Spécifications projet Warcraft

service : Villageois
types: int, double, boolean, enum ERace {ORC, HUMAN}
observators :

const race : *[Villageois] → ERace*
const largeur : *[Villageois] → int*
const hauteur : *[Villageois] → int*
const force : *[Villageois] → int*
const vitesse : *[Villageois] → double*
posx: *[Villageois] → int*
posy: *[Villageois] → int*
pointsDeVie : *[Villageois] → int*
quantiteOr : *[Villageois] → int*
estMort : *[Villageois] → boolean*
corvee: *[Villageois] → int*
estOccupe: *[Villageois] → boolean*

Constructors :

init : *int x int x ERace × int × int × int × double × int → [Villageois]*
pre init(x,y, race,largeur,hauteur,force,vitesse,pointsVie)
require largeur % 2=1 ∧
hauteur % 2=1 ∧
force > 0 ∧
vitesse > 0 ∧
pointsVie > 0 ∧
x ≥ 0 ∧
y ≥ 0

Operators :

setXY: *[Villageois] × int → [Villageois]*
pre setX(V, x) **require**
x ≥ 0 ∧
y ≥ 0
retrait : *[Villageois] × int → [Villageois]*
pre retrait(V, s) **require**
¬estMort(V) ∧
¬estOccupe(V) ∧ s > 0
chargeOr : *[Villageois] × int → [Villageois]*
pre chargeOr(V, s) **require**
¬estMort(V) ∧
s > 0
dechargeOr : *[Villageois] × int → [Villageois]*
pre dechargeOr(V, s) **require**
¬estMort(V) ∧ s > 0 ∧
s ≤ quantiteOr(V)
setCorvee: *[Villageois] × int × int × int → [Villageois]*
pre setCorvee(V, s, corveeX, corveeY) **require**
¬estMort(V) ∧
¬estOccupe(V) ∧
s > 0 ∧
corveeX ≥ 0 ∧
corveeY ≥ 0
decrCorvee: *[Villageois] → [Villageois]*
pre decrCorvee(V) **require**
¬estMort(V) ∧
estOccupe(V)

Observations :

[invariants]

estMort(V) min = pointsDeVie(V) ≤ 0
estOccupe(V) $\stackrel{m}{=}$ corvee > 0

[init]
posx(init(x,y,s,l,h,f,v,p))=x
posy(init(x,y,s,l,h,f,v,p))=y
race(init(x,y,s,l,h,f,v,p))=s
largeur(init(x,y,s,l,h,f,v,p))=l
hauteur(init(x,y,s,l,h,f,v,p))=h
force(init(x,y,s,l,h,f,v,p))=f
vitesse(init(x,y,s,l,h,f,v,p))=v
pointsDeVie(init(x,y,s,l,h,f,v,p))=p
quantiteOr(init(x,y,s,l,h,f,v,p))=0
corvee(init(x,y,s,l,h,f,v,p)) = 0

[retrait]
pointsDeVie(retrait(V,s)) = pointsDeVie(V) - s
quantiteOr(retrait(V,s)) = quantiteOr(V)
corvee(retrait(V,s)) = corvee(V)
posx(retrait(V,s)) = posx(V)
posy(retrait(V,s)) = posy(V)

[chargeOr]
pointsDeVie(chargeOr(V,s)) = pointsDeVie(V)
quantiteOr(chargeOr(V,s)) = quantiteOr(V) + s
corvee(chargeOr(V,s)) = corvee(V)
posx(chargeOr(V,s)) = posx(V)
posy(chargeOr(V,s)) = posy(V)

[dechargeOr]
pointsDeVie(dechargeOr(V,s)) = pointsDeVie(V)
quantiteOr(dechargeOr(V,s)) = quantiteOr(V) - s
corvee(dechargeOr(V,s)) = corvee(V)
posx(dechargeOr(V,s)) = posx(V)
posy(dechargeOr(V,s)) = posy(V)

[setCorvee]
pointsDeVie(setCorvee(V, s, x, y)) = pointsDeVie(V)
quantiteOr(setCorvee(V, s, x, y)) = quantiteOr(V)
corvee(setCorvee(V, s, x, y)) = s
posx(setCorvee(V, s, x, y)) = x
posy(setCorvee(V, s, x, y)) = y

[decrCorvee]
pointsDeVie(decrCorvee(V)) = pointsDeVie(V)
quantiteOr(decrCorvee(V)) = quantiteOr(V)
corvee(decrCorvee(V)) = corvee(V) - 1
posx(decrCorvee(V)) = posx(V)
posy(decrCorvee(V)) = posy(V)

[setXY]
pointsDeVie(setX(V, x,y)) = pointsDeVie(V)
quantiteOr(setX(V,x,y)) = quantiteOr(V)
corvee(setX(V, x,y)) = corvee(V)
posx(setX(V, x,y)) = x
posy(setX(V, x,y)) = y

service: Terrain
types: int, enum EEntite {MURAILLE, ROUTE, VILLAGEOIS, MINE, HDV, RIEN}, List<T>

observators:

const largeur : *[Terrain] → int*
const hauteur *: [Terrain] → int*
const getListeMuraille: *[Terrain] → List<Muraille>*
const getListeRoute: *[Terrain] → List<Route>*
const getListeVillageois : *[Terrain] → List<Villageois>*
const getListeMine : *[Terrain] → List<Mine>*
const getListeHotelVille : *[Terrain] → List<HôtelVille>*

estFranchissable: *[Terrain] × int × int × int × int → boolean*
pre estFranchissable(T, x, y, l, h) **require**
$x \geq 0 \land$
$y \geq 0 \land$
$x + l \leq$ largeur $\land$
$y + h \leq$ hauteur
getEntiteAt: *[Terrain] × int × int → Set<EEntite>*
pre getEntiteAt(T, x, y) **require**
$0 \leq x \leq$ largeur $\land$
$0 \leq y \leq$ hauteur
getBonusVitesse: *[Terrain] × int × int × int × int → int*
pre getBonusVitesse(T, x, y, l, h) **require**
$x \geq 0 \land$
$y \geq 0 \land$
$x + l \leq$ largeur $\land$
$y + h \leq$ hauteur
getRouteAt: *[Terrain] × int × int × int × int → [Route]*
pre getRouteAt(T, x, y, l, h) **require**
$x \geq 0 \land$
$y \geq 0 \land$
$x + l \leq$ largeur $\land$
$y + h \leq$ hauteur $\land$
ROUTE $\in$ getEntiteAt(T, x, y)

constructors:

init: int × int
pre init(largeur, hauteur) **require**
largeur $\geq 600 \land$
hauteur ≥ 400

operators:

setEntiteAt: *[Terrain] × EEntite × int × int × int × int → [Terrain]*
pre setEntiteAt(T, Ent, x, y, l, h) **require**
estFranchissable(T, x, y, l, h)
removeEntiteAt: *[Terrain] × EEntite × int × int × int × int → [Terrain]*
pre removeEntiteAt(T, Ent, x, y, l, h) **require**
$x \geq 0 \land$
$y \geq 0 \land$
$x + l \leq$ largeur $\land$
$y + h \leq$ hauteur $\land$
$\forall$ j $\in$ [x, x + l[, $\forall$ k $\in$ [y, y + h[
Ent $\in$ getEntiteAt(T, j, k)
reinsertVillageois: *[Terrain] × int → [Terrain]*
pre reinsertVillageois(T, numVil) **require**
$0 \leq$ numVil < |getListeVillageois(T)| $\land$
MINE $\in$ getEntiteAt(T, Villageois:: posx(get(getListeVillageois(T), numVil)),
Villageois:: posy(get(getListeVillageois(T), numVil))

observations:

[invariants]

getRouteAt(T, x, y, l, h) $\stackrel{m}{=}$ ro | ro ∈ getListeRoute(T) ∧
Route::posx(ro) ∈ [x, x + l[∧
Route::posy(ro) ∈ [y, y + h[

estFranchissable(T, x, y, l, h) $\stackrel{m}{=}$ ∀ i ∈ [x, x + l[, ∀ j ∈ [y, y + h[|
getEntiteAt(T, i, j) = {RIEN} ∨ getEntiteAt(T, i, j) = {ROUTE}

getBonusVitesse(T, x, y, l, h) $\stackrel{m}{=}$

- Route::bonusVitesse(getRouteAt(T, j, k)) si ∃ j ∈ [x, x + l[, ∃ k ∈ [y, y + h[| {ROUTE} = getEntiteAt(T, j, k)
- 0 sinon

∀ x ∈ [0, largeur[, ∀ y ∈ [0, hauteur[, |getEntiteAt(T, x, y)| > 0 ∧ (RIEN ∈ getEntiteAt(T, x, y) ⇒ |getEntiteAt(T, x, y)| = 1)

[init]

largeur(init(l,h)) = l
hauteur(init(l,h)) = h
∀ HV ∈ getListeHotelVille(init(l,h)) |
HotelVille::posx(HV) + HotelVille::largeur(HV) ≤ l ∧
HotelVille::posy(HV) + HotelVille::hauteur(HV) ≤ h ∧
HotelVille::orRestant(HV) = 16 ∧
∀ x ∈ [HotelVille::posx(HV), HotelVille::posx(HV) + HotelVille::largeur(HV)[,
∀ y ∈ [HotelVille::posy(HV), HotelVille::posy(HV) + HotelVille::hauteur(HV)[:
HDV ∈ getEntiteAt(init(l, h), x, y)

∀ Vill ∈ getListeVillageois(init(l, h)), soit HDV def= hdv ∈ getListeHotelVille(init(l, h)) | HotelVille::etatAppartenance(hdv) = Villageois::race(Vill), soit vilX def= Villageois::posx(Vill), soit vilY def= Villageois::posy(Vill),
vilX + Villageois::largeur(Vill) ≤ l ∧
vilY + Villageois::hauteur(Vill) ≤ h ∧
distance(vilX, vilY, HotelVille::posx(HDV), HotelVille::posy(HDV)) ≤ 51 ∧
Villageois::pointsDeVie(Vill) = 100 ∧
∀ x ∈ [vilX, vilX + Villageois::largeur(Vill)[,
∀ y ∈ [vilY, vilY + Villageois::hauteur(Vill)[,
VILLAGEOIS ∈ getEntiteAt(init(l, h), x, y)

∀ Mi ∈ getListeMine(init(l, h)),
Mine::posx(Mi) + Mine::largeur(Mi) ≤ l ∧
Mine::posy(Mi) + Mine::hauteur(Mi) ≤ h ∧
∀ x ∈ [Mine::posx(Mi), Mine::posx(Mi) + Mine::largeur(Mi)[,
∀ y ∈ [Mine::posy(Mi), Mine::posy(Mi) + Mine::hauteur(Mi)[,
MINE ∈ getEntiteAt(init(l, h), x, y)

∀ R ∈ getListeRoute(init(l, h)),
Route::posx(R) + Route::largeur(R) ≤ l ∧
Route::posy(R) + Route::hauteur(R) ≤ h ∧
∀ x ∈ [Route::posx(R), Route::posx(R) + Route::largeur(R)[,
∀ y ∈ [Route::posy(R), Route::posy(R) + Route::hauteur(R)[,
ROUTE ∈ getEntiteAt(init(l, h), x, y)

∀ Mu ∈ getListeMuraille(init(l, h)),
Muraille::posx(Mu) + Muraille::largeur(Mu) ≤ l ∧
Muraille::posy(Mu) + Muraille::hauteur(Mu) ≤ h ∧
∀ x ∈ [Muraille::posx(Mu), Muraille::posx(Mu) + Muraille::largeur(Mu)[,
∀ y ∈ [Muraille::posy(Mu), Muraille::posy(Mu) + Muraille::hauteur(Mu)[,
MURAILLE ∈ getEntiteAt(init(l, h), x, y)

[setEntiteAt]

∀ j ∈ [x, x + l[, ∀ k ∈ [y, y + h[
getEntiteAt(setEntiteAt(T, Ent, x, y, l, h), j, k) =

- {Ent} si getEntiteAt(T, j, k) = {RIEN}
- {Ent} ∪ getEntiteAt(T, j, k) sinon

[removeEntiteAt]

∀ j ∈ [x, x + l[, ∀ k ∈ [y, y + h[,
getEntiteAt(removeEntiteAt(T, Ent, x, y, l, h), j, k) =

- {RIEN} si |getEntiteAt(T, j, k)| = 1
- getEntiteAt(T, j, k) \ {Ent} sinon

[reinsertVillageois]

soit Tpost ≝ reinsertVillageois(T, numVil), soit Vill def= get(getListeVillageois(Tpost), numVil)
∀ x ∈ [Villageois::posx(Vill), Villageois::posx(Vill) + Villageois::largeur(Vill)[,
∀ y ∈ [Villageois::posy(Vill), Villageois::posy(Vill) + Villageois::hauteur(Vill)[,
VILLAGEOIS ∈ getEntiteAt(Tpost, x, y)

service: MoteurJeu
types: enum RESULTAT{ORC_GAGNE, HUMAN_GAGNE, NUL},
enum COMMANDE{RIEN, DEPLACER, ENTRERMINE, ENTRERHOTELVILLE, TAPERMURAILLE}, int, boolean, Terrain
observators:

const *maxPasJeu : [MoteurJeu] → int*
const *gestDepl: [MoteurJeu] → GestionDeplacement*
terrain : *[MoteurJeu] → Terrain*
pasJeuCourant : *[MoteurJeu] → int*
estFini : *[MoteurJeu] → boolean*
resultatFinal : *[MoteurJeu] → RESULTAT*
pre resultatFinal(M) **require** estFini(M)

peutEntrer : *[MoteurJeu] × int × int → boolean*
pre peutEntrer(M, vilNum, minNum) **require**
¬Villageois::estMort(getVillageois(M, vilNum)) ∧
0 ≤ minNum < |Terrain::getListeMine(terrain(M))|
peutEntrerHotelVille : *[MoteurJeu] × int × int → boolean*
pre peutEntrerHotelVille(M, vilNum, hdv) **require**
¬Villageois::estMort(getVillageois(M, vilNum)) ∧
0 ≤ hdv < |Terrain::getListeHotelVille(terrain(M))|
peutTaperMuraille: *[MoteurJeu] × int × int → boolean*
pre peutTaperMuraille(M, vilNum, mur) **require**
¬Villageois::estMort(getVillageois(M, vilNum)) ∧
¬Muraille::estDetruite(getMuraille(M, mur))
getVillageois : *[MoteurJeu] × int → Villageois*
pre getVillageois(M, vill) **require**
0 ≤ vill < |Terrain::getListeVillageois(terrain(M))|
getMine : *[MoteurJeu] × int → Mine*
pre getMine(M, mi) **require**
0 ≤ mi < |Terrain::getListeMine(terrain(M))|
getMuraille : *[MoteurJeu] × int → Muraille*
pre getMuraille(M, mi) **require**
0 ≤ mi < |Terrain::getListeMuraille(terrain(M))|
getHDV : *[MoteurJeu] × int → HotelVille*
pre getHDV(M, hdv) **require**
0 ≤ hdv < |Terrain::getListeHotelVille(terrain(M))|

constructors:

init : *int → [MoteurJeu]*
pre init(maxPas) **require** maxPas > 0

operators:

pasJeu : *[MoteurJeu] × COMMANDE × int × int → [MoteurJeu]*
pre pasJeu(M, command, vilNum, arg) **require** ¬estFini(M) ∧
- command ≠ RIEN ⇒
  - ¬Villageois::estMort(getVillageois(M, vilNum)) ∧
  - ¬Villageois::estOccupe(getVillageois(M, vilNum))
- command = DEPLACER ⇒
  - 0 ≤ arg ≤ 360
- command = ENTRERMINE ⇒
  - peutEntrer(M, vilNum, arg)
- command = ENTRERHOTELVILLE ⇒
  - peutEntrerHotelVille(M, vilNum, arg)
- command = TAPERMURAILLE ⇒
  - ¬ Muraille::estDetruite(getMuraille(M, arg)) ∧
  - peutTaperMuraille(M, vilNum, arg)

observations:
[invariants]
0 ≤ pasJeuCourant(M) ≤ maxPasJeu(M)
estFini(M) $\stackrel{m}{=}$ ∃ x | 0 ≤ x ≤ |Terrain::getListeHotelVille(terrain(M))| ∧

HotelVille::orRestant(getHDV(M, x)) ≥ 1664 ∨
pasJeuCourant(M) = maxPasJeu(M)

resultatFinal(M) ≝

- ORC_GAGNE si ∃ x | 0 ≤ x ≤ |Terrain::getListeHotelVille(terrain(M))| ∧
  HotelVille::orRestant(getHDV(M, x)) ≥ 1664 ∧
  HotelVille::etatAppartenance(getHDV(M, x)) = ORC
- HUMAN_GAGNE si ∃ x | 0 ≤ x ≤ |Terrain::getListeHotelVille(terrain(M))| ∧
  HotelVille::orRestant(getHDV(M, x)) ≥ 1664 ∧
  HotelVille::etatAppartenance(getHDV(M, x)) = HUMAN
- NUL sinon

getVillageois(M, vill) ≝ get(Terrain::getListeVillageois(terrain(M)), vill)
getMine(M, mi) ≝ get(Terrain::getListeMine(terrain(M)), mi)
getMuraille(M, mu) ≝ get(Terrain::getListeMuraille(terrain(M)), mu)
getHDV(M, hdv) ≝ get(Terrain::getListeHotelVille(terrain(M)), hdv)

Soit vilposX ≝ Villageois::posx(getVillageois(M, vill)),
Soit vilposY ≝ Villageois::posy(getVillageois(M, vill)),
Soit mineCenterX ≝ (Mine::posx(getMine(M, mi)) +(Mine::largeur(getMine(M, mi)) / 2),
Soit mineCenterY ≝ (Mine::posy(getMine(M, mi)) + (Mine::hauteur(getMine(M, mi))) / 2),
Soit hvCenterX ≝ (HotelVille::posx(getHDV(M, hdv)) + (HotelVille::largeur(getHDV(M, hdv)) / 2),
Soit hvCenterY ≝ (HotelVille::posy(getHDV(M, hdv)) + (HotelVille::hauteur(getHDV(M, hdv)) / 2),
Soit murCenterX ≝ (Muraille::posx(getMuraille(M, mu)) +(Muraille::largeur(getMuraille(M, mu)) / 2) ,
Soit murCenterY ≝ (Muraille::posy(getMuraille(M, mu)) + (Muraille::hauteur(getMuraille(M, mu)) / 2),

peutEntrer(M, vill, mi) ≝ distance(vilposX, vilposY, mineCenterX, mineCenterY) ≤ 51 ∧
¬Mine::estLaminee(getMine(M, mi))
peutEntrerHotelVille(M, vill, hdv) ≝ distance(vilposX, vilposY, hvCenterX, hvCenterY)) ≤ 51 ∧
Villageois::race(getVillageois(M, vill)) = HotelVille::etatAppartenance(getHDV(M, hdv)) ∧
Villageois::quantiteOr(getVillageois(M, vill)) > 0
peutTaperMuraille(M, vill,mu) ≝ distance(vilposX, vilposY, murCenterX, murCenterY) ≤ 51

[init]

maxPasJeu(init(m)) = m
pasJeuCourant(init(m)) = 0

[pasJeu]

Soit Mpj ≝ pasJeu(M,c,vilNum,arg)
pasJeuCourant(Mpj) = pasJeuCourant(M) + 1

∀ x ∈ [0, |Terrain::getListeVillageois(terrain(M))|[, soit Vill ≝ getVillageois(M, x),
(Villageois::estOccupe(Vill) ⇒ getVillageois(Mpj, x) = Villageois::decrCorvee(Vill)) ∧
(Villageois::corvee(Vill) = 1 ⇒ terrain(Mpj) = Terrain::reinsertVillageois(terrain(M), x))

∀ x ∈ [0, |Terrain::getListeMine(terrain(M))|[, soit Mi def= getMine(M, x),
(¬ Mine::estAbandonee(Mi) ∧
(¬ c = ENTRERMINE ∨ x != arg)) ⇒ getMine(Mpj, x) = Mine::abandoned(Mi)

Soit Villpre ≝ getVillageois(M, vilNum)
Soit Villpost ≝ getVillageois(Mpj, vilNum)
Soit pArrivee ≝ GestionDeplacement::getPointArrivee(GestionDeplacement::calcChemin(gestDepl(M), vilNum, arg)))
c = DEPLACER ⇒
Villpost = Villageois::setXY(Villpre, get(pArrivee, 0), get(pArrivee, 1)) ∧
terrain(Mpj) = Terrain::setEntiteAt(Terrain::removeEntiteAt(terrain(M), VILLAGEOIS,
Villageois::posx(Villpre), Villageois::posy(Villpre), Villageois::largeur(Villpre), Villageois::hauteur(Villpre)),
VILLAGEOIS, get(pArrivee, 0), get(pArrivee, 1), Villageois::largeur(Villpost), Villageois::hauteur(Villpost))

c = ENTRERHOTELVILLE ⇒
getHDV(Mpj, arg) = HotelVille::depot(getHDV(M, arg), Villageois::quantiteOr(Villpre)) ∧
Villpost = Villageois::dechargeOr(Villpre, Villageois::quantiteOr(Villpre))

c = ENTRERMINE ⇒

getMine(Mpj, arg) = Mine::retrait(Mine::accueil(getMine(M, arg), Villageois::race(Villpre)), 1) ∧
Villpost = Villageois::chargerOr(Villageois::setCorvee(Villpre, 16), 1) ∧
terrain(Mpj) = Terrain::removeEntiteAt(terrain(M), VILLAGEOIS, Villageois::posx(Villpre),
Villageois::posy(Villpre), Villageois::largeur(Villpre), Villageois::hauteur(Villpre)) ∧
Villageois::posx(Villpost) = Mine::posx(getMine(M, arg)) ∧
Villageois::posy(Villpost) = Mine::posy(getMine(M, arg))

c = TAPERMURAILLE ⇒

Soit Mur def= getMuraille(M, arg)

getMuraille(Mpj, arg) = Muraille::retrait(Mur, Villageois::force(Villpre)) ∧
Muraille::estDetruite(getMuraille(Mpj, arg)) ⇒
terrain(Mpj) = Terrain::removeEntiteAt(terrain(M),
MURAILLE,
Muraille::posx(Mur),
Muraille::posy(Mur),
Muraille::largeur(Mur),
Muraille::hauteur(Mur))

service : GestionDeplacement
types: List<int>, Terrain, Villageois, boolean, int
observators:

const terr: *[GestionDeplacement]* → *Terrain*
estCalcChemin: *[GestionDeplacement]* → *boolean*
cheminX: *[GestionDeplacement]* → *List<int>*
pre cheminX(GD) **require** estCalcChemin(GD)
cheminY: *[GestionDeplacement]* → *List<int>*
pre cheminY(GD) **require** estCalcChemin(GD)
getPointArrivee: *[GestionDeplacement]* → *List<int>*
pre getPointArrivee(GD) **require** estCalcChemin(GD)
firstObstacle: *[GestionDeplacement]* → *int*
pre firstObstacle(GD) **require** estCalcChemin(GD)

constructors:

init: → *[GestionDeplacement]*

operators:

calcChemin: *[GestionDeplacement]* × *int* × *int* → *[GestionDeplacement]*
pre calcChemin(GD, vilNum, angle) **require**
$0 \leq angle \leq 360 \land$
$vilNum \in [0, |Terrain::getListeVillageois(terr(GD))|[$

observations:

[init]
estCalcChemin(init()) = false

[calcChemin]
estCalcChemin(calcChemin(GD, vilNum, angle)) = true
Soit chemPost def= calcChemin(GD, vilNum, angle)
Soit Vill ≝ get(Terrain::getListeVillageois(terr(chemPost)), vilNum)
Soit bonus ≝ ∑ i \from 0 \to |cheminX(chemPost)| - 1,
Terrain::getBonusVitesse(terr(chemPost), get(cheminX(chemPost), i), get(cheminY(chemPost), i))
|cheminX(chemPost)| = |cheminY(chemPost)|
$|cheminX(chemPost)| \leq$ bonus + Villageois::vitesse(Vill)|
firstObstacle(calcChemin(chemPost)) =

- 0 si |cheminX| = 0
- i si $\exists$ min i $\in$ {i | i $\in$ [0, |cheminX|[$\land$
  $\exists$ x $\in$ [get(cheminX(chemPost), i), get(cheminX(chemPost), i) + Villageois::largeur(Vill)[,
  $\exists$ y $\in$ [get(cheminY(chemPost), i), get(cheminY(chemPost), i) + Villageois::hauteur(Vill)[,
  ¬Terrain::estFranchissable(terr(chemPost), x, y)}
- -1 sinon

getPointArrivee(chemPost) =

- {get(cheminX(chemPost), |cheminX(chemPost)| - 1), get(cheminY(chemPost), |cheminY(chemPost)| - 1)} si firstObstacle(chemPost) = -1
- {Villageois::posx(Vill), Villageois::posy(Vill)} si firstObstacle(chemPost) = 0
- {get(cheminX(chemPost), firstObstacle(chemPost) - 1), get(cheminY(chemPost), firstObstacle(chemPost) - 1)} sinon

service : Mine
types: int, boolean, enum ERace {ORC, HUMAN}
observators :

const posx : [Mine] → int
const posy : [Mine] → int
const largeur : [Mine] → int
const hauteur : [Mine] → int
orRestant : [Mine] → int
estAbandonnee : [Mine] → boolean
estLaminee : [Mine] → boolean
abandonCompteur : [Mine] → int
etatAppartenance : [Mine] → ERace
pre etatAppartenance(M) **require** ¬estAbandonnee()

constructors :

init : *int* × *int* × *int* × *int* → *[Mine]*
pre init(x, y, largeur, hauteur) **require**
largeur % 2 = 1 ∧
hauteur % 2 = 1 ∧
x ≥ 0 ∧
y ≥ 0 ∧

operators :

retrait : [Mine] × int → [Mine]
pre retrait(M,s) **require** ¬estAbandonnee(M) ∧ 0 < s ≤ orRestant(M)
acceuil : [Mine] × ERace → [Mine]
pre acceuil(M, r) **require** estAbandonnee(M) v etatAppartenance(M) = r
abandoned : [Mine] → [Mine]
pre abandoned(M) **require** ¬estAbandonnee(M)

observations :

[invariants]

estLaminee(M) min = orRestant(M) ≤ 0
estAbandonnee(M) min = abandonCompteur = 51
0 ≤ abandonCompteur(M) ≤ 51

[init]

posx(init(x, y, l, h)) = x
posy(init(x, y, l, h)) = y
largeur(init(x, y, l, h)) = l
hauteur(init(x, y, l, h)) = h
orRestant(init(x, y, l, h)) = 51
abandonCompteur(init(x, y, l, h)) = 51
etatAppartenance(init(x, y, l, h)) = ORC

[retrait]

orRestant(retrait(M,s)) = orRestant(M) - s
abandonCompteur(retrait(M,s)) = abandonCompteur(M)
etatAppartenance(retrait(M,s)) = etatAppartenance(M)

[acceuil]

orRestant(acceuil(M, r)) = orRestant(M)
abandonCompteur(accueil(M, r)) = 0
etatAppartenance(accueil(M, r)) = r

[abandoned]

orRestant(abandoned(M)) = orRestant(M)
abandonCompteur(abandoned(M)) = abandonCompteur() + 1
etatAppartenance(abandoned(M)) = etatAppartenance(M)

service : HotelVille
observators :

const etatAppartenance : [HotelVille] → ERace
const largeur : [HotelVille] → int
const hauteur : [HotelVille] → int
const posx : [HotelVille] -> int
const posy : [HotelVille] -> int
orRestant : [HotelVille] → int

Constructors :

init : *int* × *int* × *int* × *int* × *int* × *ERace* → *[HotelVille]*
pre init(x, y, largeur, hauteur, or, appr) **require**
largeur % 2 = 1 ∧
hauteur % 2 = 1 ∧
x ≥ 0 ∧
y ≥ 0 ∧
or ≥ 0

Operators :

depot : [HotelVille] × int → [HotelVille]
pre depot(H,s) **require** s > 0

Observations :

[init]

posx(init(x, y, l, h, or, appr))=x
posy(init(x, y, l, h, or, appr))=y
largeur(init(x, y, l, h, or, appr))=l
hauteur(init(x, y, l, h, or, appr))=h
orRestant(init(x, y, l, h, or, appr))=or
etatAppartenence(init(x, y, l, h, or, appr)) = appr

[depot]

orRestant(depot(H, s)) = orRestant(H) + s

service : Route
observators :
const largeur : [Route] → int
const hauteur : [Route] → int
const posx : [Route] → int
const posy : [Route] → int
const bonusVitesse : [Route] → int

Constructors :
init : int x int x int × int → [Route]
pre init(x, y, largeur, hauteur, bv) **require**
largeur % 2 = 1 ∧
hauteur % 2 = 1 ∧
x ≥ 0 ∧
y ≥ 0 ∧
bv > 0

Operators :
//

Observations :
[init]
posx(init(x, y, l, h, bv)) = x
posy(init(x, y, l, h, bv)) = y
largeur(init(x, y, l, h, bv)) = l
hauteur(init(x, y, l, h, bv)) = h
bonusVitesse(init(x, y, l, h, bv)) = bv

service : *Muraille*
observators :

const largeur : [Muraille] → int
const hauteur : [Muraille] → int
const posx : [Muraille] → int
const posy : [Muraille] → int
pointsDeVie : [Muraille] → int
estDetruite : [Muraille] → boolean

Constructors :

init : int x int x int × int → [Muraille]
pre init(x,y, largeur,hauteur, pv) **require**
largeur % 2=1 ∧
hauteur % 2=1 ∧
x ≥ 0 ∧
y ≥ 0 ∧
pv > 0

Operators :

retrait : [Muraille] × int → [Muraille]
pre *retrait(M,s)* ***require*** *¬estDetruite(M) ∧ s>0*

Observations :

[invariants]
estDetruite(M) min = pointsDeVie(M) ≤ 0

[init]
posx(init(x,y,l,h,pv))=x
posy(init(x,y,l,h,pv))=y
largeur(init(x,y,l,h,pv))=l
hauteur(init(x,y,l,h,pv))=h
pointsDeVie(init(x,y,l,h,pv))=pv

[retrait]
pointsDeVie(retrait(M,s))=pointsDeVie(M)-s