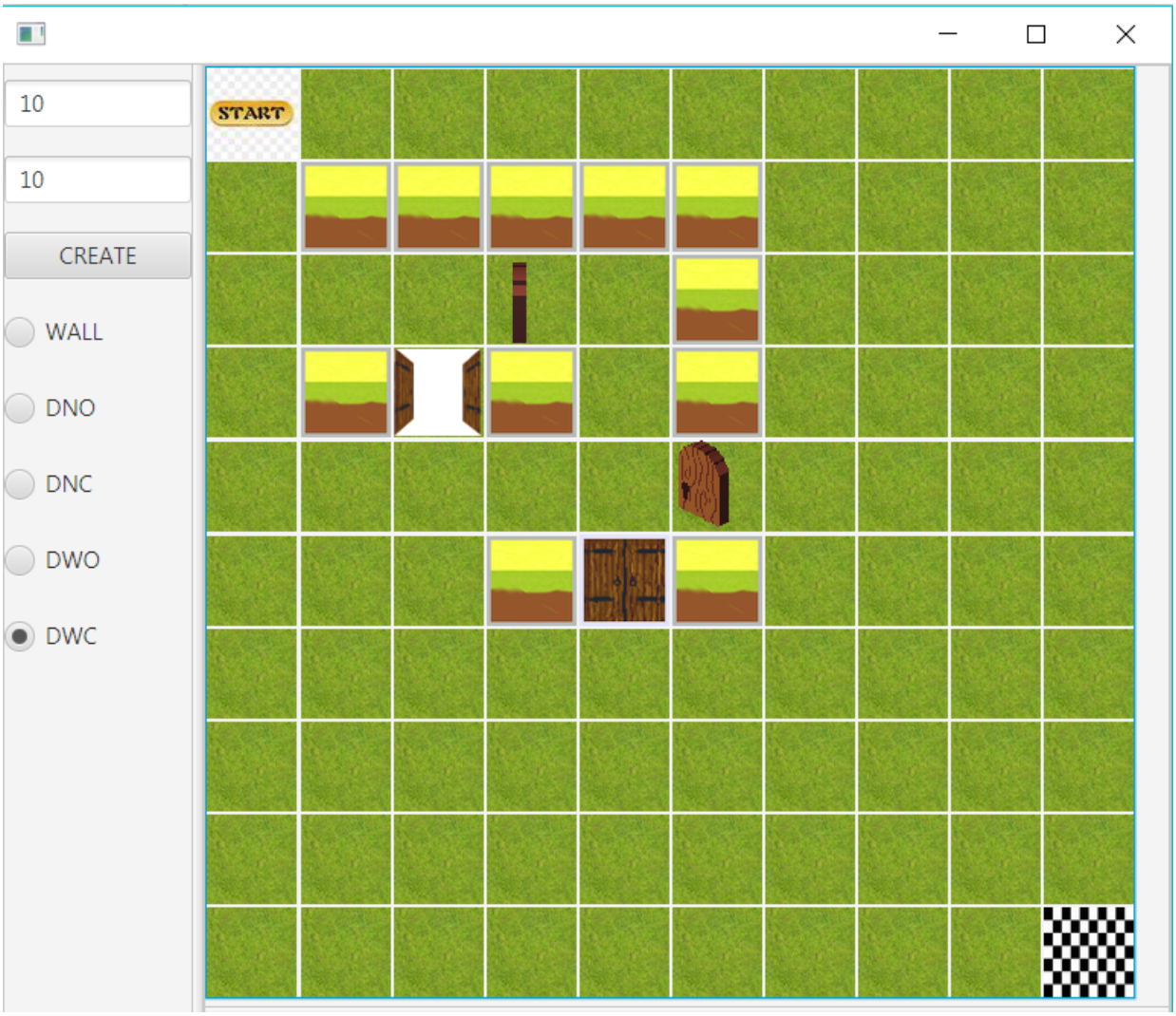
3.4.4 Affichage et Interface

Une interface 2D (vue d'en haut) a été intégrée au jeu, L'affichage de la map contient sept types de textures murs,sol,porte nord sud ouverte / fermée , porte est ouest ouverte/ fermé, zone de départ, zone d'arriver, clef et

trésor, chacune représenté par une image intégrée à l'interface graphique. La version actuelle de l'affichage permet à l'utilisateur de voir l'ensemble du labyrinthe.

3.4.5 Gestion des grilles

Une option permettant de créer des grille à été intégrée, cette dernière est accessible via la classe **src.application.MainE**
La version actuelle du jeu permet seulement d'éditer une map via cette interface graphique, une prochaine version ajoutera la possibilité d'importer la map afin de l'utiliser pour jouer.



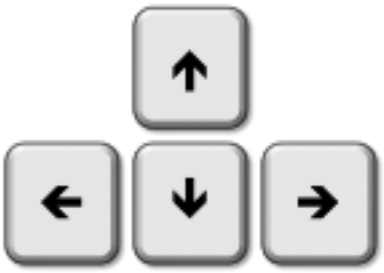
3.5 Extensions

3.5.1 Clef

Pour s'évader du labyrinthe le joueur doit passer par des portes qui ne sont pas tous le temps déverrouillées, donc au lancement du jeu une clef est placer dans la map d'une façon qu'elle soit atteignable. Cette clef permet d'ouvrir toutes les porte.

4 Présentation de l'interface

Comme nous l'avons déjà cité, l'interface graphique représente la grille complète du labyrinthe. Au lancement le joueur est placer sur la case (0,0), pour ce déplacer le joueur utilise les touche directionnelles du clavier



Une fois face à un monstre le joueurs à la possibilité de combattre en appuyant sur la touche **A du clavier**, si le monstre est abattu, ce dernier est enlevé de la grille, sinon si le héros est mort un message **SORRY YOU'R DEAD** est affiché.

Le joueurs peut aussi ouvrir et fermer les portes, et cela grâce à la clef placer dans la grille, une fois la clef récupérer le joueur peut ouvrir respectivement fermer une porte avec les bouton **O et C** du clavier.

N'oublions pas que le but principale du héros est de s'évader du labyrinthe, pour cela il doit tous d'abord récupérer le trésor **Pièce d'or** placée dans la grille.

5 Manuel d'utilisation

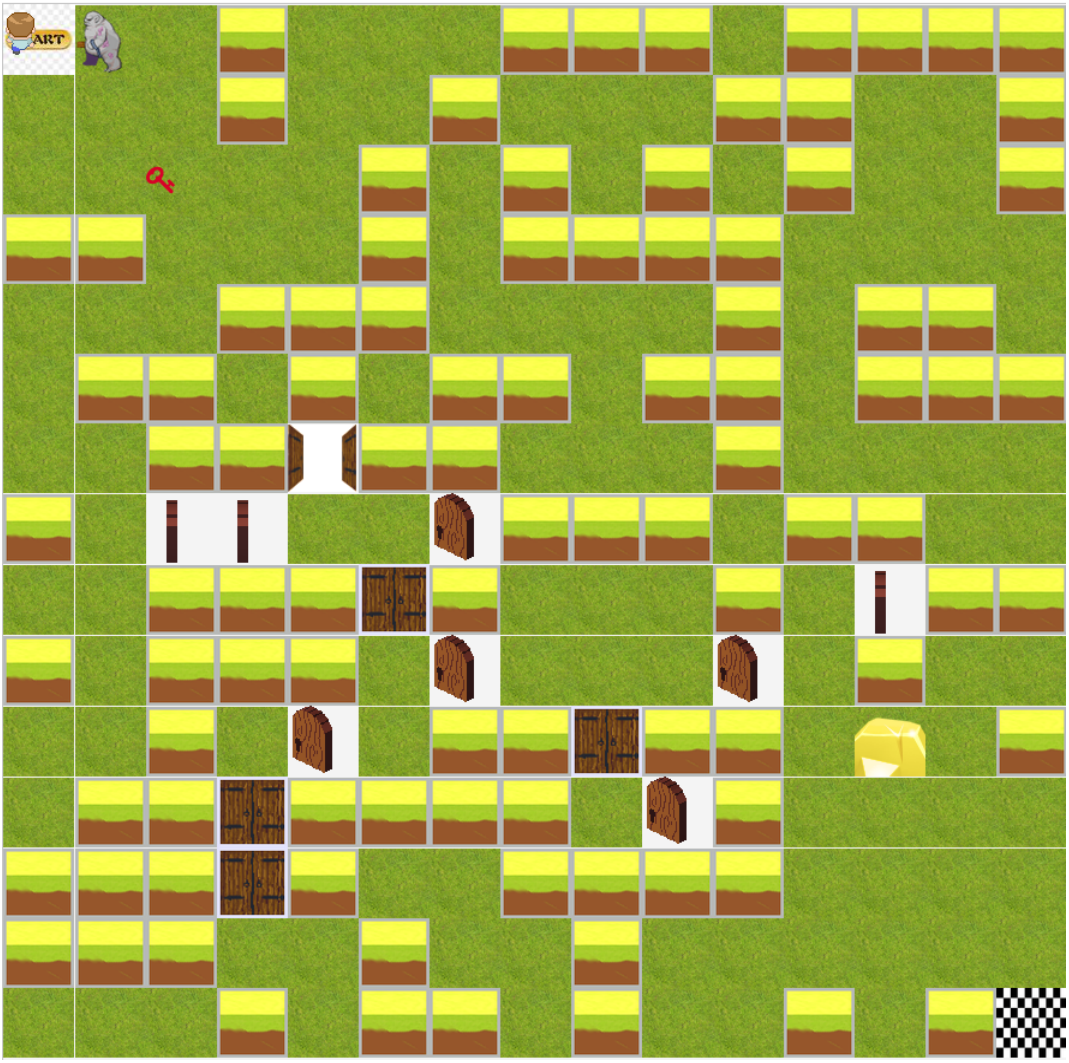
Afin de lancer le jeux sur vos machines, un build.xml a été intégré. Trois commande **ANT** sont ncéssaires pour lancer l'interface principale. (les commandes sont à exécuter à la racine du projet ou se trouve le build.xml)

- **ant clean**
- **ant build**
- **ant run**

Si vous souhaitez lancer l'édition de map utiliser la commande suivant. - **ant MainEditMap**

Pour ce qui est des test utilisez la suite de commande suivante

- **ant clean**
- **ant build**
- **ant RunTests** permet de lancer tous les tests.
- **ant junitreport** crée un fichier all-test.html dans le dossier junit. En ouvrant ce fichier sur un navigateur vous avez les résultats des tests.


START