Specifications RiverRansomCity

Charles-Emmanuel Dias - Marwan Ghanem

April 18, 2014

1 Service Personnage

1.1 Spécification:

```
service: Personnage
use: Object
types : String,int,boolean
observators:
    const nom: [Personnage] -> String
    const width: [Personnage] -> int
    const height: [Personnage] -> int
    const depth: [Personnage] -> int
    const force: [Personnage] -> int
    currentForce: [Personnage] -> int
    hp:[Personnage] -> int
    money:[Personnage] -> int
    youDeadMan:[Personnage] -> boolean
    isEquiped:[Personnage] -> boolean
    getObject:[Personnage] -> Object
    getCarried: [Personnage] -> Personnage

Constructors:
    init: String * int * int * int * int -> Personnage
        pre init(nom,width,height,depth,force) require nom != ""
        AND force > 0 AND width > 0 AND height > 0 AND depth > 0

Operators:
    addHp: [Personnage] * int -> Personnage
        pre addHp(P,s) require s > 0 & !youDeadMan(P)
    removeHp: [Personnage] * int -> Personnage
        pre removeHp(P,s) require s> 0 & !youDeadMan(P)
    addMoney: [Personnage] * int -> Personnage
        pre addMoney(P,s) require s>0 & !youDeadMan(P)
    removeMoney: [Personnage] * int -> Personnage
        pre removeMoney(P,s) require s>0 & money(p)-s >= 0 & !youDeadMan(p)
    throw: [Personnage] -> Personnage
        pre throw(p) require isEquiped(p) & !youDeadMan(P)
    pickUp: [Personnage] * Object -> Personnage
        pre pickUp(p,o) require !youDeadMan(p)
    pickUpPersonnage: [Personnage] * [Personnage] -> Personnage
        pre pickUp(p,p1) require !youDeadMan(p)

Observation:
[invariant]
    youDeadMan(p) min= hp(p) <= 0
    isEquiped(p) min= getObject(p) != null OR getCarried(p) != null
    currentForce(p) min= Force(p) + getObject(p)::power

[init]
    nom(init(n,l,h,p,f))=n
    width(init(n,l,h,p,f))=l
    height(init(n,l,h,p,f))=h
    depth(init(n,l,h,p,f))=p
    force(init(n,l,h,p,f))=f
    hp(init(n,l,h,p,f))=100
    money(init(n,l,h,p,f))=0
    isEquiped(init(n,l,h,p,f))=false
    getObject(init(n,l,h,p,f))=null
    currentForce((init(n,l,h,p,f)) = f
[addHp]
    hp(addHp(p,s))=hp(p)+s
[removeHp]
    hp(removeHp(p,s))=min(hp(p)-s,0)
    youDeadMan(remove(p,hp(p)))=true
[addMoney]
    money(addMoney(p,s))=money(p)+s
[removeMoney]
    money(removeMoney(p,s))=min(money(p)-s,0)
[throw]
    isEquiped(throw(p))=false
    getObject(throw(p))= false
    getCarried(throw(p))=false
    currentForce(throw(p))=force(p)

[pickUp]
    isEquiped(pickUp(p,Object::init("o",usable,10)))=true
    isEquiped(pickUp(p,Object::init("o",sellabe,10)))=false
    getObject(pickUp(p,Object::init("o",usable,10))))=Object::init("o",usable,10))
    money(pickUp(p,Object::init("o",sellable,10))))=money(p)+10
    getCarried(pickUp(p,o)) = null
    currentForce(pickUp(p,Object::init("o",usable,10))) = currentForce(p) +10
[pickUpPersonnage]
    isEquiped(pickUpPersonnage(p,p1)) = true
    getObject(pickUpPersonnage(p,p1)) = null
    getCarried(pickUpPersonnage(p,p1)) = p1
```

1.2 Test:

```
init

Test:   p = init("m",10,10,10,10)
oracle: nom(p) = "m"
        largeur(p) = 10
        hauter(p) = 10
        profonder(p) = 10
        force(p) = 10
        hp(p) = 100
        money(p) = 0
        alive(p) = true
        equiped(p) = false
        object?(p) = null


Test: p = init("",10,10,10,10)
oracle: p = null

Test: p = init("m",10,10,10,-1)
oracle: p = null

Test: p = init("m",10,10,10,0)
oracle: p = null

Test: p = init("m",0,10,10,10)
oracle: p = null

Test: p = init("m",10,0,10,10)
oracle: p = null

Test: p = init("m",10,10,0,10)
oracle: p = null

Test pour le addHp

Initial: p = init("m",10,10,10,10)
Test: p1 = addHp(p,10)
oracle: hp(p1) = 100

Initial: p = init("m",10,10,10,10)
            removeHp(p,50)
Test: p1 = addHp(p,10)
oracle: hp(p1) = hp(p) + 10

Initial: p = init("m",10,10,10,10)
Test: p1 = addHp(p,-10)
oracle: hp(p1) == hp(p)
```

```
Initial: p = init("m",10,10,10,10)
         youDeadMan(p) = true
Test: p1 = addHp(p)
oracle: hp(p1) = hp(p)
```

```
removeHp
```

```
Initial: p = init("m",10,10,10,10)
Test: p1 = removeHp(p,10)
oracle: hp(p1) = hp(p) - 10
```

```
Initial: p = init("m",10,10,10,10)
Test: p1 = removeHp(p,-10)
oracle: hp(p1) == hp(p)

Initial: p = init("m",10,10,10,10)
        youDeadMan(p) = true
Test: p1 = removeHp(p,10)
oracle: hp(p1) == hp(p)

addMoney
```

```
Initial: p = init("m",10,10,10,10)
        youDeadMan(p) = true
Test: p1 = addMoney(p,10)
oracle: money(p1) = money(p)
```

```
Initial: p = init("m",10,10,10,10)
Test: p1 = addMoney(p,10)
oracle: money(p1) = money(p) + 10
```

```
Initial: p = init("m",10,10,10,10)
Test: p1 = addMoney(p,-10)
oracle: money(p1) = money(p)

removeMoney

Initial: p = init("m",10,10,10,10)
Test: p1 = removeMoney(p,10)
oracle: money(p1) = money(p)

Initial: p = init("m",10,10,10,10)
        addMoney(p,50)
```

```
Test: p1 = removeMoney(p,10)
oracle: money(p1) = money(p) - 10

Initial: p = init("m",10,10,10,10)
Test: p1 = removeMoney(p,-10)
oracle: money(p1) == money(p)


jeter

Initial: p = init("m",10,10,10,10)
        pickUp(p,obj)
Test: p = jeter(p)
oracle: object?(p) = null


Initial: p = init("m",10,10,10,10)
        pickUpPersonnage(p,p1)
Test: p = jeter(p)
oracle: getCarried(p) = null

Initial: p = init("m",10,10,10,10)
        pickUp(p,obj)
        youDeadMan(p) = true
Test: p1 = jeter(p)
oracle: object?(p) = object?(p1)

Test pour le pickUp
Initial: p = init("m",10,10,10,10)
        obj= init("m",usuable,10)
Test: p = pickUp(p,obj)
oracle: object?(p) = obj
        isEquiped(p) = true


Initial: p = init("m",10,10,10,10)
        obj= init("m",sellable,10)
Test: p1 = pickUp(p,obj)
oracle: object?(p1) = object?(p1)
        isEquiped(p) = isEquiped(p1)
        money(p1) = money(p) + 10


Initial: p = init('m',10,10,10,10)
        obj= init("m",sellable,10)
        youDeadMan(p) = true
Test: p1 = pickUp(p,obj)
Oracle: object?(p1) = object?(p1)
    isEquiped(p) = isEquiped(p1)
```

```
pickUpPersonnage

Initial: p = init(\m",10,10,10,10)
     p1 = init(\m1",10,10,10,10)

Test: p2 = pickUpPersonnage(p,p1)
Oracle: Object(p2) = null
    isEquiped(p2) = true
    getCarried(p2) = p1

Initial: p = init("m",10,10,10,10)
     p1 = init("m1",10,10,10,10)
     youDeadMan(p) = true
Test: p2 = pickUpPersonnage(p,p1)
Oracle: Object(p2) = Object(p)
    isEquiped(p2) = Object(p)
    getCarried(p2) = getCarried(p)
```

2 Service Bloc

2.1 Spécification:

```
service: Bloc
use : object
types: boolean

observators:
    isEmpty: [Bloc] -> boolean
    isPit: [Bloc] -> boolean
    hasTreasure: [Bloc] -> boolean
    getTreasure: [Bloc] -> object
     pre getTreasure(b) require hasTreasure(b)

Constructors:
    init: boolean * object -> [Bloc]

Operators:
    removeTreasure:[Bloc] -> object
      pre: removeTreasure(b) require hasTreasure(b)

Observations:
    [invariants]
        hasTreasure(b) min= getTreasure(b) != null
        isPit(b) min= !isEmpty(b)
    [init]
        isEmpty(init(true,null)) = true
        isPit(init(true,null)) = false
        hasTreasure(init(true,null)) = false
        hasTreasure(init(true,obj) = true
        getTreasure(init(true,obj) = obj
    [removeTreasure]
        getTreasure(removeTreasure(init(p,obj)) = null
        hasTreasure(removeTreasure(init(p,obj)) = false
```

2.2 Test:

```
Tests pour le service Bloc


Test pour init:

Test: b= init(true,null)
oracle:
    isEmpty(b) = true
    isPit(b) = false
    hasTreasure(b) = false
    getTreasure(b) = null


Test: b = init(false,null)
oracle:
    isEmpty(b) = false
    isPit(b) = true
    hasTreasure(b) = false
    getTreasure(b) = null

Test: b = init(true,obj)
oracle:
    isEmpty(b) = true
    isPit(b) = false
    hasTreasure(b) = true
    getTreasure(b) = obj

Test pour removeTreasure:

Initial: b = init(true,null)
Test :  removeTreasure(b)
oracle:
    hasTreasure(b) = false
    getTreasure(b) = null

Initial: b = init(true,obj)
Test :  removeTreasure(b)
oracle:
    hasTreasure(b) = false
    getTreasure(b) = null
```

3 Service Terrain

3.1 Spécification:

```
service: Terrain
use: Bloc,List,Object
types : int

observators:
    - const length: [Terrain] -> int
    - const height: [Terrain] -> int
    - const depth: [Terrain] -> int
    - const nbBloc: [Terrain] -> int

    - BlocCoord: [Terrain] *int * int * int -> Bloc
        pre: getBlocCoord(w,h,d) require w>=0 AND w<width AND h>=0 AND h<height
        AND d>=0 AND d<depth


constructors:
    - init: int * int * int  -> [Terrain]
        pre: init(w,h,d) require w>0 AND h>0 AND d>0

operators:
    - setBlocCoord: int * int * int * Bloc -> [Terrain]
        pre: setBlocCoord(w,h,d,b) require w>=0 AND w<width AND h>=0 AND h<height
        AND d>=0 AND d<depth AND b !=null

Observations:

[Invariant]
    width > 0
    height > 0
    depth > 0
    nbBloc(T) = width*height*depth;

[Init]
    width(init(w,h,d)) = w;
    height(init(w,h,d)) = h;
    depth(init(w,h,d)) = d;
    nbBloc(init(w,h,d)) = width*height*depth
    BlocCoord(init(w,h,d),0 <= i < w, 0 <= j < h ,0 <= k <d) =
    Bloc::init(b,Object::Init(s,t,i)) //random

[setBlocCoord]
```

```
    BlocCoord(setBlocCoord(w,h,d),i,j,k,B) = B
```

3.2 Test:

```
Tests pour le service Terrain

Test init:

    testOk:
        test: t = init(1,1,1)

        oracle: width(t) = 1
                height(t) = 1
                length(t) = 1


    testPbm1:
        test: t = init(-1,1,1)

        oracle: t = null

    testPbm2:
        test: t = init(1,-1,1)

        oracle: t = null

    testPbm3:
        test: t = init(1,1,-1)

        oracle: t = null

    testPbm4:
        test: t = init(-1,-1,1 )

        oracle: t = null

    testPbm5:
        test: t = init(1,-1,-1)

        oracle: t = null

    testPbm6:
        test: t = init(-1,1,-1)

        oracle: t = null

    testPbm7:
        test: t = init(-1,-1,-1)

        oracle: t = null

TestSetBloc:

    testOK:
        init: t = init(1,1,1)
        test t2 = setBloc(t,1,1,1,B)
        oracle t2 != null

    testPbm1:
        init: t = init(1,1,1)
        test t2 = setBloc(t,-1,1,1,B)
        oracle t2 = null

    testPbm2:
        init: t = init(1,1,1)
        test t2 = setBloc(t,1,-1,1,B)
        oracle t2 = null

    testPbm3:
        init: t = init(1,1,1)
        test t2 = setBloc(t,1,1,-1,B)
        oracle t2 = null

    testPbm4:
        init: t = init(1,1,1)
        test t2 = setBloc(t,-1,-1,1,B)
        oracle t2 = null

    testPbm5:
        init: t = init(1,1,1)
        test t2 = setBloc(t,1,-1,-1,B)
        oracle t2 = null

    testPbm6:
        init: t = init(1,1,1)
        test t2 = setBloc(t,-1,1,-1,B)
        oracle t2 = null

    testPbm7:
        init: t = init(1,1,1)
        test t2 = setBloc(t,-1,-1,-1,B)
        oracle t2 = null

    testPbm21:
        init: t = init(1,1,1)
        test t2 = setBloc(t,2,1,1,B)
        oracle t2 = null
    testPbm22:
        init: t = init(1,1,1)
        test t2 = setBloc(t,1,2,1,B)
        oracle t2 = null

    testPbm23:
        init: t = init(1,1,1)
        test t2 = setBloc(t,1,1,2,B)
        oracle t2 = null

    testPbm24:
        init: t = init(1,1,1)
        test t2 = setBloc(t,2,2,1,B)
        oracle t2 = null

    testPbm25:
        init: t = init(1,1,1)
        test t2 = setBloc(t,1,2,2,B)
        oracle t2 = null

    testPbm26:
        init: t = init(1,1,1)
        test t2 = setBloc(t,2,1,2,B)
        oracle t2 = null

    testPbm27:
        init: t = init(1,1,1)
        test t2 = setBloc(t,2,2,2,B)
        oracle t2 = null
```

4 Service Objet

4.1 Spécification:

```
service: Object
types: String,int,enum Type{usable,sellable}
observators:
    - const nom : [Object] -> String
    - const type : [Object] -> Type
    - const power : [Object] -> int
    - const value : [Object] -> int
Constructors:
    init: String * Type * int
          pre init(nom,Type,i) require nom != "" AND i > 0
Operators:
    // we have none

Observations:
    [init]
      nom(init(m,usable,10))=m
      type(init(m,usable,10))=usable
      power(init(m,usable,10))=10
      value(init(m,usable,10))=0
      power(init(m,sellable,10))=0
      value(init(m,sellable,10))=10
```

4.2 Test:

```
Test pour service Object

Init:

Test: p = init("m",usable,10)
oracle: nom(p) = m
    type(p) = usable
    power(p) = 10
    value(p) = 0

Test: p = init("",usable,10)
oracle: p = null


Test: p = init("m",sellable,10)
oracle: nom(p) = "m"
    type(p) = sellable
    value(p) = 10
    power(p) = 0


Test: p = init("m",sellable,-10)
oracle: p = null



Test: p = init("m",sellable,0)
oracle: p = null
```

5 Service Gangster

5.1 Spécification:

```
Service: Gangster
Refine: Personnage
Constructors:
    init: String  -> Gangster
```

```
          pre init(nom) require nom != \"

##Observation:
  [init]
    getObject(init(n))= o
    nom(init(n))=n
    width(init(n))=5
    height(init(n))=5
    depth(init(n))=5
    force(init(n))=20
    hp(init(n))=100
    money(init(n))=0
    isEquiped(init(n))=false

  [addHp]
    hp(addHp(p,s))=hp(p)

  [addMoney]
    money(addMoney(p,s))=0

  [removeMoney]
    money(removeMoney(p,s))=0

  [throw]
    isEquiped(throw(p))=isEquiped(p)

  [pickUp]
    isEquiped(pickUp(p,o))=isEquiped(p)
    getObject(pickUp(p,o))=getObject(p)
```

6 Service StatusWrapper

6.1 Spécification:

```
service: StatusWrapper
use : Personnage
types: int, enum COMMAND{NONE,UP,DOWN,LEFT,RIGHT, JUMP_UP,JUMP_DOWN,
JUMP_LEFT,JUMP_RIGHT,KICK,THROW, PICKUP}

observators:
    x: [StatusWrapper] -> int
    y: [StatusWrapper] -> int
    z: [StatusWrapper] -> int
    freeze: [StatusWrapper] -> int
    direction: [StatusWrapper] -> COMMAND
    personnage: [StatusWrapper] -> Personnage
    isFrozen: [StatusWrapper] -> boolean
```

```
Constructors:
    init: int * int * int * COMMAND * Personnage -> [StatusWrapper]
        pre: init(x,y,z,c,p) require x>=y>=z>=0 AND c IN {UP,DOWN,LEFT,RIGHT}
        AND p != null

Operators:
    setX: [StatusWrapper] * int -> [StatusWrapper]
        pre: setX(S,x) require x>=0
    setY: [StatusWrapper] * int -> [StatusWrapper]
        pre: setY(S,x) require x>=0
    setZ: [StatusWrapper] * int -> [StatusWrapper]
        pre: setZ(S,x) require x>=0
    setFreeze: [StatusWrapper] * int -> [StatusWrapper]
        pre: setFreeze(S,x) require x>=0
    setDirection: [StatusWrapper] * int -> [StatusWrapper]
        pre: setDirections(S,c) require c IN {UP,DOWN,LEFT,RIGHT}
    decFreeze: [StatusWrapper] -> [StatusWrapper]

Observations:
    [invariants]
        0<=x(S)
        0<=y(S)
        0<=z(S)
        0<=freeze(S)
        isFrozen(S) =min= (freeze==0)
        getDirection(c) IN {UP,DOWN,LEFT,RIGHT}

    [init]
        x(S,init(x,y,z,c,p)) = x
        y(S,init(x,y,z,c,p)) = y
        z(S,init(x,y,z,c,p)) = z
        freeze(S,init(x,y,z,c,p)) = 0
        direction(S,init(x,y,z,c,p)) = c
        personnage(S,init(x,y,z,c,p)) = p

    [setX]
        x(setX(S,n)) = n

    [setY]
        y(setY(S,n)) = n

    [setZ]
        z(setZ(S,n)) = n

    [setFreeze]
        freeze(setFreeze(S,n)) = n

    [setDirection]
        direction(setDirection(S,n)) = n
    [decFreeze]
        freeze(decFreeze(S,n)) = max(0,freeze(S)-1);
```

6.2 Test:

```
TestStatusWrapper.txt

test invariants:
        c is a statusWrapper
        Oracle:
            x(c)>=0 & y(c)>=0 & z(c)>=0
            freeze(c) >= 0
            getDirection(c) in {UP,DOWN,LEFT,RIGHT}

test init:

    testOK:
        c = init(1,1,1,UP,{JACK});
        oracle: CheckInvariants
            x(c)=y(c)=z(c) = 1
            getPerso(c) = {JACK}
            getDirection(c) = UP
            getFreeze(c) = 0

    testPre1:
        c = init(-1,1,1,LEFT,{JACK});
        oracle: c = precondition error

    testPre2:
        c = init(1,-1,1,LEFT,{JACK});
        oracle: c = precondition error

    testPre3:
        c = init(1,1,-1,LEFT,{JACK});
        oracle: c = precondition error

    testPre4:
        c = init(1,1,1,JUMP_LEFT,{JACK});
        oracle: c = precondition error

    testPre5:
        c = init(1,1,1,LEFT,NULL);
        oracle: c = precondition error
    + toutes les combinaisons

test setX/Y/Z/Freeze:

    testOK:
        c = init(1,1,1,UP,{JACK});
        c2 = setX(c,2)
        oracle: getX(c2) = 2;
        c3 = setY(c2,3)
        oracle: getY(c3) = 3;
        c4 = setZ(c3,4)
        oracle: getZ(c4) = 4;
        c5 = setFreeze(c4,5)
        oracle: freeze(c5) = 5;

    testKO:
        c = init(1,1,1,UP,{JACK});
        c2 = setX(c,-1)
        oracle: precondition error
        c3 = setY(c2,-1)
        oracle: precondition error
        c4 = setZ(c3,-3)
        oracle: precondition error
        c5 = setFreeze(c4,-5)
        oracle: precondition error

test setDirection:

    testOK:
        c = init(1,1,1,UP,{JACK});
        c2 = setDirection(c,DOWN);
        oracle: getDirection(c2) = DOWN;
    testKO:
        c = init(1,1,1,UP,{JACK});
        c2 = setDirection(c,PICKUP)
        oracle: precondition error

test decFreeze

    testOK:
        c = init(1,1,1,UP,{JACK});
        c2 = setFreeze(c,1)
        c3 = decFreeze(c2)
        oracle: freeze(c3) = 0
        c4 = decFreeze(c3)
        oracle: freeze(c4) = 0
```

7 Service MoteurJeu

7.1 Spécification:

```
service: MoteurJeu
use: GestionCombat
types: boolean, enum COMMAND{NONE,UP,DOWN,LEFT,RIGHT, JUMP_UP,JUMP_DOWN,
JUMP_LEFT,JUMP_RIGHT,KICK,THROW, PICKUP}, enum RESULT{WON,LOST,TIED}

Observators:

    isFinished: [MoteurJeu] -> boolean
    finalResult : [MoteurJeu] -> RESULT
        pre finalResult(M) require isFinished(M)
    combat : [MoteurJeu] -> GestionCombat

Constructor:

    init: int*int*int -> [MoteurJeu]
        pre init(w,h,d) require w >= h >= d > 0

Operators:

    gameStep: [MoteurJeu] * COMMAND * COMMAND -> [MoteurJeu]
        gameStep(M,CR,CA) require M != null ^ !isFinished(M)

Observations:

    [invariants]

        isFinished(M) =min=
            (Personnage::youDeadMan(GestionCombat::Slick(combat(M))) = true)
            OR
            (
            Personnage::youDeadMan(GestionCombat::Alex(combat(M))) = true
            AND
            Personnage::youDeadMan(GestionCombat::Ryan(combat(M))) = true
            )

        finalResult(M) =min=
            1-  if (Gangster::youDeadMan(GestionCombat::Slick(combat(M))) = true)
                AND [(Personnage::youDeadMan(GestionCombat::Alex(combat(M))) = false)
                    OR (Personnage::youDeadMan(GestionCombat::Ryan(combat(M))) = false)]
                WON

            2- if (Gangster::youDeadMan(GestionCombat::Slick(combat(M))) = false)
                AND [(Personnage::youDeadMan(GestionCombat::Alex(combat(M))) = true)
                    AND (Personnage::youDeadMan(GestionCombat::Ryan(combat(M))) = true)]
                LOST
            3- if (Gangster::youDeadMan(GestionCombat::Slick(combat(M))) = true)
                AND [(Personnage::youDeadMan(GestionCombat::Alex(combat(M))) = true)
                    AND (Personnage::youDeadMan(GestionCombat::Ryan(combat(M))) = true)]
                TIED

    [init]

        combat(init(w,h,d)) = GestionCombat::init(w,h,d);

    [gameStep]

        combat(gamestep(M,CR,CA)) = GestionCombat::step(Combat(M),CR,CA);
```

8 Service GestionCombat

8.1 Spécification:

```
Service: GestionCombat
uses: Personnage , Terrain , Gangster, StatusWrapper
types: boolean , int , String , enum COMMAND{UP,DOWN,LEFT,RIGHT, JUMP_UP,
JUMP_DOWN,JUMP_LEFT,JUMP_RIGHT,KICK,THROW, PICKUP}

observators:

    const length: [GestionCombat] -> int
    const height : [GestionCombat] -> int
    const width : [GestionCombat] -> int
    const nbGangsters: [GestionCombat] -> int

    const Terrain: [GestionCombat] -> Terrain

    alex: [GestionCombat] -> StatusWrapper
    ryan: [GestionCombat] -> StatusWrapper
    slick: [GestionCombat] -> StatusWrapper
    gangsters: [GestionCombat] -> StatusWrapper[]

    inRange: [GestionCombat] * [StatusWrapper] * [StatusWrapper] -> boolean
        pre: inRange(C,p1,p2) require p1 != p2 =! NULL
            AND p1 is {slick,alex,ryan, gangster[0:nbGangsters-1]}
            AND p2 is {slick,alex,ryan, gangster[0:nbGangsters-1]}


Constructors:
    init: int * int * int -> [GestionCombat]
          pre init(x,y,z) require x > 50 && y > 50 && z > 50

Operators:
    step: [GestionCombat] * COMMANDE * COMMANDE -> [GestionCombat]
```

```
        pre: step(C,CR,CA) require CR != Null AND CA != Null

Observations:

[Invariant]
    inRange(C,p1,p2) =min=
    DIM:["x","y","z"]: AND{
        Position(C,recupPersonnage(C,p),DIM) = Position(C,recupPersonnage(C,p2),DIM)+1
        OR
        Position(C,recupPersonnage(C,p),DIM) = Position(C,recupPersonnage(C,p2),DIM)-1
    }


[init]

  width(init(w,l,d))=w
  length(init(w,l,d))=l
  depth(init(w,l,d))=d
  nbGangsters(init(w,l,d)) = w*l*0.30 //30 % du territoire est peuplé de vil méchants

  alex(init(w,l,d))) = Personnage::init("alex",5,5,5,50)
  ryan(init(w,l,d))) = Personnage::init("ryan",5,6,5,50)
  slick(init(w,l,d)) = Gangster::init("slick")
  Terrain(init(w,l,d)) = Terrain::init(w,l,d)
  lastCommand(init(w,l,d),P) = null;


  [FOR ALL( i=[5:w-1] , j=[0:h-1] , k=[0,d-1])
   WHERE Bloc::IsEmpty(Terrain::BlocCoord(i,j,k)
   AND Random::True
   AND Index < nbGangster
   ]:index
    ==> {
        Gangster[index](init(w,l,d)) = Gangster::init("Scumbag");
        position(C,"Gangster_"+index,"x") = i;
        position(C,"Gangster_"+index,"y") = j;
        position(C,"Gangster_"+index,"z") = k;
        }

   freeze(C,p) = 0
   position(C,"Ryan","x") = 0;
   position(C,"Alex","x") = 0;
   position(C,"Slick","x") = l-1;
   position(C,"Ryan","y") = 0;
   position(C,"Alex","y") = 0;
   position(C,"Slick","y") = 0;
   position(C,"Ryan","z") = 1;
   position(C,"Alex","z") = 0;
   position(C,"Slick","z") = 0;
```

```
[step]

    lastCommand(step(C,CA,CR)) = (lastCommand(Alex(C))
    ==
    CA AND lastCommand(Ryan(c)) == CR)

    alex(step(C,CR,CA)) =
        Personnage::removeHP(alex(C),Gangster::force(p))
            SI: p is Gangster
                AND inRange(C,p, Alex(C))
                AND CA != COMMAND.KICK
                AND freeze(Alex(C)) != 0
                AND freeze(p) = 0;

        Personnage::addMoney(Alex(C),
        Object::Value(Bloc::RemoveTreasure(
        Terrain::GetBlocCoord(Position(C,"Alex","x"),y,z))));
            SI: CA = COMMAND.PICKUP AND BLOC::HasTreasure And Terrain::isEmpty;

    ryan(step(C,CR,CA)) =
        Personnage::removeHP(ryan(C),Gangster::force(p)
            SI: p is Gangster
                AND inRange(C,p, ryan(C))
                AND CR != COMMAND.KICK
                AND freeze(ryan(C)) != 1
                AND freeze(p) != 1;

        Personnage::addMoney(Ryan(C),
        Object::Value(Bloc::RemoveTreasure(
        Terrain::GetBlocCoord(Position(C,"ryan","x"),y,z))));
            SI: CA = COMMAND.PICKUP AND BLOC::HasTreasure And Terrain::isEmpty;s

    slick(step(C,CR,CA)) =
        Personnage::removeHP(slick(C),Personnage::force(p)
            SI: p is Alex(C)
            AND CA == COMMAND.KICK OR p is Ryan(C) AND CR == COMMAND.KICK
                AND inRange(C,p,slick(C))
                AND freeze(p) = 0

    gangsters[i](step(C,CR,CA))
        Personnage::removeHP(Gangster[i](C),Personnage::force(p)
            SI: p is Alex(C)
            AND CA == COMMAND.KICK OR p is Ryan(C) AND CR == COMMAND.KICK
                AND inRange(C,p,Gangster[i](C))
                AND freeze(p) = 0


    position(step(C,CR,CA),Alex(C)) =
        position(C,"alex","x")+1 SI CA = COMMAND.RIGHT AND COMMAND.JUMP_RIGHT;
        position(C,"alex","y")+1 SI CA = COMMAND.UP AND COMMAND.JUMP_UP;
        position(C,"alex","x")-1 SI CA = COMMAND.LEFT AND COMMAND.JUMP_LEFT;
        position(C,"alex","y")-1 SI CA = COMMAND.DOWN AND COMMAND.JUMP_DOWN;



    position(step(C,CR,CA),Ryan(C)) =
        position(C,"ryan","x")+1 SI CR = COMMAND.RIGHT AND COMMAND.JUMP_RIGHT;
        position(C,"ryan","y")+1 SI CR = COMMAND.UP AND COMMAND.JUMP_UP;
        position(C,"ryan","x")-1 SI CR = COMMAND.LEFT AND COMMAND.JUMP_LEFT;
        position(C,"ryan","y")-1 SI CR = COMMAND.DOWN AND COMMAND.JUMP_DOWN;

    position(step(C,CR,CA),Slick(C)) =
        min(position(C,"slick","x")+3,l)
            SI:
            [freeze(C,Ryan(C)) = 0 AND inRange(C,Ryan(C),Slick(C))
            AND lastCommand(C,Ryan(C)) = COMMAND.LEFT OR COMMAND.JUMP_LEFT;
            AND CR = COMMAND.KICK]
            OR
            [freeze(C,Alex(C)) = 0 AND inRange(C,Alex(C),Slick(C))
            AND lastCommand(C,Alex(C)) = COMMAND.LEFT OR COMMAND.JUMP_LEFT;
            AND CR = COMMAND.KICK]

        min(position(C,"slick","y")+3,w)
            SI:
            [freeze(C,Ryan(C)) = 0 AND inRange(C,Ryan(C),Slick(C))
            AND lastCommand(C,Ryan(C)) = COMMAND.UP OR COMMAND.JUMP_UP;
            AND CR = COMMAND.KICK]
            OR
            [freeze(C,Alex(C)) = 0 AND inRange(C,Alex(C),Slick(C))
            AND lastCommand(C,Alex(C)) = COMMAND.UP OR COMMAND.JUMP_UP;
            AND CR = COMMAND.KICK]

        max(position(C,"slick","x")-3,0)
            SI:
            [freeze(C,Ryan(C)) = 0 AND inRange(C,Ryan(C),Slick(C))
            AND lastCommand(C,Ryan(C)) = COMMAND.RIGHT OR COMMAND.JUMP_RIGHT;
            AND CR = COMMAND.KICK]
            OR
            [freeze(C,Alex(C)) = 0 AND inRange(C,Alex(C),Slick(C))
            AND lastCommand(C,Alex(C)) = COMMAND.RIGHT OR COMMAND.JUMP_RIGHT;
            AND CR = COMMAND.KICK]

        max(position(C,"slick","y")-3,0)
            SI:
            [freeze(C,Ryan(C)) = 0 AND inRange(C,Ryan(C),Slick(C))
            AND lastCommand(C,Ryan(C)) = COMMAND.DOWN OR COMMAND.JUMP_DOWN;
            AND CR = COMMAND.KICK]
            OR
            [freeze(C,Alex(C)) = 0 AND inRange(C,Alex(C),Slick(C))
            AND lastCommand(C,Alex(C)) = COMMAND.DOWN OR COMMAND.JUMP_DOWN;
            AND CR = COMMAND.KICK]


    position(step(C,CR,CA),Gangster[i](C)) =
        min(position(C,"Gangster_i","x")+3,l)
            SI:
            [freeze(C,Ryan(C)) = 0 AND inRange(C,Ryan(C),Gangster[i](C))
            AND lastCommand(C,Ryan(C)) = COMMAND.LEFT OR COMMAND.JUMP_LEFT;
            AND CR = COMMAND.KICK]
            OR
            [freeze(C,Alex(C)) = 0 AND inRange(C,Alex(C),Gangster[i](C))
            AND lastCommand(C,Alex(C)) = COMMAND.LEFT OR COMMAND.JUMP_LEFT;
            AND CR = COMMAND.KICK]

        min(position(C,"Gangster_i","y")+3,w)
            SI:
            [freeze(C,Ryan(C)) = 0 AND inRange(C,Ryan(C),Gangster[i](C))
            AND lastCommand(C,Ryan(C)) = COMMAND.UP OR COMMAND.JUMP_UP;
            AND CR = COMMAND.KICK]
            OR
            [freeze(C,Alex(C)) = 0 AND inRange(C,Alex(C),Gangster[i](C))
            AND lastCommand(C,Alex(C)) = COMMAND.UP OR COMMAND.JUMP_UP;
            AND CR = COMMAND.KICK]

        max(position(C,"Gangster_i","x")-3,0)
            SI:
            [freeze(C,Ryan(C)) = 0 AND inRange(C,Ryan(C),Gangster[i](C))
            AND lastCommand(C,Ryan(C)) = COMMAND.RIGHT OR COMMAND.JUMP_RIGHT;
            AND CR = COMMAND.KICK]
            OR
            [freeze(C,Alex(C)) = 0 AND inRange(C,Alex(C),Gangster[i](C))
            AND lastCommand(C,Alex(C)) = COMMAND.RIGHT OR COMMAND.JUMP_RIGHT;
            AND CR = COMMAND.KICK]

        max(position(C,"Gangster_i","y")-3,0)
            SI:
            [freeze(C,Ryan(C)) = 0 AND inRange(C,Ryan(C),Gangster[i](C))
            AND lastCommand(C,Ryan(C)) = COMMAND.DOWN OR COMMAND.JUMP_DOWN;
            AND CR = COMMAND.KICK]
            OR
            [freeze(C,Alex(C)) = 0 AND inRange(C,Alex(C),Gangster[i](C))
            AND lastCommand(C,Alex(C)) = COMMAND.DOWN OR COMMAND.JUMP_DOWN;
            AND CR = COMMAND.KICK]

    freeze(step(C,CR,CA),Alex(C)) =
        freeze(C,Alex(C))+1
            SI: freeze(C,Alex(C)) = 0 AND CA = COMMAND.KICK;

    freeze(step(C,CR,CA),Ryan(C)) =
        freeze(C,Ryan(C))+1
```

```
        SI: freeze(C,Ryan(C)) = 0 AND CR = COMMAND.KICK;

freeze(step(C,CR,CA),Slick(C)) =
    freeze(C,Slick(C)) = freeze(C,Slick(C))+3
        SI: CR = COMMAND.KICK AND inRange(C,Slick(C), ryan(C))
        AND freeze(ryan(c)) = 0
            OR CA = COMMAND.KICK AND inRange(C,Slick(C), Alex(C)) AND freeze(alex(c))

freeze(step(C,CR,CA),Gangster[i](C)) =
    freeze(C,Gangster[i](C)) = freeze(C,Gangster[i](C))+3
        SI: CR = COMMAND.KICK
        AND inRange(C,Gangster[i](C), ryan(C)) AND freeze(ryan(c)) = 0
        OR CA = COMMAND.KICK AND inRange(C,Gangster[i](C), Alex(C))
        AND freeze(alex(c)) = 0

freeze(step(C,CR,CA),Gangster[i](C))=
    freeze(C,Gangster[i](C)) = freeze(C,Gangster[i](C))+3
        SI: CR = COMMAND.CARRY
freeze(step(C,CR,CA),Gangster[i](C))= !freeze(C,Gangster[i](C))
            SI CR = COMMAND.THROW AND alex::getCarried=Gangster[i](C)
```