Bloc

Service: Bloc

Types: boolean, enum TypeBloc { VIDE, TERRE, MUR, HERO, SORTIE_FERMEE, SORTIE_OUVERTE, ROCHER, DIAMANT }

Use: Position

Observators:

getType: [Bloc] → TypeBloc
getPosition: [Bloc] → Position
isVide: [Bloc] → boolean
isSolide: [Bloc] → boolean
isDeplacable: [Bloc] → boolean
isTombable: [Bloc] → boolean
isSortie: [Bloc] → boolean
isSortieFermee: [Bloc] → boolean
isHero: [Bloc] → boolean
isTerre: [Bloc] → boolean

Constructors:

init: TypeBloc × Position → [Bloc]

Operators:

setType: [Bloc] × TypeBloc → [Bloc]

Observations:

[invariant]

isVide(b) $\overset{min}{=}$ getType(b) = VIDE
isSolide(b) $\overset{min}{=}$ getType(b) \in { SORTIE_FERMEE, MUR, ROCHER }
isDeplacable(b) $\overset{min}{=}$ getType(b) = ROCHER
isTombable(b) $\overset{min}{=}$ getType(b) \in { ROCHER, DIAMANT }
isSortie(b) $\overset{min}{=}$ getType(b) \in { SORTIE_FERMEE, SORTIE_OUVERTE }
isSortieFermee(b) $\overset{min}{=}$ getType(b) = SORTIE_FERMEE
isHero(b) $\overset{min}{=}$ getType(b) = HERO
isTerre(b) $\overset{min}{=}$ getType(b) = TERRE

[init]

getType(init(tb, pos)) = tb
getPosition(init(tb, pos)) = pos

[setType]

getType(setType(b, tb)) = tb
getPosition(setType(b, tb)) = getPosition(b)

Position

Service: Position

Types: integer, enum Direction { HAUT, BAS, GAUCHE, DROITE }

Observators:

```
const getLargeur: [Position] → integer
const getHauteur: [Position] → integer
getX: [Position] → integer
getY: [Position] → integer
```

Constructors:

```
init: integer × integer × integer × integer → [Position]
    pre init(l, h, x, y) require (l > 0) ∧ (h > 0) ∧ (x ≥ 0) ∧ (y ≥ 0)
```

Operators:

```
deplacerVersDirection: [Position] × Direction → [Position]
```

Observations:

```
[init]
    getLargeur(init(l, h, x, y)) = l
    getHauteur(init(l, h, x, y)) = h
    getX(init(l, h, x, y)) = x % l
    getY(init(l, h, x, y)) = y % h

[deplacerVersDirection]
    getX(deplacerVersDirection(p, dir)) =
        if dir = GAUCHE then
            (getX(p) - 1) % getLargeur(p)
        else if dir = DROITE then
            (getX(p) + 1) % getLargeur(p)
        else
            getX(p)

    getY(deplacerVersDirection(p, dir)) =
        if dir = HAUT then
            (getY(p) - 1) % getHauteur(p)
        else if dir = BAS then
            (getY(p) + 1) % getHauteur(p)
        else
            getY(p)
```

Terrain

Service: Terrain

Types: integer, boolean, **Set**, enum Direction { HAUT, BAS, GAUCHE, DROITE }, enum TypeBloc { VIDE, TERRE, MUR, HERO, SORTIE_FERMEE, SORTIE_OUVERTE, ROCHER, DIAMANT }

Use: Bloc, Position

Observators:

```
const getLargeur: [Terrain] → integer
const getHauteur: [Terrain] → integer
getPosSortie: [Terrain] → Position
getPosHero: [Terrain] → Position
getBlocHero: [Terrain] → Bloc
    pre getBlocHero(t) require isHeroVivant(t)
getBlocDepuisPosition: [Terrain] × Position → Bloc
getBloc: [Terrain] × integer × integer → Bloc
getBlocVersDirection: [Terrain] × Bloc × Direction → Bloc
getBlocs: [Terrain] → Set<Bloc>
isHeroVivant: [Terrain] → boolean
isDiamantsRestants: [Terrain] → boolean
isDeplacementBlocPossible: [Terrain] × Bloc × Direction → boolean
```

Constructors:

```
init: integer × integer → [Terrain]
    pre init(l, h) require l > 0 ∧ h > 0
```

Operators:

```
setBloc: [Terrain] × TypeBloc × integer × integer → [Terrain]
deplacerBlocVersDirection: [Terrain] × Bloc × Direction → [Terrain]
    pre deplacerBlocVersDirection(t, bloc, dir)
    require isDeplacementBlocPossible(t, bloc, dir)
fairePasDeMiseAJour: [Terrain] → [Terrain]
```

Observations:

[invariants]

```
getBlocHero(t) =^min getBlocDepuisPosition(getPosHero(t))

∀bloc ∈ getBlocs(t), dir ∈ Direction, getBlocVersDirection(t, bloc, dir) =^min
    getBloc(t, Position::deplacerVersDirection(Bloc::getPosition(bloc), dir))

isHeroVivant(t) =^min ∃bloc ∈ getBlocs(t), Bloc::getType(bloc) = HERO

isDiamantsRestants(t) =^min ∃bloc ∈ getBlocs(t), Bloc::getType(bloc) = DIAMANT
```

```
    ∀bloc ∈ getBlocs(t), dir ∈ Direction, isDeplacementBlocPossible(t, bloc, dir) =min=
        let blocDest = getBlocVersDirection(t, bloc, dir)
        in
            (Bloc::isHero(bloc) ∧ Bloc::isTerre(blocDest))
            ∨ ¬Bloc::isSolide(getBlocVersDirection(t, bloc, dir))

    getBlocDepuisPosition(t, pos) =min=
        getBloc(t, Position::getX(pos), Position::getY(pos))

    getBlocs(t) =min=
        ∑x ∈ [0..getLargeur() - 1], y ∈ [0..getHauteur() - 1], getBloc(t, x, y)

[init]
    getLargeur(init(l, h)) = l
    getHauteur(init(l, h)) = h
    getPosSortie(init(l, h)) = null
    getPosHero(init(l, h)) = null

    ∀x ∈ [0..getLargeur() - 1], y ∈ [0..getHauteur() - 1],
        let* bloc = getBloc(init(l, h), x, y)
        and blocPos = Bloc::getPosition(bloc)
        in Bloc::isVide(bloc) ∧ Position::getX(blocPos) = x
            ∧ Position::getY(blocPos) = y

[setBloc]
    getPosSortie(setBloc(t, type, x, y)) =
        if type \in { SORTIE_FERMEE, SORTIE_OUVERTE } then
            Bloc::getPosition(getBloc(t, x, y))
        else
            getPosSortie(t)

    getPosHero(setBloc(t, type, x, y)) =
        if type = HERO then
            Bloc::getPosition(getBloc(t, x, y))
        else
            getPosHero(t)

    ∀x' ∈ [0..getLargeur() - 1], y' ∈ [0..getHauteur() - 1],
        getBloc(setBloc(t, type, x, y), x', y') =
            if x = x' ∧ y = y' then
                Bloc::setType(getBloc(t, x, y), type)
            else
                getBloc(t, x', y')

[deplacerBlocVersDirection]
    getPosSortie(deplacerBlocVersDirection(t, bloc, dir)) = getPosSortie(t)

    getPosHero(deplacerBlocVersDirection(t, bloc, dir)) =
```

```
        if bloc = getBlocHero(t) then
            Bloc::getPosition(getBlocVersDirection(t, bloc, dir))
        else
            getPosHero(t)

    ∀x ∈ [0..getLargeur() - 1], y ∈ [0..getHauteur() - 1],
    getBloc(deplacerBlocVersDirection(t, bloc, dir), x, y) =
        let* blocPos = Bloc::getPosition(bloc)
        and blocX = Position::getX(blocPos)
        and blocY = Position::getY(blocPos)
        and blocDest = getBlocVersDirection(t, bloc, dir)
        and blocDestPos = Bloc::getPosition(blocDest)
        and blocDestX = Position::getX(blocDestPos)
        and blocDestY = Position::getY(blocDestPos)
        in
            if blocX = x ∧ blocY = y then
                Bloc::setType(bloc, VIDE)
            else if blocDestX = x ∧ blocY = y then
                Bloc::setType(blocDest, Bloc::getType(bloc))
            else
                getBloc(x, y)

[fairePasDeMiseAJour]
    getPosSortie(fairePasDeMiseAJour(t)) = getPosSortie(t)
    getPosHero(fairePasDeMiseAJour(t)) = getPosHero(t)

    ∀x ∈ [0..getLargeur() - 1], y ∈ [0..getHauteur() - 1],
    getBloc(fairePasDeMiseAJour(t), x, y) =
        let bloc = getBloc(t, x, y)
        in
            if Bloc::isSortieFermee(bloc) ∧ ¬isDiamantsRestants(t) then
                Bloc::setType(bloc, SORTIE_OUVERTE)
            else if Bloc::isTombable(bloc)
            ∧ Bloc::isVide(getBlocVersDirection(t, bloc, BAS))
                Bloc::setType(bloc, VIDE)
            else if Bloc::isVide(bloc)
            ∧ Bloc::isTombable(getBlocVersDirection(t, bloc, HAUT))
                Bloc::setType(bloc,
                    Bloc::getType(getBlocVersDirection(t, bloc, HAUT)))
            else
                getBloc(t, pos)
```

MoteurJeu

Service: MoteurJeu

Types: integer, boolean, enum Direction { HAUT, BAS, GAUCHE, DROITE }, enum TypeBloc { VIDE, TERRE, MUR, HERO, SORTIE_FERMEE, SORTIE_OUVERTE, ROCHER, DIAMANT }

Use: Terrain, Bloc, Position

Observators:

```
getTerrain: [MoteurJeu] → Terrain
getPasRestants: [MoteurJeu] → integer
isDeplacementHeroPossible: [MoteurJeu] × Direction → boolean
isPartieTerminee: [MoteurJeu] → boolean
isPartieGagnee: [MoteurJeu] → boolean
```

Constructors:

```
init: Terrain × integer → [MoteurJeu]
    pre init(t, nbPas) require nbPas > 0
```

Operators:

```
deplacerHero: [MoteurJeu] × Direction → [MoteurJeu]
    pre deplacerHero(mj, dir) require ¬isPartieTerminee(mj) ∧
        isDeplacementHeroPossible(mj, dir)
```

Observations:

[invariant]

```
isPartieTerminee(mj) =min
    getPasRestants(mj) = 0
    ∨ ¬Terrain::isHeroVivant(getTerrain(mj))
    ∨ Terrain::getPosSortie(getTerrain(mj)) = Terrain::getPosHero(getTerrain(mj))

isPartieGagnee(mj) =min isPartieTerminee(mj) ∧ Terrain::isHeroVivant(getTerrain(mj))

∀dir ∈ Direction, isDeplacementHeroPossible(mj, dir) =min
    let* terrain = getTerrain(mj)
    and blocHero = Terrain::getBlocHero(terrain)
    and blocDest = Terrain::getBlocVersDirection(terrain, blocHero, dir)
    in
        if dir \in { GAUCHE, DROITE } then
            ¬Bloc::isSolide(blocDest) ∨ (Bloc::isDeplacable(blocDest)
            ∧ Bloc::isVide(Terrain::getBlocVersDirection(terrain, blocDest, dir))
        else
            ¬Bloc::isSolide(blocDest)
```

[init]

```
    getPasRestants(init(t, nbPas)) = nbPas
    getTerrain(init(t, nbPas)) = t

[deplacerHero]
    getPasRestants(deplacerHero(mj, dir)) = getPasRestants(mj) - 1
    getTerrain(deplacerHero(mj, dir)) =
        let* terrain = getTerrain(mj)
        and blocHero = Terrain::getBlocHero(terrain)
        and blocDest = Terrain::getBlocVersDirection(terrain, blocHero, dir)
        in
            if ¬Bloc::isSolide(blocDest) then
                Terrain::deplacerBlocVersDirection(terrain, blocHero, dir)
            else if Bloc::isDeplacable(blocDest) and dir \in  GAUCHE, DROITE  then
                let terrain' = Terrain::deplacerBlocVersDirection(terrain, blocDest, dir)
                in Terrain::deplacerBlocVersDirection(terrain', blocHero, dir)
```

Légende

- observator
- operator/constructor
- External::observator
- External::operator