

Rapport de projet

CPS : River City Ransom

Béatrice CARRE
Steven VAROUMAS

16 avril 2014

introduction

Méthodologie pour la conception et l'implémentation de trusted components

1 Analyse : spécifications algébriques (cours 2)

2 Conception : par contrat à partir des spécifications (cours 3 et 4)

3 Implémentation : garantie vis-à-vis des contrats

Conception par contrat : vérification « légère » en ligne (self-testing)

1 Problème

petite intro ?

1.1 Specification

Le but de la spécification est de reconnaître les éléments logiques d'un programme à partir d'une définition du jeu River City Ransom.

problèmes d'interprétation personnelle

combler le manque d'éléments

rester cohérents

importante notion de référentiel pour chaque service

représentation d'une liste difficile

représentation de la notion de temps difficile

1.2 Tests

Décrire et implémenter les tests pour le respect du contrat.

Pour n conditions sur chaque opération $\Rightarrow (2 \text{ à } \sim 6)^n$ tests ! Pour chaque argument de méthode, il faut trouver les valeurs à tester. Si booléen, il suffit de tester avec true et false. Mais si entier float string etc, il est impossible de tester toutes les valeurs. Il faut

donc sélectionner les valeurs les plus pertinentes à tester, qui sont en général les valeurs juste avant et après les limites d'intervalles de tests. Par exemple :

Schéma à insérer ?

Théorie : tous les tests réussissent signifierait que nous avons une application sans bug...

Réalité : il faut aussi que l'implémentation respecte ce qui est demandé, et toutes les valeurs ne sont pas testées, il est donc impossible d'être sûrs.

1.3 Implémentation par contrat

développer les services et conditions soulevés, en respectant le squelette étudié.

2 Solution

petite intro ?

2.1 Specification

choix d'implémentation :

ramasser => 3 méthodes différentes

1 objet par bloc => jeter un objet que sur bloc vide

action gangster => commande aléatoire

ramasser un perso = équipé comme un objet donc invisible

argent n'est pas un objet équipé => peut se récolter autant que l'on veut, même si équipé de quelque chose

Les blocs sont en 2D au sol, mais les personnages peuvent sauter sur le troisième axe.

observateur estVisible() pour afficher ou non des gangsters/persos

2.2 Tests

Nous n'avons malheureusement pas fait tout les tests.

Idéalement : pour chaque méthodes, pour chaque valeur :

2.3 Implémentation par contrat

CCL

Spécifications

0.1 Le service Personnage

service : Personnage
use : Objet
types : String , int , boolean

Observers :

```
const nom : [Personnage] → String
const largeur : [Personnage] → int
const hauteur : [Personnage] → int
const profondeur : [Personnage] → int
const force : [Personnage] → int
points_de_vie : [Personnage] → int
somme_d_argent : [Personnage] → int
est_vaincu : [Personnage] → boolean
est_equipe_objet : [Personnage] → boolean
est_equipe_perso : [Personnage] → boolean
objet_equipe : [Personnage] → Objet
pre objet_equipe(P) require est_equipe_objet(P)
perso_equipe : [Personnage] → Personnage
pre perso_equipe(P) require est_equipe_perso(P)
```

Constructors :

```
init : String × int × int × int × int × int × int → [Personnage]
pre init(nom, largeur, hauteur, profondeur, force, pdv, argent) require nom = "Alex" ∨ nom
= "Ryan" ∧ largeur > 0 ∧ hauteur > 0 ∧ profondeur > 0 ∧ force > 0 ∧ pdv > 0 ∧ argent ≥ 0
```

Operators :

```
retrait_vie : [Personnage] × int → [Personnage]
pre retrait_vie(P, s) require ¬est_vaincu(P) ∧ s > 0

retrait_argent : [Personnage] × int → [Personnage]
pre retrait_argent(P, s) require ¬est_vaincu(P) ∧ s > 0 ∧ somme_d_argent(P) ≥ s

depot_argent : [Personnage] × int → [Personnage]
pre depot_argent(P, s) require ¬est_vaincu(P) ∧ s > 0

ramasser_argent : [Personnage] × Object → [Personnage]
pre ramasser_argent(P, o) require ¬est_vaincu(P) ∧ Objet :: est_de_valeur(o)

ramasser_objet : [Personnage] × Object → [Personnage]
pre ramasser_objet(P, o) require ¬est_vaincu(P) ∧ ¬est_equipe_objet(P) ∧ ¬
est_equipe_perso(P) ∧ Objet :: est_equipable(o)

ramasser_perso : [Personnage] × Personnage → [Personnage]
pre ramasser_perso(P, p) require ¬est_vaincu(P) ∧ ¬est_equipe_objet(P) ∧
¬est_equipe_perso(P)

jeter : [Personnage] → [Personnage]
```

pre jeter(P) **require** $\neg \text{est_vaincu}(P) \wedge (\text{est_equipe_objet}(P) \vee \text{est_equipe_perso}(P))$

Observations :

[invariants]

$\text{est_vaincu}(P) \stackrel{\text{min}}{=} \text{points_de_vie}(P) \leq 0$
 $\text{est_equipe_perso}(P) \stackrel{\text{min}}{=} \text{perso_equipe}(P) \neq \text{null}$
 $\text{est_equipe_objet}(P) \stackrel{\text{min}}{=} \text{objet_equipe}(P) \neq \text{null}$

[init]

$\text{nom}(\text{init}(n, l, h, p, f, v, a)) = n$
 $\text{largeur}(\text{init}(n, l, h, p, f, v, a)) = l$
 $\text{hauteur}(\text{init}(n, l, h, p, f, v, a)) = h$
 $\text{profondeur}(\text{init}(n, l, h, p, f, v, a)) = p$
 $\text{force}(\text{init}(n, l, h, p, f, v, a)) = f$
 $\text{points_de_vie}(\text{init}(n, l, h, p, f, v, a)) = v$
 $\text{somme_d_argent}(\text{init}(n, l, h, p, f, v, a)) = a$
 $\text{objet_equipe}(\text{init}(n, l, h, p, f, v, a)) = \text{null}$
 $\text{perso_equipe}(\text{init}(n, l, h, p, f, v, a)) = \text{null}$

[retrait_vie]

$\text{points_de_vie}(\text{retrait_vie}(P, s)) = \max(0, \text{points_de_vie}(P) - s)$

[retrait_argent]

$\text{somme_d_argent}(\text{retrait_argent}(P, s)) = \text{argent}(P) - s$

[depot_argent]

$\text{somme_d_argent}(\text{depot_argent}(P, s)) = \text{argent}(P) + s$

[ramasser_objet]

$\text{objet_equipe}(\text{ramasser_objet}(P, \text{objet})) = \text{objet}$
 $\text{force}(\text{ramasser_objet}(P, \text{objet})) = \text{force}(P) + \text{Objet} :: \text{bonus_force}(\text{objet})$

[ramasser_argent]

$\text{somme_d_argent}(\text{ramasser_objet}(P, \text{objet})) = \text{somme_d_argent}(P) + \text{Objet} :: \text{valeur_marchande}(\text{objet})$

[ramasser_perso]

$\text{perso_equipe}(\text{ramasser_perso}(P, \text{perso})) = \text{perso}$

[jeter]

$\text{perso_equipe}(\text{jeter}(P)) = \text{null}$
 $\text{force}(\text{jeter}(P)) =$
 $\begin{cases} \text{force}(P) - \text{Objet} :: \text{bonus_force}(\text{objet_equipe}(P)) & \text{si } \text{est_equipe_objet}(P) \\ \text{force}(P) & \text{sinon} \end{cases}$
 $\text{objet_equipe}(\text{jeter}(P)) = \text{null}$

0.2 Gangster

service : Gangster
Refine : Personnage

Constructors :

init : String × int × int × int × int × int → [Gangster]
pre **init**(nom, largeur, hauteur, profondeur, force, pdv) **require** nom ≠ "" ∧ largeur > 0 ∧ hauteur > 0 ∧ profondeur > 0 ∧ force > 0 ∧ pdv > 0

Observations :

[init]
nom(**init**(n, l, h, p, f, v)) = n
largeur(**init**(n, l, h, p, f, v)) = l
hauteur(**init**(n, l, h, p, f, v)) = h
profondeur(**init**(n, l, h, p, f, v)) = p
force(**init**(n, l, h, p, f, v)) = f
points_de_vie(**init**(n, l, h, p, f, v)) = v
somme_d_argent(**init**(n, l, h, p, f, v)) = 0
objet_equipe(**init**(n, l, h, p, f, v)) = null
perso_equipe(**init**(n, l, h, p, f, v)) = null

[retrait_argent]
somme_d_argent(retrait_argent(G, s)) = argent(G)

[depot_argent]
somme_d_argent(depot_argent(G, s)) = argent(G)

[ramasser_argent]
somme_d_argent(ramasser_objet(G, objet)) = somme_d_argent(G)

0.3 Bloc

service : Bloc
use : Objet
types : enum TYPE{VIDE, FOSSE, OBJET},
Observers :

const type : [Bloc] → TYPE
const objet : [Bloc] → Objet

Constructors :

init : TYPE × Objet → [Bloc]
pre **init**(t, o) **require**
(t = VIDE ∨ t = FOSSE) ∧ o = null ∨ (t = OBJET ∧ o ≠ null)

Operators :

retirerObjet : [Bloc] → [Bloc]
pre retirerObjet(B) **require** type(B) = OBJET
poserObjet : [Bloc] × Objet → [Bloc]
pre poserObjet(B, o) **require** type(B) = VIDE

Observations :

[init]
type(**init**(t, o)) = t
objet(**init**(t, o)) = o
[retirerObjet]
type(retirerObjet(B)) = VIDE
objet(retirerObjet(B)) = null
[poserObjet]
type(poserObjet(B, o)) = OBJET
objet(poserObjet(B, o)) = o

0.4 Objet

```
service : Objet
types : String, boolean, int
Observers :
  const nom : [Object] → String
  est_equipable : [Object] → boolean
  est_de_valeur : [Object] → boolean
  bonus_force : [Object] → int
  pre bonus_force(O) require est_equipable(O)
  valeur_marchande : [Object] → int
  pre valeur_marchande(O) require est_de_valeur(O)

Constructors :

  init : String × int × int → [Object]
  pre (init(n,t,bonus,valeur) require n≠"" ∧ ( ( bonus >0 ∧ valeur = 0) ∨ (bonus = 0 ∧
    valeur > 0) ) )

Observations :
  [Invariants]
    est_equipable(O)  $\stackrel{min}{=}$  bonus_force > 0
    est_de_valeur(O)  $\stackrel{min}{=}$  valeur_marchande > 0
    est_equipable(O)  $\stackrel{min}{=}$  ¬est_de_valeur(O)

  [init]
    nom(init(n,bonus,valeur)) = n
    bonus_force(init(n,bonus,valeur)) = bonus
    valeur_marchande(init(n,bonus,valeur)) = valeur
```

0.5 Terrain

```
service : Terrain
use : Bloc
types : int
Observers :
  const largeur : [Terrain] → int
  const hauteur : [Terrain] → int
  const profondeur : [Terrain] → int
  bloc : [Terrain] × int × int → Bloc
  pre bloc(T, x,y) require 0 ≤ x ≤ largeur ∧ 0 ≤ y ≤ profondeur

Constructors :

  init : int × int × int → [Terrain]
  pre init(largeur, hauteur, prof) require largeur > 50 ∧ hauteur > 100 ∧ prof > 50 ∧
    largeur%50=0 ∧ profondeur%50=0

Operators :

  modifier_bloc : [Terrain] × int × int × Bloc → [Terrain]
  pre bloc(T, x, y, b) require 0 ≤ x ≤ largeur ∧ 0 ≤ y ≤ profondeur ∧ b ≠ null

Observations :

  [Invariants]

  [init]
```

```

largeur(init(l, h, p)) = l
hauteur(init(l, h, p)) = h
profondeur(init(l, h, p)) = p
bloc(init(l, h, p), x, y) ≠ NULL

```

```

[modifier_bloc]
bloc(modifier_bloc(T, x, y, b), x, y) = b

```

0.6 Moteur de jeu

```

service : MoteurJeu
use : GestionCombat
types : boolean, int, enum RESULTAT{DEUXGAGNANTS, RYANGAGNANT, ALEXGAGNANT, SLICKGAGNANT,
    NULLE},
    enum COMMANDE{RIEN, GAUCHE, DROITE, BAS, HAUT, FRAPPE, SAUT, SAUTHAUT, SAUTDROIT,
    SAUTGAUCHE, SAUTBAS, RAMASSER, JETER}
Observers :
    estFini : [MoteurJeu] → boolean
    resultat : [MoteurJeu] → RESULTAT
    pre resultat(M) require estFini(M)
    combat : [MoteurJeu] → GestionCombat
    pasCourant : [MoteurJeu] → int
Constructors :
    init : ∅ → [MoteurJeu]
Operators :
    pasJeu : [MoteurJeu] × COMMANDE × COMMANDE → [MoteurJeu]
    pre pasJeu(M, comAlex, comRyan) require ¬estFini(M)
Observations :
    [Invariants]

```

```

estFini(M)  $\stackrel{min}{=}$  (Personnage :: est_vaincu(GestionCombat :: alex(combat(M)))
    ∧ Personnage :: est_vaincu(GestionCombat :: ryan(combat(M)))
    ∨ Gangster :: est_vaincu(GestionCombat :: slick(combat(M)))

```

$$\text{resultat}(M) \stackrel{min}{=} \left\{ \begin{array}{ll} \text{ALEXGAGNANT} & \begin{array}{l} \text{si } \neg \text{Personnage} :: \text{est_vaincu}(\text{GestionCombat} :: \text{alex}(\text{combat}(M))) \\ \wedge \text{Gangster} :: \text{est_vaincu}(\text{GestionCombat} :: \text{slick}(\text{combat}(M))) \\ \wedge \text{Personnage} :: \text{est_vaincu}(\text{GestionCombat} :: \text{ryan}(\text{combat}(M))) \end{array} \\ \text{RYANGAGNANT} & \begin{array}{l} \text{si } \neg \text{Personnage} :: \text{est_vaincu}(\text{GestionCombat} :: \text{ryan}(\text{combat}(M))) \\ \wedge \text{Gangster} :: \text{est_vaincu}(\text{GestionCombat} :: \text{slick}(\text{combat}(M))) \\ \wedge \text{Personnage} :: \text{est_vaincu}(\text{GestionCombat} :: \text{alex}(\text{combat}(M))) \end{array} \\ \text{DEUXGAGNANTS} & \begin{array}{l} \text{si } \neg \text{Personnage} :: \text{est_vaincu}(\text{GestionCombat} :: \text{ryan}(\text{combat}(M))) \\ \wedge \text{Gangster} :: \text{est_vaincu}(\text{GestionCombat} :: \text{slick}(\text{combat}(M))) \\ \wedge \neg \text{Personnage} :: \text{est_vaincu}(\text{GestionCombat} :: \text{alex}(\text{combat}(M))) \end{array} \\ \text{SLICKGAGNANT} & \begin{array}{l} \text{si } \text{Personnage} :: \text{est_vaincu}(\text{GestionCombat} :: \text{ryan}(\text{combat}(M))) \\ \wedge \neg \text{Gangster} :: \text{est_vaincu}(\text{GestionCombat} :: \text{slick}(\text{combat}(M))) \\ \wedge \text{Personnage} :: \text{est_vaincu}(\text{GestionCombat} :: \text{alex}(\text{combat}(M))) \end{array} \\ \text{NULLE} & \text{sinon} \end{array} \right.$$

```

[init]
combat(init()) = GestionCombat :: init()
[pasJeu]
combat(pasJeu(M, cA, cR)) = GestionCombat :: gerer(combat(M), cA, cR)
pasCourant(pasJeu(M, cA, cR)) = pasCourant(M) + 1

```

0.7 GestionCombat

```
service : GestionCombat
use : Terrain, Personnage, Gangster
types : string, boolean, enum COMMANDE{RIEN, GAUCHE, DROITE, BAS, HAUT, FRAPPER, SAUT, SAUTHAUT, SAUTDROITE, SAUTGAUCHE,
SAUTBAS, RAMASSER, JETER}

Observers :
  terrain : [GestionCombat] → Terrain
  alex : [GestionCombat] → Personnage
  ryan : [GestionCombat] → Personnage
  slick : [GestionCombat] → Gangster
  gangsters : [GestionCombat] → List<Gangster>

  actionGangster : [GestionCombat] × Gangster → COMMANDE
    pre actionGangster (G, gang) require ¬Gangster::est_vaincu(gang)

  estGele : [GestionCombat] × Personnage → boolean
    pre estGele(G, perso) require perso = alex(G) ∨ perso = ryan(G) ∨ perso = slick(G) ∨ perso ∈ gangsters(G)

  estFrappe : [GestionCombat] × Personnage → boolean
    pre estFrappe(G, perso) require perso = alex(G) ∨ perso = ryan(G) ∨ perso = slick(G) ∨ perso ∈ gangsters(G)

  estVisible : [GestionCombat] × Personnage → boolean
    pre estVisible(G, perso) require perso = alex(G) ∨ perso = ryan(G) ∨ perso = slick(G) ∨ perso ∈ gangsters(G)

  posX : [GestionCombat] × Personnage → int
    pre posX(G, perso) require perso = alex(G) ∨ perso = ryan(G) ∨ perso = slick(G) ∨ perso ∈ gangsters(G)

  posY : [GestionCombat] × Personnage → int
    pre posY(G, perso) require perso = alex(G) ∨ perso = ryan(G) ∨ perso = slick(G) ∨ perso ∈ gangsters(G)

  posZ : [GestionCombat] × Personnage → int
    pre posZ(G, perso) require perso = alex(G) ∨ perso = ryan(G) ∨ perso = slick(G) ∨ perso ∈ gangsters(G)

  collisionDroite : [GestionCombat] × Personnage × Gangster → boolean
    pre collisionDroite(G, persol, perso2) require
      (persol = alex(G) ∨ persol = ryan(G)) ∧ (perso2 = slick(G) ∨ perso2 ∈ gangsters(G))

  collisionGauche : [GestionCombat] × Personnage × Gangster → boolean
    pre collisionGauche(G, persol, perso2) require
      (persol = alex(G) ∨ persol = ryan(G)) ∧ (perso2 = slick(G) ∨ perso2 ∈ gangsters(G))
```



```

collisionDevant : [GestionCombat] × Personnage × Gangster → boolean
pre collisionDevant(G, persol, perso2) require
(persol = alex(G) ∨ persol = ryan(G)) ∧ (perso2 = slick(G) ∨ perso2 ∈ gangsters(G))

collisionDerriere : [GestionCombat] × Personnage × Gangster → boolean
pre collisionDerriere(G, persol, perso2) require
(persol = alex(G) ∨ persol = ryan(G)) ∧ (perso2 = slick(G) ∨ perso2 ∈ gangsters(G))

collisionDessus : [GestionCombat] × Personnage × Gangster → boolean
pre collisionDessus(G, persol, perso2) require
(persol = alex(G) ∨ persol = ryan(G)) ∧ (perso2 = slick(G) ∨ perso2 ∈ gangsters(G))

collisionDessous : [GestionCombat] × Personnage × Gangster → boolean
pre collisionDessous(G, persol, perso2) require
(persol = alex(G) ∨ persol = ryan(G)) ∧ (perso2 = slick(G) ∨ perso2 ∈ gangsters(G))

collision : [GestionCombat] × Personnage × Gangster → boolean
pre collision(G, persol, perso2) require
(persol = alex(G) ∨ persol = ryan(G)) ∧ (perso2 = slick(G) ∨ perso2 ∈ gangsters(G))

```

Constructors :

```
init : ∅ → [GestionCombat]
```

Operators :

```
gerer : [GestionCombat] × COMMANDE × COMMANDE → [GestionCombat]
```

Observations :

[Invariants]

```
0 <= posX(G, s) <= Terrain::largeur(terrain)
```

```
0 <= posY(G, s) <= Terrain::profondeur(terrain)
```

```
0 <= posZ(G, s) <= Terrain::hauteur(terrain)
```

```
collisionDroite(G, p1, p2)  $\stackrel{min}{\equiv}$  ( -d ≤ posX(G, p1) - posX(G, p2) ≤ d+1) ∧ ( d = Personnage::largeur(p1)/2 + d = Personnage::largeur(p2)/2 )
```

```
collisionGauche(G, p1, p2)  $\stackrel{min}{\equiv}$  ( -d ≤ posX(G, p2) - posX(G, p1) ≤ d+1) ∧ ( d = Personnage::largeur(p1)/2 + d = Personnage::largeur(p2)/2 )
```

```
collisionDevant(G, p1, p2)  $\stackrel{min}{\equiv}$  ( -d ≤ posY(G, p1) - posY(G, p2) ≤ d+1) ∧ ( d = Personnage::profondeur(p1)/2 + d = Personnage::profondeur(p2)/2 )
```

```
collisionDerriere(G, p1, p2)  $\stackrel{min}{\equiv}$  ( -d ≤ posY(G, p2) - posY(G, p1) ≤ d+1) ∧ ( d = Personnage::profondeur(p1)/2 + d =
```

Personnage :: profondeur (p2)/2)

collisionDessous (G,p1,p2) $\stackrel{min}{=}$ ($-d \leq \text{posZ}(G,p1) - \text{posZ}(G,p2) \leq d+1$) \wedge ($d = \text{Personnage} :: \text{hauteur}(p1)/2 + d = \text{Personnage} :: \text{hauteur}(p2)/2$)

collisionDessus (G,p1,p2) $\stackrel{min}{=}$ ($-d \leq \text{posZ}(G,p2) - \text{posZ}(G,p1) \leq d+1$) \wedge ($d = \text{Personnage} :: \text{hauteur}(p1)/2 + d = \text{Personnage} :: \text{hauteur}(p2)/2$)

collision (G,p1,p2) $\stackrel{min}{=}$ estVisible (p1) \wedge estVisible (p2)
 \wedge collisionDroite (G,p1,p2) \wedge collisionGauche (G,p1,p2)
 \wedge collisionDevant (G,p1,p2) \wedge collisionDerriere (G,p1,p2)
 \wedge collisionDessous (G,p1,p2) \wedge collisionDessus (G,p1,p2)

actionGangster (G,g) = RIEN si estGele (G,g) \vee est_vaincu (G,g) \forall g \in gangsters (G)

[init]

terrain (init ()) = Terrain :: init (1000,1000,1000)

alex (init ()) = Personnage :: init ("Alex" ,30,30,30,100,100,0)

ryan (init ()) = Personnage :: init ("Ryan" ,30,30,30,100,100,0)

slick (init ()) = Gangster :: init ("Slick" ,50,50,50,100,100)

gangsters (init ()) = {g = Personnage :: init ("noname" ,20,20,20,10,50)} , \forall g \in gangsters (G)

actionGangster (G,g) = RIEN \forall g \in gangsters (G)

estGele (init () , s) = false

collisionGauche (init () ,p1,p2) = false

collisionDroite (init () ,p1,p2) = false

collisionDevant (init () ,p1,p2) = false

collisionDerriere (init () ,p1,p2) = false

collisionDessous (init () ,p1,p2) = false

collisionDessus (init () ,p1,p2) = false

collision (init () ,p1,p2) = false

```

estFrappe(init(), s) = false

posX(init(), alex(G)) < 50

posX(init(), slick(G)) > Terrain::largeur(terrain(G)) - 50

posX(init(), ryan(G)) < 50

posZ(init(), p) = 0

posY(init(), perso) = random

Bloc::type(Terrain: bloc(terrain(G), posX(init(), g), posY(init(), g), posZ(init(), g))) = VIDE  $\forall g \in \text{gangsters}(G)$ 

Bloc::type(Terrain: bloc(terrain(G), posX(init(), slick(G)), posY(init(), slick(G)), posZ(init(), slick(G)))) = VIDE

Bloc::type(Terrain: bloc(terrain(G), posX(init(), alex(G)), posY(init(), alex(G)), posZ(init(), alex(G))))  $\neq$  FOSSE

Bloc::type(Terrain: bloc(terrain(G), posX(init(), ryan(G)), posY(init(), ryan(G)), posZ(init(), ryan(G))))  $\neq$  FOSSE

[gerer]

posX(G, gerer(G, cA, cR), alex(G)) =

$$\begin{cases} \min(\text{posX}(G, \text{alex}(G)) + 10, \text{Terrain.largeur}(\text{terrain}(G)) - \text{Personnage} : \text{largeur}(\text{alex}(G))) \\ \text{si } cA = \text{DROITE} \vee cA = \text{SAUTDROITE} \wedge \neg \text{Personnage} : \text{est\_vaincu}(\text{alex}(G)) \wedge \neg \text{estGele}(G, \text{alex}(G)) \\ \wedge \neg \text{collisionDroite}(\text{alex}(G), p) \forall p \in \text{gangster} \vee p = \text{slick}(G) \\ \max(\text{posX}(G, \text{alex}(G)) - 10, 0) \\ \text{si } cA = \text{GAUCHE} \vee cA = \text{SAUTGAUCHE} \wedge \neg \text{Personnage} : \text{est\_vaincu}(\text{alex}(G)) \wedge \neg \text{estGele}(G, \text{alex}(G)) \\ \wedge \neg \text{collisionGauche}(\text{alex}(G), p) \forall p \in \text{gangster} \vee p = \text{slick}(G) \\ \text{posX}(G, \text{alex}(G)) \text{ sinon} \end{cases}$$

% VIRER LES CONDITIONS DE COLLISION (affreuses)

posY(G, gerer(G, cA, cR), alex(G)) =

$$\begin{cases} \min(\text{posY}(G, \text{alex}(G)) + 10, \text{Terrain.profondeur}(\text{terrain}(G)) - \text{Personnage} : \text{profondeur}(\text{alex}(G))) \\ \text{si } cA = \text{HAUT} \vee cA = \text{SAUTHAUT} \wedge \neg \text{Personnage} : \text{est\_vaincu}(\text{alex}(G)) \wedge \neg \text{estGele}(G, \text{alex}(G)) \wedge \neg \text{collisionDerriere}(\text{alex}(G), p) \forall p \in \text{gangster} \vee p = \text{slick}(G) \\ \max(\text{posY}(G, \text{alex}(G)) - 10, 0) \\ \text{si } cA = \text{BAS} \vee cA = \text{SAUTBAS} \wedge \neg \text{Personnage} : \text{est\_vaincu}(\text{alex}(G)) \wedge \neg \text{estGele}(G, \text{alex}(G)) \wedge \neg \text{collisionDevant}(\text{alex}(G), p) \forall p \in \text{gangster} \vee p = \text{slick}(G) \\ \text{posY}(G, \text{alex}(G)) \text{ sinon} \end{cases}$$


posZ(gerer(G, cA, cR), alex(G)) =

$$\begin{cases} 100 \\ \text{si } cA = \text{SAUT} \vee cA = \text{SAUTBAS} \vee cA = \text{SAUTHAUT} \vee cA = \text{SAUTDROITE} \vee cA = \text{SAUTGAUCHE} \\ \wedge \neg \text{Personnage} : \text{est\_vaincu}(\text{alex}(G)) \wedge \neg \text{estGele}(G, \text{alex}(G)) \wedge \neg \text{collisionDessus}(\text{alex}(G), p) \forall p \in \text{gangster} \vee p = \text{slick}(G) \\ \text{pos}(G, \text{alex}(G)) \quad \text{si estGele}(G, \text{alex}(G)) \\ 0 \quad \text{Sinon} \end{cases}$$


posX(G, gerer(G, cA, cR), ryan(G)) =

```

$$\begin{cases} \min(\text{posX}(G, \text{ryan}(G)) + 10, \text{Terrain} : \text{largeur}(\text{terrain}(G)) - \text{Personnage} : \text{largeur}(\text{ryan}(G))) \\ \text{si } cA = \text{DROITE} \vee cA = \text{SAUTDROITE} \wedge \neg \text{Personnage} : \text{est_vaincu}(\text{ryan}(G)) \wedge \neg \text{estGele}(G, \text{ryan}(G)) \\ \wedge \neg \text{collisionDroite}(\text{ryan}(G), p) \forall p \in \text{gangster} \vee p = \text{slick}(G) \\ \max(\text{posX}(G, \text{ryan}(G)) - 10, 0) \\ \text{si } cA = \text{GAUCHE} \vee cA = \text{SAUTGAUCHE} \wedge \neg \text{Personnage} : \text{est_vaincu}(\text{ryan}(G)) \wedge \neg \text{estGele}(G, \text{ryan}(G)) \\ \wedge \neg \text{collisionGauche}(\text{ryan}(G), p) \forall p \in \text{gangster} \vee p = \text{slick}(G) \\ \text{posX}(G, \text{ryan}(G)) \text{ sinon} \end{cases} \\
\text{posY}(G, \text{gerer}(G, cA, cR), \text{ryan}(G)) = \\
\begin{cases} \min(\text{posY}(G, \text{ryan}(G)) + 10, \text{Terrain} : \text{profondeur}(\text{terrain}(G)) - \text{Personnage} : \text{profondeur}(\text{ryan}(G))) \\ \text{si } cA = \text{HAUT} \vee cA = \text{SAUTHAUT} \wedge \neg \text{Personnage} : \text{est_vaincu}(\text{ryan}(G)) \wedge \neg \text{estGele}(G, \text{ryan}(G)) \wedge \neg \text{collisionDerriere}(\text{ryan}(G), p) \forall p \in \text{gangster} \vee p = \text{slick}(G) \\ \max(\text{posY}(G, \text{ryan}(G)) - 10, 0) \\ \text{si } cA = \text{BAS} \vee cA = \text{SAUTBAS} \wedge \neg \text{Personnage} : \text{est_vaincu}(\text{ryan}(G)) \wedge \neg \text{estGele}(G, \text{ryan}(G)) \wedge \neg \text{collisionDevant}(\text{ryan}(G), p) \forall p \in \text{gangster} \vee p = \text{slick}(G) \\ \text{posY}(G, \text{ryan}(G)) \text{ sinon} \end{cases} \\
\text{posZ}(\text{gerer}(G, cA, cR), \text{ryan}(G)) = \\
\begin{cases} 100 \\ \text{si } cA = \text{SAUT} \vee cA = \text{SAUTBAS} \vee cA = \text{SAUTHAUT} \vee cA = \text{SAUTDROITE} \vee cA = \text{SAUTGAUCHE} \\ \wedge \neg \text{Personnage} : \text{est_vaincu}(\text{ryan}(G)) \wedge \neg \text{estGele}(G, \text{ryan}(G)) \wedge \neg \text{collisionDessus}(\text{ryan}(G), p) \forall p \in \text{gangster} \vee p = \text{slick}(G) \\ \text{pos}(G, \text{ryan}(G)) \\ 0 \end{cases} \text{Sinon} \\
\text{posX}(G, \text{gerer}(G, cA, cR), \text{slick}(G)) = \\
\begin{cases} \min(\text{posX}(G, \text{slick}(G)) + 10, \text{Terrain} : \text{largeur}(\text{terrain}(G)) - \text{Personnage} : \text{largeur}(\text{slick}(G))) \\ \text{si } \text{actionGangster}(G, \text{slick}(G)) = \text{DROITE} \vee \text{actionGangster}(G, \text{slick}(G)) = \text{SAUTDROITE} \wedge \neg \text{Personnage} : \text{est_vaincu}(\text{slick}(G)) \wedge \neg \text{estGele}(G, \text{slick}(G)) \\ \wedge \neg \text{collisionDroite}(\text{slick}(G), p) \forall p \in \text{gangster} \vee p = \text{slick}(G) \\ \max(\text{posX}(G, \text{slick}(G)) - 10, 0) \\ \text{si } \text{actionGangster}(G, \text{slick}(G)) = \text{GAUCHE} \vee \text{actionGangster}(G, \text{slick}(G)) = \text{SAUTGAUCHE} \wedge \neg \text{Personnage} : \text{est_vaincu}(\text{slick}(G)) \wedge \neg \text{estGele}(G, \text{slick}(G)) \\ \wedge \neg \text{collisionGauche}(\text{slick}(G), p) \forall p \in \text{gangster} \vee p = \text{slick}(G) \\ \text{posX}(G, \text{slick}(G)) \text{ sinon} \end{cases} \\
\text{posY}(G, \text{gerer}(G, cA, cR), \text{slick}(G)) = \\
\begin{cases} \min(\text{posY}(G, \text{slick}(G)) + 10, \text{Terrain} : \text{profondeur}(\text{terrain}(G)) - \text{Personnage} : \text{profondeur}(\text{slick}(G))) \\ \text{si } \text{actionGangster}(G, \text{slick}(G)) = \text{HAUT} \vee \text{actionGangster}(G, \text{slick}(G)) = \text{SAUTHAUT} \wedge \neg \text{Personnage} : \text{est_vaincu}(\text{slick}(G)) \wedge \neg \text{estGele}(G, \text{slick}(G)) \\ \wedge \neg \text{collisionDerriere}(\text{slick}(G), p) \forall p \in \text{gangster} \vee p = \text{slick}(G) \\ \max(\text{posY}(G, \text{slick}(G)) - 10, 0) \\ \text{si } \text{actionGangster}(G, \text{slick}(G)) = \text{BAS} \vee \text{actionGangster}(G, \text{slick}(G)) = \text{SAUTBAS} \wedge \neg \text{Personnage} : \text{est_vaincu}(\text{slick}(G)) \wedge \neg \text{estGele}(G, \text{slick}(G)) \\ \wedge \neg \text{collisionDevant}(\text{slick}(G), p) \forall p \in \text{gangster} \vee p = \text{slick}(G) \\ \text{posY}(G, \text{slick}(G)) \text{ sinon} \end{cases} \\
\text{posZ}(\text{gerer}(G, cA, cR), \text{slick}(G)) = \\
\begin{cases} 100 \\ \text{si } \text{actionGangster}(G, \text{slick}(G)) = \text{SAUT} \vee \text{actionGangster}(G, \text{slick}(G)) = \text{SAUTBAS} \vee \text{actionGangster}(G, \text{slick}(G)) = \text{SAUTHAUT} \\ \vee \text{actionGangster}(G, \text{slick}(G)) = \text{SAUTDROITE} \vee \text{actionGangster}(G, \text{slick}(G)) = \text{SAUTGAUCHE} \\ \wedge \neg \text{Personnage} : \text{est_vaincu}(\text{slick}(G)) \wedge \neg \text{estGele}(G, \text{slick}(G)) \wedge \neg \text{collisionDessus}(\text{slick}(G), p) \forall p \in \text{gangster} \vee p = \text{slick}(G) \\ \text{pos}(G, \text{slick}(G)) \text{ si } \text{estGele}(G, \text{slick}(G)) \\ 0 \end{cases} \text{Sinon}
\end{cases}$$

$$\begin{aligned}
& \text{posX}(G, \text{gerer}(G, cA, cR), \text{gangsters}(G)) = \{ \text{g} = \\
& \quad \left\{ \begin{array}{l} \min(\text{posX}(G, g) + 10, \text{Terrain} : \text{largeur}(\text{terrain}(G))) - \text{Personnage} : \text{largeur}(g) \\ \text{si actionGangster}(G, g) = \text{DROITE} \vee \text{actionGangster}(G, g) = \text{SAUTDROITE} \wedge \neg \text{Personnage} : \text{est_vaincu}(g) \wedge \neg \text{estGele}(G, g) \\ \wedge \neg \text{collisionDroite}(g, p) \vee p \in \text{gangster} \vee p = g \\ \max(\text{posX}(G, g) - 10, 0) \\ \text{si actionGangster}(G, g) = \text{GAUCHE} \vee \text{actionGangster}(G, g) = \text{SAUTGAUCHE} \wedge \neg \text{Personnage} : \text{est_vaincu}(g) \wedge \neg \text{estGele}(G, g) \\ \wedge \neg \text{collisionGauche}(g, p) \vee p \in \text{gangster} \vee p = g \\ \text{posX}(G, g) \text{ sinon} \end{array} \right. \\
& \quad \} \vee g \in \text{gangsters}(G) \\
\\
& \text{posY}(G, \text{gerer}(G, cA, cR), \text{gangsters}(G)) = \{ \text{g} = \\
& \quad \left\{ \begin{array}{l} \min(\text{posY}(G, g) + 10, \text{Terrain} : \text{profondeur}(\text{terrain}(G))) - \text{Personnage} : \text{profondeur}(g) \\ \text{si actionGangster}(G, g) = \text{HAUT} \vee \text{actionGangster}(G, g) = \text{SAUTHAUT} \wedge \neg \text{Personnage} : \text{est_vaincu}(g) \wedge \neg \text{estGele}(G, g) \wedge \neg \text{collisionDerriere}(g, p) \vee p \in \text{gangster} \vee p = \\ \max(\text{posY}(G, g) - 10, 0) \\ \text{si actionGangster}(G, g) = \text{BAS} \vee \text{actionGangster}(G, g) = \text{SAUTBAS} \wedge \neg \text{Personnage} : \text{est_vaincu}(g) \wedge \neg \text{estGele}(G, g) \wedge \neg \text{collisionDevant}(g, p) \vee p \in \text{gangster} \vee p = g \\ \text{posY}(G, g) \text{ sinon} \end{array} \right. \\
& \quad \} \vee g \in \text{gangsters}(G) \\
\\
& \text{posZ}(\text{gerer}(G, cA, cR), \text{gangsters}(G)) = \{ \text{g} = \\
& \quad \left\{ \begin{array}{l} 100 \text{ si actionGangster}(G, g) = \text{SAUT} \vee \text{actionGangster}(G, g) = \text{SAUTBAS} \vee \text{actionGangster}(G, g) = \text{SAUTHAUT} \vee \text{actionGangster}(G, g) = \text{SAUTDROITE} \\ \vee \text{actionGangster}(G, g) = \text{SAUTGAUCHE} \wedge \neg \text{Personnage} : \text{est_vaincu}(g) \wedge \neg \text{estGele}(G, g) \wedge \neg \text{collisionDessus}(g, p) \vee p \in \text{gangster} \vee p = g \\ \text{pos}(G, g) \text{ si estGele}(G, g) \\ 0 \text{ Sinon} \end{array} \right. \\
& \quad \} \vee g \in \text{gangsters}(G) \\
\\
& \text{alex}(\text{gerer}(G, cA, cR)) = \\
& \quad \left\{ \begin{array}{l} - \text{Personnage} : \text{jeter}(\text{alex}(G)) \\ \text{si } cA = \text{JETER} \wedge \neg \text{Personnage} : \text{est_vaincu}(\text{alex}(G)) \wedge \neg \text{estGele}(G, \text{alex}(G)) \wedge \text{Bloc} : \text{type}(\text{Terrain} : \text{bloc}(\text{terrain}(G), \text{posX}(G, \text{alex}(G))), \text{posY}(G, \text{alex}(G))) = \text{VIDE} \\ - \text{Personnage} : \text{ramasser_objet}(\text{alex}(G), \text{Bloc} : \text{objet}(\text{Terrain} : \text{bloc}(\text{terrain}(G), \text{posX}(G, \text{alex}(G))), \text{posY}(G, \text{alex}(G)))) \\ \text{si } cA = \text{RAMASSER} \wedge \text{posZ}(G, \text{alex}(G)) = 0 \wedge \neg \text{Personnage} : \text{est_vaincu}(\text{alex}(G)) \wedge \neg \text{estGele}(G, \text{alex}(G)) \\ - \text{Personnage} : \text{ramasser_perso}(\text{alex}(G), p) \\ \text{si collision}(G, \text{alex}(G), p) \wedge cA = \text{RAMASSER} \wedge \neg \text{Personnage} : \text{est_vaincu}(\text{alex}(G)) \wedge \neg \text{estGele}(G, \text{alex}(G)) \\ - \text{Personnage} : \text{ramasser_argent}(\text{alex}(G), \text{Bloc} : \text{objet}(\text{Terrain} : \text{bloc}(\text{terrain}(G), \text{posX}(G, \text{alex}(G))), \text{posY}(G, \text{alex}(G)))) \\ \text{si } cA = \text{RAMASSER} \wedge \text{posZ}(G, \text{alex}(G)) = 0 \wedge \neg \text{Personnage} : \text{est_vaincu}(\text{alex}(G)) \wedge \neg \text{estGele}(G, \text{alex}(G)) \\ - \text{Personnage} : \text{retrait_vie}(\text{alex}(G), \text{Personnage} : \text{force}(p)) \\ \text{si collision}(G, \text{alex}(G), p) \wedge \text{actionGangster}(G, p) = \text{FRAPPER} \wedge \neg \text{Personnage} : \text{est_vaincu}(\text{alex}(G)) \wedge \neg \text{estGele}(G, \text{alex}(G)) \\ - \text{Personnage} : \text{retrait_vie}(\text{alex}(G), \text{Personnage} : \text{points_de_vie}(\text{alex}(G))) \\ \text{si Bloc} : \text{type}(\text{Terrain} : \text{bloc}(\text{terrain}(G), \text{posX}(G, \text{alex}(G))), \text{posY}(G, \text{alex}(G))) = \text{FOSSE} \wedge \neg \text{Personnage} : \text{est_vaincu}(\text{alex}(G)) \wedge \neg \text{estGele}(G, \text{alex}(G)) \\ - \text{alex}(G) \text{ Sinon} \end{array} \right.
\end{aligned}$$

```

ryan ( gerer ( G, cA, cR ) ) =
{
- Personnage : jeter(ryan(G))
- si cR = JETER  $\wedge$   $\neg$ Personnage : est_vaincu(ryan(G))  $\wedge$   $\neg$ estGele(G,ryan(G))  $\wedge$  Bloc : :type(Terrain : :bloc(Terrain(G),posX(G,ryan(G)),posY(G,ryan(G))))=VIDE
- Personnage : ramasser_objet(ryan(G), Bloc : :objet(Terrain : :bloc(Terrain(G),posX(G,ryan(G)),posY(G,ryan(G))))
- si cR = RAMASSER  $\wedge$  posZ(G,ryan(G))=0  $\wedge$   $\neg$ Personnage : est_vaincu(ryan(G))  $\wedge$   $\neg$ estGele(G,ryan(G))
- Personnage : ramasser_perso(ryan(G), p)
- si collision(G,ryan(G), p)  $\wedge$  cR = RAMASSER  $\wedge$   $\neg$ Personnage : est_vaincu(ryan(G))  $\wedge$   $\neg$ estGele(G,ryan(G))
- Personnage : ramasser_argent(ryan(G), Bloc : :objet(Terrain : :bloc(Terrain(G),posX(G,ryan(G)),posY(G,ryan(G))))
- si cR = RAMASSER  $\wedge$  posZ(G,ryan(G))=0  $\wedge$   $\neg$ Personnage : est_vaincu(ryan(G))  $\wedge$   $\neg$ estGele(G,ryan(G))
- Personnage : retrait_vie(ryan(G), Personnage : :force(p))
- si collision(G,ryan(G),p)  $\wedge$  actionGangster(G,p) = FRAPPER  $\wedge$   $\neg$ Personnage : est_vaincu(ryan(G))  $\wedge$   $\neg$ estGele(G,ryan(G))
- Personnage : retrait_vie(ryan(G), Personnage : :points_de_vie(ryan(G))
- si Bloc : :type(Terrain : :bloc(Terrain(G),posX(G,ryan(G)),posY(G,ryan(G)))) = FOSSE  $\wedge$   $\neg$ Personnage : est_vaincu(ryan(G))  $\wedge$   $\neg$ estGele(G,ryan(G))
- ryan(G) Sinon
}

gangsters ( gerer ( G, cA, cR ) ) = { g =
- Gangster : jeter(g)
- si actionGangster(G,g) = JETER  $\wedge$   $\neg$ Personnage : est_vaincu(g)  $\wedge$   $\neg$ estGele(G,g)  $\wedge$  Bloc : :type(Terrain : :bloc(Terrain(G),posX(G,g),posY(G,g))))=VIDE
- Gangster : ramasser_objet(g, Bloc : :objet(Terrain : :bloc(Terrain(G),posX(G,g),posY(G,g))))
- si actionGangster(G,g) = RAMASSER  $\wedge$  posZ(G,g)=0  $\wedge$   $\neg$ Gangster : est_vaincu(g)  $\wedge$   $\neg$ estGele(G,g)
- Gangster : retrait_vie(g, Personnage : :force(p))
- si collision(G,alex(G),g)  $\wedge$  cA = FRAPPER  $\wedge$   $\neg$ Gangster : est_vaincu(g)  $\wedge$   $\neg$ estGele(G,g)
- Gangster : retrait_vie(g, Personnage : :force(p))
- si collision(G,ryan(G),g)  $\wedge$  cR = FRAPPER  $\wedge$   $\neg$ Gangster : est_vaincu(g)  $\wedge$   $\neg$ estGele(G,g)
- Gangster : retrait_vie(g, Gangster : :points_de_vie(g)
- si Bloc : :type(Terrain : :bloc(Terrain(G),posX(G,g),posY(G,g))) = FOSSE  $\wedge$   $\neg$ Gangster : est_vaincu(g)  $\wedge$   $\neg$ estGele(G,g)
- g Sinon
}
 $\forall g \in$  gangsters (G)

slick ( gerer ( G, cA, cR ) ) =
{
- Gangster : jeter(slick(G))
- si actionGangster(G,slick(G)) = JETER  $\wedge$   $\neg$ Personnage : est_vaincu(slick(G))  $\wedge$   $\neg$ estGele(G,slick(G))  $\wedge$ 
Bloc : :type(Terrain : :bloc(Terrain(G),posX(G,slick(G)),posY(G,slick(G))))=VIDE
- Gangster : ramasser_objet(g, Bloc : :objet(Terrain : :bloc(Terrain(G),posX(G,slick(G)),posY(G,slick(G))))
- si actionGangster(G,g) = RAMASSER  $\wedge$  posZ(G,slick(G))=0  $\wedge$   $\neg$ Gangster : est_vaincu(slick(G))  $\wedge$   $\neg$ estGele(G,slick(G))
- Gangster : retrait_vie(slick(G), Personnage : :force(p))
- si collision(G,alex(G),slick(G))  $\wedge$  cA = FRAPPER  $\wedge$   $\neg$ Gangster : est_vaincu(slick(G))  $\wedge$   $\neg$ estGele(G,g)
- Gangster : retrait_vie(slick(G), Personnage : :force(p))
- si collision(G,ryan(G),slick(G))  $\wedge$  cR = FRAPPER  $\wedge$   $\neg$ Gangster : est_vaincu(slick(G))  $\wedge$   $\neg$ estGele(G,slick(G))
- Gangster : retrait_vie(slick(G), Gangster : :points_de_vie(slick(G))
- si Bloc : :type(Terrain : :bloc(Terrain(G),posX(G,slick(G)),posY(G,slick(G)))) = FOSSE  $\wedge$   $\neg$ Gangster : est_vaincu(slick(G))  $\wedge$   $\neg$ estGele(G,slick(G))
- slick(G) Sinon
}

```

estVisible (gerer (G,cA,cR) , alex (G)) =

$$\begin{cases} \text{true} & \text{si } \neg \text{Personnage} : \text{est_vaincu}(\text{alex}(\text{G})) \\ \text{false} & \text{sinon} \end{cases}$$

estVisible (gerer (G,cA,cR) , ryan (G)) =

$$\begin{cases} \text{true} & \text{si } \neg \text{Personnage} : \text{est_vaincu}(\text{ryan}(\text{G})) \\ \text{false} & \text{sinon} \end{cases}$$

estVisible (gerer (G,cA,cR) , g) =

$$\begin{cases} \text{true} & \text{si } \neg \text{Gangster} : \text{est_vaincu}(\text{g}) \\ \text{true} & \text{si } \text{Personnage} : \text{perso_equipe}(\text{alex}(\text{G})) = \text{g} \wedge \text{cA} = \text{JETER} \\ \text{false} & \text{si } \text{collision}(\text{G},\text{alex}(\text{G}),\text{g}) \wedge \text{cA} = \text{RAMASSER} \\ \text{false} & \text{sinon} \end{cases} \quad \forall \text{ g} \in \text{gangsters}(\text{G})$$

estVisible (gerer (G,cA,cR) , slick (G)) =

$$\begin{cases} \text{true} & \text{si } \neg \text{Gangster} : \text{est_vaincu}(\text{slick}(\text{G})) \\ \text{true} & \text{si } \text{Personnage} : \text{perso_equipe}(\text{alex}(\text{G})) = \text{slick}(\text{G}) \wedge \text{cA} = \text{JETER} \\ \text{false} & \text{si } \text{collision}(\text{G},\text{alex}(\text{G}),\text{slick}(\text{G})) \wedge \text{cA} = \text{RAMASSER} \\ \text{false} & \text{sinon} \end{cases}$$

13

posX (G, gerer (G,cA,cR) , p) =

$$\begin{cases} \text{posX}(\text{G},\text{alex}(\text{G})) & \text{si } \text{cA} = \text{JETER} \wedge \text{Personnage} : \text{perso_equipe}(\text{alex}(\text{G})) = \text{p} \wedge \neg \text{Personnage} : \text{est_vaincu}(\text{alex}(\text{G})) \\ \text{posX}(\text{G},\text{p}) & \text{sinon} \end{cases}$$

posY (G, gerer (G,cA,cR) , p) =

$$\begin{cases} \text{posY}(\text{G},\text{alex}(\text{G})) & \text{si } \text{cA} = \text{JETER} \wedge \text{Personnage} : \text{perso_equipe}(\text{alex}(\text{G})) = \text{p} \wedge \neg \text{Personnage} : \text{est_vaincu}(\text{alex}(\text{G})) \\ \text{posY}(\text{G},\text{p}) & \text{sinon} \end{cases}$$

posZ (gerer (G,cA,cR) , p) =

$$\begin{cases} 0 & \text{si } \text{cA} = \text{JETER} \wedge \text{Personnage} : \text{perso_equipe}(\text{alex}(\text{G})) = \text{p} \wedge \neg \text{Personnage} : \text{est_vaincu}(\text{alex}(\text{G})) \\ \text{posZ}(\text{G},\text{p}) & \text{sinon} \end{cases}$$

posX (G, gerer (G,cA,cR) , p) =

$$\begin{cases} \text{posX}(\text{G},\text{ryan}(\text{G})) & \text{si } \text{cA} = \text{JETER} \wedge \text{Personnage} : \text{perso_equipe}(\text{ryan}(\text{G})) = \text{p} \wedge \neg \text{Personnage} : \text{est_vaincu}(\text{ryan}(\text{G})) \\ \text{posX}(\text{G},\text{p}) & \text{sinon} \end{cases}$$

posY (G, gerer (G,cA,cR) , p) =

$$\begin{cases} \text{posY}(\text{G},\text{ryan}(\text{G})) & \text{si } \text{cA} = \text{JETER} \wedge \text{Personnage} : \text{perso_equipe}(\text{ryan}(\text{G})) = \text{p} \wedge \neg \text{Personnage} : \text{est_vaincu}(\text{ryan}(\text{G})) \\ \text{posY}(\text{G},\text{p}) & \text{sinon} \end{cases}$$

posZ (gerer (G,cA,cR) , p) =

$$\begin{cases} 0 & \text{si } \text{cA} = \text{JETER} \wedge \text{Personnage} : \text{perso_equipe}(\text{ryan}(\text{G})) = \text{p} \wedge \neg \text{Personnage} : \text{est_vaincu}(\text{ryan}(\text{G})) \\ \text{posZ}(\text{G},\text{p}) & \text{sinon} \end{cases}$$

Terrain :: bloc (terrain (gerer (G,cA,cR)) , posX (G, alex (G)) , posY (G, alex (G))) =

$$\left\{ \begin{array}{l}
- \text{Bloc} : \text{retirerObjet}(\text{Terrain} : \text{bloc}(\text{terrain}(G), \text{posX}(G, \text{alex}(G)), \text{posY}(G, \text{alex}(G)))) \\
\text{si } cA = \text{RAMASSER} \wedge \neg \text{Personnage} : \text{est_vaincu}(\text{alex}(G)) \wedge \neg \text{estGele}(G, \text{alex}(G)) \\
- \text{Bloc} : \text{poserObjet}(\text{Terrain} : \text{bloc}(\text{terrain}(G), \text{posX}(G, \text{alex}(G)), \text{posY}(G, \text{alex}(G))), \text{Personnage} : \text{objet_equipe}(\text{alex}(G))) \\
\text{si } cA = \text{JETER} \wedge \text{Personnage} : \text{est_equipe_objet}(\text{alex}(G)) = \text{true} \wedge \neg \text{Personnage} : \text{est_vaincu}(\text{alex}(G)) \wedge \neg \text{estGele}(G, \text{alex}(G)) \\
- \text{Terrain} : \text{bloc}(\text{terrain}(G), \text{posX}(G, \text{alex}(G)), \text{posY}(G, \text{alex}(G))) \text{ Sinon}
\end{array} \right.$$

$$\text{Terrain} :: \text{bloc}(\text{terrain}(\text{gerer}(G, cA, cR)), \text{posX}(G, \text{ryan}(G)), \text{posY}(G, \text{ryan}(G))) =
\left\{ \begin{array}{l}
- \text{Bloc} : \text{retirerObjet}(\text{Terrain} : \text{bloc}(\text{terrain}(G), \text{posX}(G, \text{ryan}(G)), \text{posY}(G, \text{ryan}(G)))) \\
\text{si } cR = \text{RAMASSER} \wedge \neg \text{Personnage} : \text{est_vaincu}(\text{ryan}(G)) \wedge \neg \text{estGele}(G, \text{ryan}(G)) \\
- \text{Bloc} : \text{poserObjet}(\text{Terrain} : \text{bloc}(\text{terrain}(G), \text{posX}(G, \text{ryan}(G)), \text{posY}(G, \text{ryan}(G))), \text{Personnage} : \text{objet_equipe}(\text{ryan}(G))) \\
\text{si } cR = \text{JETER} \wedge \text{Personnage} : \text{est_equipe_objet}(\text{ryan}(G)) = \text{true} \wedge \neg \text{Personnage} : \text{est_vaincu}(\text{ryan}(G)) \wedge \neg \text{estGele}(G, \text{ryan}(G)) \\
- \text{Terrain} : \text{bloc}(\text{terrain}(G), \text{posX}(G, \text{ryan}(G)), \text{posY}(G, \text{ryan}(G))) \text{ Sinon}
\end{array} \right.$$

$$\text{Terrain} :: \text{bloc}(\text{terrain}(\text{gerer}(G, cA, cR)), \text{posX}(G, \text{slick}(G)), \text{posY}(G, \text{slick}(G))) =
\left\{ \begin{array}{l}
- \text{Bloc} : \text{retirerObjet}(\text{Terrain} : \text{bloc}(\text{terrain}(G), \text{posX}(G, \text{slick}(G)), \text{posY}(G, \text{slick}(G)))) \\
\text{si } \text{actionGangster}(G, \text{slick}(G)) = \text{RAMASSER} \wedge \neg \text{Personnage} : \text{est_vaincu}(\text{slick}(G)) \wedge \neg \text{estGele}(G, \text{slick}(G)) \\
- \text{Bloc} : \text{poserObjet}(\text{Terrain} : \text{bloc}(\text{terrain}(G), \text{posX}(G, \text{slick}(G)), \text{posY}(G, \text{slick}(G))), \text{Personnage} : \text{objet_equipe}(\text{slick}(G))) \\
\text{si } \text{actionGangster}(G, \text{slick}(G)) = \text{JETER} \wedge \text{Personnage} : \text{est_equipe_objet}(\text{slick}(G)) = \text{true} \wedge \neg \text{Personnage} : \text{est_vaincu}(\text{slick}(G)) \wedge \neg \text{estGele}(G, \text{slick}(G)) \\
- \text{Terrain} : \text{bloc}(\text{terrain}(G), \text{posX}(G, \text{slick}(G)), \text{posY}(G, \text{slick}(G))) \text{ Sinon}
\end{array} \right.$$

$$\text{Terrain} :: \text{bloc}(\text{terrain}(\text{gerer}(G, cA, cR)), \text{posX}(G, g), \text{posY}(G, g)) =
\left\{ \begin{array}{l}
- \text{Bloc} : \text{retirerObjet}(\text{Terrain} : \text{bloc}(\text{terrain}(G), \text{posX}(G, g), \text{posY}(G, g))) \\
\text{si } \text{actionGangster}(G, g) = \text{RAMASSER} \wedge \neg \text{Personnage} : \text{est_vaincu}(g) \wedge \neg \text{estGele}(G, g) \\
- \text{Bloc} : \text{poserObjet}(\text{Terrain} : \text{bloc}(\text{terrain}(G), \text{posX}(G, g), \text{posY}(G, g))), \text{Personnage} : \text{objet_equipe}(g) \\
\text{si } \text{actionGangster}(G, g) = \text{JETER} \wedge \text{Personnage} : \text{est_equipe_objet}(g) = \text{true} \wedge \neg \text{Personnage} : \text{est_vaincu}(g) \wedge \neg \text{estGele}(G, g) \quad \forall g \in \text{gangsters}(G) \\
- \text{Bloc} : \text{poserObjet}(\text{Terrain} : \text{bloc}(\text{terrain}(G), \text{posX}(G, g), \text{posY}(G, g)), \text{Objet} : \text{init}(\text{"Recompense", 0, 1000})) \\
\text{si } \text{Gangster} : \text{est_vaincu}(g) \\
- \text{Terrain} : \text{bloc}(\text{terrain}(G), \text{posX}(G, g), \text{posY}(G, g)) \text{ Sinon}
\end{array} \right.$$

Tests

0.1 Le service Personnage

Cas de test : Personnage::testInitWorking

CI : $\text{nom} = \text{"Ryan"} \wedge l = 30 \wedge h = 30 \wedge p = 30 \wedge a = 10 \wedge v = 100 \wedge f = 100$

Operation : $P0 = \text{def init}(\text{nom}, l, h, p, f, v, a)$

Oracle :

$\text{nom}(P0) = \text{"Ryan"}$

$\text{largeur}(P0) = 30$

$\text{profondeur}(P0) = 30$

$\text{hauteur}(P0) = 30$

$\text{pointsDeVie}(P0) = 100$

$\text{force}(P0) = 100$

$\text{argent}(P0) = 10$

Cas de test : Personnage::testInitFailing

CI : $\text{nom} = \text{"Joe"} \wedge l = 30 \wedge h = 30 \wedge p = 30 \wedge a = 10 \wedge v = 100 \wedge f = 100$

Operation : $P0 = \text{def init}(\text{nom}, l, h, p, f, v, a)$

Oracle :

$\text{nom} \neq \text{"Alex"} \wedge \text{nom} \neq \text{"Ryan"}$

Une exception est levee

Cas de test : Personnage::testInitFailing

CI : $\text{nom} = \text{"Alex"} \wedge l = -5 \wedge h = 30 \wedge p = 30 \wedge a = 10 \wedge v = 100 \wedge f = 100$

Operation : $P0 = \text{def init}(\text{nom}, l, h, p, f, v, a)$

Oracle :

$l \leq 0$

Une exception est levee

Cas de test : Personnage::testInitFailing

CI : $\text{nom} = \text{"Alex"} \wedge l = 30 \wedge h = -5 \wedge p = 30 \wedge a = 10 \wedge v = 100 \wedge f = 100$

Operation : $P0 = \text{def init}(\text{nom}, l, h, p, f, v, a)$

Oracle :

$h \leq 0$

Une exception est levee

Cas de test : Personnage::testInitFailing

CI : $\text{nom} = \text{"Alex"} \wedge l = 30 \wedge h = 30 \wedge p = -5 \wedge a = 10 \wedge v = 100 \wedge f = 100$

Operation : $P0 = \text{def init}(\text{nom}, l, h, p, f, v, a)$

Oracle :

$p \leq 0$

Une exception est levee

Cas de test : Personnage::testInitFailing

CI : $\text{nom} = \text{"Alex"} \wedge l = 30 \wedge h = 30 \wedge p = 30 \wedge a = -10 \wedge v = 100 \wedge f = 100$

Operation : $P0 = \text{def init}(\text{nom}, l, h, p, f, v, a)$

Oracle :

$a < 0$

Une exception est levee

```

Cas de test : Personnage::testInitFailing
CI : nom = "Alex"  $\wedge$  l = 30  $\wedge$  h = 30  $\wedge$  p = 30  $\wedge$  a = 10  $\wedge$  v = 0  $\wedge$  f = 100
Operation : P0 =def init(nom,l,h,p,f,v,a)
Oracle :
  v  $\leq$  0
  Une exception est levee

Cas de test : Personnage::testInitFailing
CI : nom = "Alex"  $\wedge$  l = 30  $\wedge$  h = 30  $\wedge$  p
= 30  $\wedge$  a = 10  $\wedge$  v = 100  $\wedge$  f = -8
Operation : P0 =def init(nom,l,h,p,f,v,a)
Oracle :
  f  $\leq$  0
  Une exception est levee

Cas de test : Personnage::RetraitVieWorking
CI : personnage = init("Alex", 10, 10, 10, 10, 100, 100)
Operation : P0 =def retraitPdV(personnage, 3)
Oracle :
  argent(P0) = 7 = (10-3)

Cas de test : Personnage::RetraitVieFailing
CI : personnage = init("Alex", 10, 10, 10, 10, 100, 100)
Operation : P0 =def retraitPdV(personnage, -5);
Oracle :
  -5 < 0
  Une exception est levee

Cas de test : Personnage::RetraitArgentWorking
CI : personnage = init("Alex", 10, 10, 10, 10, 100, 100)
Operation : P0 =def retraitArgent(personnage, 3)
Oracle :
  pointsDeVie(P0) = 7 = (10-3)

Cas de test : Personnage::RetraitArgentFailing
CI : personnage = init("Alex", 10, 10, 10, 10, 100, 100)
Operation : P0 =def retraitArgent(personnage, -5);
Oracle :
  -5 < 0
  Une exception est levee

Cas de test : Personnage::RetraitArgentFailing
CI : personnage = init("Alex", 10, 10, 10, 10, 100, 100)
Operation : P0 =def retraitArgent(personnage, 108);
Oracle :
  108 > 100
  Une exception est levee

Cas de test : Personnage::DepotAgentWorking
CI : personnage = init("Alex", 10, 10, 10, 10, 100, 100)
Operation : P0 =def depotArgent(personnage, 3)
Oracle :
  pointsDeVie(P0) = 7 = (10-3)

Cas de test : Personnage::DepotArgentFailing
CI : personnage = init("Alex", 10, 10, 10, 10, 100, 100)
Operation : P0 =def depotArgent(personnage, -5);
Oracle :
  -5 < 0
  Une exception est levee

```

```

Cas de test : Personnage::ramasserObjetWorking
CI : personnage = init("Alex", 10, 10, 10, 10, 100, 100),
    obj = Objet::init("arme",10,0)
Operation : P0 =def ramasserObjet(personnage, obj)
Oracle :
    objetEquipe(personnage) = obj
    force(personnage) = 110

Cas de test : Personnage::ramasserObjetFailing1
CI : personnage = init("Alex", 10, 10, 10, 10, 100, 100)
    obj = Objet::init("sous",0,40)
Operation : P0 =def ramasserObjet(personnage, obj);
Oracle :
    Objet:est_equipable(obj) = false
    Une exception est levee

Cas de test : Personnage::ramasserObjetFailing2
CI : personnage = init("Alex", 10, 10, 10, 10, 100, 100)
    p = Personnage::init("Ryan",10,10,10,100,100)
    personnage = ramasser_perso(p)
    obj = Objet::init("arme",10,0)
Operation : P0 =def ramasserObjet(personnage, obj);
Oracle :
    est_equipe_perso(personnage) = true
    Une exception est levee

Cas de test : Personnage::ramasserArgentWorking
CI : personnage = init("Alex", 10, 10, 10, 10, 100, 100),
    obj = Objet::init("piece",0,40)
Operation : P0 =def ramasserObjet(personnage, obj)
Oracle :
    objetEquipe(personnage) = obj
    force(personnage) = 110

Cas de test : Personnage::ramasserArgentFailing1
CI : personnage = init("Alex", 10, 10, 10, 10, 100, 100)
    personnage = retraitVie(personnage,200)
    obj = Objet::init("soussou",0,40)
Operation : P0 =def ramasserArgent(personnage, obj)
Oracle :
    estVaincu(P0) = true
    Une exception est levee

Cas de test : Personnage::ramasserArgentFailing2
CI : personnage = init("Alex", 10, 10, 10, 10, 100, 100)
    obj = Objet::init("soussou",30,0)
Operation : P0 =def ramasserArgent(personnage, obj);
Oracle :
    Objet:est_DeValeur(obj) = false
    Une exception est levee

Cas de test : Personnage::ramasserPersoWorking
CI : personnage = init("Alex", 10, 10, 10, 10, 100, 100)
    perso2 = init("Ryan", 10, 10, 10, 10, 100, 100)
Operation : P0 =def ramasserPerso(personnage, perso2)
Oracle :
    persoEquipe(P0) = perso2

Cas de test : Personnage::ramasserPersoFailing1
CI : personnage = init("Alex", 10, 10, 10, 10, 100, 100)
    perso2 = init("Ryan", 10, 10, 10, 10, 100, 100)

```

```

        personnage = personnage.retraitPdV(10000);
Operation : P0 =def ramasserPerso(personnage, perso2 )
Oracle :
    estVaincu(P0) = true
    Une exception est levee

Cas de test : Personnage::jeterWorking
CI : personnage = init("Alex", 10, 10, 10, 10, 100, 100)
    obj = Objet::init("arme",10,0)
    personnage = ramasserObjet(personnage,obj)
Operation : P0 =def jeter(personnage)
Oracle :
    persoEquipe(P0) = null
    force(P0) = 100
    objetEquipe(P0) = false

Cas de test : Personnage::jeterFailing1
CI : personnage = init("Alex", 10, 10, 10, 10, 100, 100)
Operation : P0 =def jeter(personnage)
Oracle :
    estEquipeObjet(personnage) = false
    estEquipePerso(personnage) = false
    Une exception est levee

```

0.2 Gangster

0.3 Objet

0.4 Terrain

0.5 Moteur de jeu

0.6 GestionCombat