Rapport de projet
CPS : River City Ransom

Béatrice CARRÉ
Steven VAROUMAS

18 avril 2014

Introduction

La méthodologie pour la *conception par contrat* se compose de trois phases :

1. Analyse : spécifications algébriques
2. Conception : par contrat à partir des spécifications
3. Implémentation : garantie vis-à-vis des contrats

Nous avons suivi scrupuleusement chacune de ces étapes pour l'élaboration du projet et ainsi honorer notre contrat défini.

Le projet consiste à développer un jeu : *River city ransom* TODO explication projet

Dans ce rapport, seront présentées la méthodologie utilisée pour réaliser ce projet, ainsi que les difficultés rencontrées.

Ensuite, notre réalisation du procédé appliqué et les solutions apportées au problème posé vous seront détaillées.

1 Problème posé

Le problème est de rester cohérents tout au long des phases d'analyse, de conception et d'implémentation, tout en respectant la méthodologie de la *conception par contrat*, pour ainsi obtenir un programme tout aussi cohérent et fiable.

1.1 Analyse : la 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. Pour chaque service décrit, il faut en sortir une spécification la plus détaillée et complète possible.

Nous avons rencontré plusieurs difficultés :

– Les problèmes d'interprétation personnelle, qui peuvent être très différentes selon les membres du groupe de travail.

- Le manque d'information dans la description de chaque service est à compléter avec son imagination, qui doit être réaliste.
- Il faut rester cohérents au fur et à mesure de la création de chaque service, pour avoir le moins possible à revenir aux services déjà traités.
- Il est important de garder en tête la notion de référentiel pour chaque service, qui permet de ne pas traiter des éléments qui le concerneraient de l'extérieur, et donc de ne pas définir dans celui-ci.
- La représentation de certains éléments nous a parue difficile, comme pour une liste ou encore la notion de temps.

1.2 Tests

La seconde phase est de décrire et d'implémenter les tests pour préparer le respect du contrat lors de la dernière phase.

Celle-ci est importante dans la mesure où elle représente le liens entre la spécification et l'implémentation pure, tant au niveau des conditions, qu'au niveau de TODO.

Pour n conditions sur chaque opération => $(2 \text{ à } \sim 6)^n$ 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'intervales de tests.

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

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'implementation :
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

Observators :
   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 ¬est_vaincu(P) ∧ (est_equipe_objet(P) ∨ est_equipe_perso (P))

Observations :

[**invariants**]

est_vaincu(P) $\overset{min}{=}$ points_de_vie(P) ≤ 0

est_equipe_perso(P) $\overset{min}{=}$ perso_equipe(P) ≠ null

est_equipe_objet(P) $\overset{min}{=}$ objet_equipe(P) ≠ null

[**init**]

nom(init(n,l,h,p,f,v,a))=n

largeur(init(n,l,h,p,f,v,a))=l

hauteur(init(n,l,h,p,f,v,a))=h

profondeur(init(n,l,h,p,f,v,a))=p

force(init(n,l,h,p,f,v,a))=f

points_de_vie(init(n,l,h,p,f,v,a))=v

somme_d_argent(init(n,l,h,p,f,v,a))=a

objet_equipe(init(n,l,h,p,f,v,a))=null

perso_equipe(init(n,l,h,p,f,v,a))=null

[**retrait_vie**]

points_de_vie(retrait_vie(P,s)) = max(0,points_de_vie(P) − s)

[**retrait_argent**]

somme_d_argent(retrait_argent(P,s)) = argent(P) − s

[**depot_argent**]

somme_d_argent(depot_argent(P,s)) = argent(P) + s

[**ramasser_objet**]

objet_equipe(ramasser_objet(P,objet)) = objet

force(ramasser_objet(P,objet)) = force(P) + Objet::bonus_force(objet)

[**ramasser_argent**]

somme_d_argent(ramasser_objet(P,objet)) = somme_d_argent(P) + Objet::valeur_marchande(objet)

[**ramasser_perso**]

perso_equipe(ramasser_perso(P,perso)) = perso

[**jeter**]

perso_equipe(jeter(P)) = null

force(jeter(P)) =

$$\begin{cases} \text{force(P) - Objet : :bonus\_force(objet\_equipe(P)) si est\_equipe\_objet(P)} \\ \text{force(P) sinon} \end{cases}$$

objet_equipe(jeter(P)) = 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},
Observators :
      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
Observators :
const nom : [Object] → String
est_equipable : [Objet] → boolean
est_de_valeur : [Objet] → boolean
bonus_force : [Objet] → int
pre bonus_force(O) require est_equipable(O)
valeur_marchande : [Objet] → 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) $\overset{min}{=}$ bonus_force > 0
est_de_valeur(O) $\overset{min}{=}$ valeur_marchande > 0
est_equipable(O) $\overset{min}{=}$ ¬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
Observators :
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}
Observators :
      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) =min (Personnage:: est_vaincu(GestionCombat:: alex(combat(M)))
        ∧ Personnage:: est_vaincu(GestionCombat:: ryan(combat(M))))
        ∨ Gangster:: est_vaincu(GestionCombat:: slick(combat(M)))
```

resultat(M) $\overset{min}{=}$

ALEXGAGNANT	si ¬Personnage : :est_vaincu(GestionCombat : :alex(combat(M))) ∧ Gangster : :est_vaincu(GestionCombat : :slick(combat(M))) ∧ Personnage : :est_vaincu(GestionCombat : :ryan(combat(M)))
RYANGAGNANT	si ¬Personnage : :est_vaincu(GestionCombat : :ryan(combat(M))) ∧ Gangster : :est_vaincu(GestionCombat : :slick(combat(M))) ∧ Personnage : :est_vaincu(GestionCombat : :alex(combat(M)))
DEUXGAGNANTS	si ¬Personnage : :est_vaincu(GestionCombat : :ryan(combat(M))) ∧ Gangster : :est_vaincu(GestionCombat : :slick(combat(M))) ∧ ¬Personnage : :est_vaincu(GestionCombat : :alex(combat(M)))
SLICKGAGNANT	si Personnage :est_vaincu(GestionCombat : :ryan(combat(M))) ∧ ¬Gangster : :est_vaincu(GestionCombat : :slick(combat(M))) ∧ Personnage : :est_vaincu(GestionCombat : :alex(combat(M)))
NULLE	sinon

```
[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}

Observators :

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, perso1, perso2) require
(perso1 = alex(G) ∨ perso1 = ryan(G)) ∧ (perso2 = slick(G) ∨ perso2 ∈ gangsters(G))

collisionGauche : [GestionCombat] × Personnage × Gangster → boolean
pre collisionGauche(G, perso1, perso2) require
(perso1 = alex(G) ∨ perso1 = ryan(G)) ∧ (perso2 = slick(G) ∨ perso2 ∈ gangsters(G))

```
        collisionDevant : [GestionCombat] × Personnage × Gangster → boolean
         pre collisionDevant(G, perso1, perso2) require
         (perso1 = alex(G) ∨ perso1 = ryan(G)) ∧ (perso2 = slick(G) ∨ perso2 ∈ gangsters(G))

        collisionDerriere : [GestionCombat] × Personnage × Gangster → boolean
         pre collisionDerriere(G, perso1, perso2) require
         (perso1 = alex(G) ∨ perso1 = ryan(G)) ∧ (perso2 = slick(G) ∨ perso2 ∈ gangsters(G))

        collisionDessus: [GestionCombat] × Personnage × Gangster → boolean
         pre collisionDessus(G, perso1, perso2) require
         (perso1 = alex(G) ∨ perso1 = ryan(G)) ∧ (perso2 = slick(G) ∨ perso2 ∈ gangsters(G))

        collisionDessous: [GestionCombat] × Personnage × Gangster → boolean
         pre collisionDessous(G, perso1, perso2) require
         (perso1 = alex(G) ∨ perso1 = ryan(G)) ∧ (perso2 = slick(G) ∨ perso2 ∈ gangsters(G))

        collision : [GestionCombat] × Personnage × Gangster → boolean
         pre collision(G, perso1, perso2) require
         (perso1 = alex(G) ∨ perso1 = 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) ≝^min( -d ≤ posX(G,p1) - posX(G,p2) ≤ d+1) ∧ ( d = Personnage::largeur(p1)/2 + d = Personnage::
        largeur(p2)/2 )

    collisionGauche(G,p1,p2) ≝^min( -d ≤ posX(G,p2) - posX(G,p1) ≤ d+1) ∧ ( d = Personnage::largeur(p1)/2 + d = Personnage::
        largeur(p2)/2 )

    collisionDevant(G,p1,p2) ≝^min( -d ≤ posY(G,p1) - posY(G,p2) ≤ d+1) ∧ ( d = Personnage::profondeur(p1)/2 + d = Personnage
        ::profondeur(p2)/2 )

    collisionDerriere(G,p1,p2) ≝^min( -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 ≤ posZ(G,p1) − posZ(G,p2) ≤ d+1) ∧ (d = Personnage::hauteur(p1)/2 + d = Personnage::hauteur(p2)/2)

collisionDessus(G,p1,p2) $\stackrel{min}{=}$(−d ≤ posZ(G,p2) − posZ(G,p1) ≤ d+1) ∧ (d = Personnage::hauteur(p1)/2 + d = Personnage::hauteur(p2)/2)

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

actionGangster(G,g) = RIEN si estGele(G,g) ∨ est_vaincu(G,g) ∀ g ∈ 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)}, ∀ g ∈ gangsters(G)

actionGangster(G,g) = RIEN ∀ g ∈ 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

$$estFrappe(init(), s) = false$$

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

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

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

$$posZ(init(), p) = 0$$

$$posY(init(), perso) = random$$

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

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

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

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

[gerer]

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

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

% VIRER LES CONDITIONS DE COLLISION (affreuses)

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

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

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

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

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

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

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

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

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

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

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

$$
\begin{aligned}
&\mathrm{posX}(G, \mathrm{gerer}(G, cA, cR), \mathrm{gangsters}(G)) = \{\ g = \\
&\quad \begin{cases}
\min(\mathrm{posX}(G,g) + 10, \mathrm{Terrain} : \mathrm{largeur}(\mathrm{terrain}(G)) - \mathrm{Personnage} : : \mathrm{largeur}(g)) \\
\text{si } \mathrm{actionGangster}(G,g) = \mathrm{DROITE} \vee \mathrm{actionGangster}(G,g) = \mathrm{SAUTDROITE} \wedge \neg \mathrm{Personnage} : : \mathrm{est\_vaincu}(g) \wedge \neg \mathrm{estGele}(G,g) \\
\wedge \neg \mathrm{collisionDroite}(g,p)\ \forall\ p \in \mathrm{gangster} \vee p = g \\
\max(\mathrm{posX}(G,g) - 10, 0) \\
\text{si } \mathrm{actionGangster}(G,g) = \mathrm{GAUCHE} \vee \mathrm{actionGangster}(G,g) = \mathrm{SAUTGAUCHE} \wedge \neg \mathrm{Personnage} : : \mathrm{est\_vaincu}(g) \wedge \neg \mathrm{estGele}(G,g) \\
\wedge \neg \mathrm{collisionGauche}(g,p)\ \forall\ p \in \mathrm{gangster} \vee p = g \\
\mathrm{posX}(G,g) \text{ sinon}
\end{cases} \\
&\} \quad \forall\ g \in \mathrm{gangsters}(G)
\end{aligned}
$$

$$
\begin{aligned}
&\mathrm{posY}(G, \mathrm{gerer}(G, cA, cR), \mathrm{gangsters}(G)) = \{\ g = \\
&\quad \begin{cases}
\min(\mathrm{posY}(G,g) + 10, \mathrm{Terrain} : \mathrm{profondeur}(\mathrm{terrain}(G)) - \mathrm{Personnage} : : \mathrm{profondeur}(g)) \\
\text{si } \mathrm{actionGangster}(G,g) = \mathrm{HAUT} \vee \mathrm{actionGangster}(G,g) = \mathrm{SAUTHAUT} \wedge \neg \mathrm{Personnage} : : \mathrm{est\_vaincu}(g) \wedge \neg \mathrm{estGele}(G,g) \wedge \neg \mathrm{collisionDerriere}(g,p)\ \forall\ p \in \mathrm{gangster} \vee p = \\
\max(\mathrm{posY}(G,g) - 10, 0) \\
\text{si } \mathrm{actionGangster}(G,g) = \mathrm{BAS} \vee \mathrm{actionGangster}(G,g) = \mathrm{SAUTBAS} \wedge \neg \mathrm{Personnage} : : \mathrm{est\_vaincu}(g) \wedge \neg \mathrm{estGele}(G,g) \wedge \neg \mathrm{collisionDevant}(g,p)\ \forall\ p \in \mathrm{gangster} \vee p = g \\
\mathrm{posY}(G,g) \text{ sinon}
\end{cases} \\
&\} \quad \forall\ g \in \mathrm{gangsters}(G)
\end{aligned}
$$

$$
\begin{aligned}
&\mathrm{posZ}(\mathrm{gerer}(G, cA, cR), \mathrm{gangsters}(G)) = \{\ g = \\
&\quad \begin{cases}
100 \text{ si } \mathrm{actionGangster}(G,g) = \mathrm{SAUT} \vee \mathrm{actionGangster}(G,g) = \mathrm{SAUTBAS} \vee \mathrm{actionGangster}(G,g) = \mathrm{SAUTHAUT} \vee \mathrm{actionGangster}(G,g) = \mathrm{SAUTDROITE} \\
\vee\ \mathrm{actionGangster}(G,g) = \mathrm{SAUTGAUCHE} \wedge \neg \mathrm{Personnage} : : \mathrm{est\_vaincu}(g) \wedge \neg \mathrm{estGele}(G,g) \wedge \neg \mathrm{collisionDessus}(g,p)\ \forall\ p \in \mathrm{gangster} \vee p = g \\
\mathrm{pos}(G,g) \text{ si } \mathrm{estGele}(G,g) \\
0 \text{ Sinon}
\end{cases} \\
&\} \ \forall\ g \in \mathrm{gangsters}(G)
\end{aligned}
$$

$$
\begin{aligned}
&\mathrm{alex}(\mathrm{gerer}(G, cA, cR)) = \\
&\begin{cases}
- \mathrm{Personnage} : : \mathrm{jeter}(\mathrm{alex}(G)) \\
\text{si } cA = \mathrm{JETER} \wedge \neg \mathrm{Personnage} : : \mathrm{est\_vaincu}(\mathrm{alex}(G)) \wedge \neg \mathrm{estGele}(G, \mathrm{alex}(G)) \wedge \mathrm{Bloc} : : \mathrm{type}(\mathrm{Terrain} : : \mathrm{bloc}(\mathrm{terrain}(G), \mathrm{posX}(G, \mathrm{alex}(G)), \mathrm{posY}(G, \mathrm{alex}(G)))) = \mathrm{VIDE} \\
- \mathrm{Personnage} : : \mathrm{ramasser\_objet}(\mathrm{alex}(G), \mathrm{Bloc} : : \mathrm{objet}(\mathrm{Terrain} : : \mathrm{bloc}(\mathrm{terrain}(G), \mathrm{posX}(G, \mathrm{alex}(G)), \mathrm{posY}(G, \mathrm{alex}(G))))) \\
\text{si } cA = \mathrm{RAMASSER} \wedge \mathrm{posZ}(G, \mathrm{alex}(G)) = 0 \wedge \neg \mathrm{Personnage} : : \mathrm{est\_vaincu}(\mathrm{alex}(G)) \wedge \neg \mathrm{estGele}(G, \mathrm{alex}(G)) \\
- \mathrm{Personnage} : : \mathrm{ramasser\_perso}(\mathrm{alex}(G), p) \\
\text{si } \mathrm{collision}(G, \mathrm{alex}(G), p) \wedge cA = \mathrm{RAMASSER} \wedge \neg \mathrm{Personnage} : : \mathrm{est\_vaincu}(\mathrm{alex}(G)) \wedge \neg \mathrm{estGele}(G, \mathrm{alex}(G)) \\
- \mathrm{Personnage} : : \mathrm{ramasser\_argent}(\mathrm{alex}(G), \mathrm{Bloc} : : \mathrm{objet}(\mathrm{Terrain} : : \mathrm{bloc}(\mathrm{terrain}(G), \mathrm{posX}(G, \mathrm{alex}(G)), \mathrm{posY}(G, \mathrm{alex}(G))))) \\
\text{si } cA = \mathrm{RAMASSER} \wedge \mathrm{posZ}(G, \mathrm{alex}(G)) = 0 \wedge \neg \mathrm{Personnage} : : \mathrm{est\_vaincu}(\mathrm{alex}(G)) \wedge \neg \mathrm{estGele}(G, \mathrm{alex}(G)) \\
- \mathrm{Personnage} : : \mathrm{retrait\_vie}(\mathrm{alex}(G), \mathrm{Personnage} : : \mathrm{force}(p)) \\
\text{si } \mathrm{collision}(G, \mathrm{alex}(G), p) \wedge \mathrm{actionGangster}(G,p) = \mathrm{FRAPPER} \wedge \neg \mathrm{Personnage} : : \mathrm{est\_vaincu}(\mathrm{alex}(G)) \wedge \neg \mathrm{estGele}(G, \mathrm{alex}(G)) \\
\ - \mathrm{Personnage} : : \mathrm{retrait\_vie}(\mathrm{alex}(G), \mathrm{Personnage} : : \mathrm{points\_de\_vie}(\mathrm{alex}(G)) \\
\ \text{si } \mathrm{Bloc} : : \mathrm{type}(\mathrm{Terrain} : : \mathrm{bloc}(\mathrm{terrain}(G), \mathrm{posX}(G, \mathrm{alex}(G)), \mathrm{posY}(G, \mathrm{alex}(G)))) = \mathrm{FOSSE} \wedge \neg \mathrm{Personnage} : : \mathrm{est\_vaincu}(\mathrm{alex}(G)) \wedge \neg \mathrm{estGele}(G, \mathrm{alex}(G)) \\
- \mathrm{alex}(G) \text{ Sinon}
\end{cases}
\end{aligned}
$$

$$
\begin{aligned}
&ryan(gerer(G, cA, cR)) = \\
&\begin{cases}
\text{- Personnage : :jeter(ryan(G))} \\
\text{si } cR = \text{JETER} \wedge \neg\text{Personnage : :est\_vaincu(ryan(G))} \wedge \neg\text{estGele(G,ryan(G))} \wedge \text{Bloc : :type(Terrain : :bloc(terrain(G),posX(G,ryan(G)),posY(G,ryan(G))))=VIDE} \\
\text{- Personnage : :ramasser\_objet(ryan(G), Bloc : :objet(Terrain : :bloc(terrain(G), posX(G,ryan(G)),posY(G,ryan(G)))))} \\
\text{si } cR = \text{RAMASSER} \wedge \text{posZ(G,ryan(G))=0} \wedge \neg\text{Personnage : :est\_vaincu(ryan(G))} \wedge \neg\text{estGele(G,ryan(G))} \\
\text{- Personnage : :ramasser\_perso(ryan(G), p)} \\
\text{si collision(G,ryan(G), p)} \wedge cR = \text{RAMASSER} \wedge \neg\text{Personnage : :est\_vaincu(ryan(G))} \wedge \neg\text{estGele(G,ryan(G))} \\
\text{- Personnage : :ramasser\_argent(ryan(G), Bloc : :objet(Terrain : :bloc(terrain(G), posX(G,ryan(G)),posY(G,ryan(G)))))} \\
\text{si } cR = \text{RAMASSER} \wedge \text{posZ(G,ryan(G))=0} \wedge \neg\text{Personnage : :est\_vaincu(ryan(G))} \wedge \neg\text{estGele(G,ryan(G))} \\
\text{- Personnage : :retrait\_vie(ryan(G), Personnage : :force(p))} \\
\text{si collision(G,ryan(G),p)} \wedge \text{actionGangster(G,p)} = \text{FRAPPER} \wedge \neg\text{Personnage : :est\_vaincu(ryan(G))} \wedge \neg\text{estGele(G,ryan(G))} \\
\text{- Personnage : :retrait\_vie(ryan(G), Personnage : :points\_de\_vie(ryan(G))} \\
\text{si Bloc : :type(Terrain : :bloc(terrain(G), posX(G,ryan(G)), posY(G,ryan(G)))) = FOSSE} \wedge \neg\text{Personnage : :est\_vaincu(ryan(G))} \wedge \neg\text{estGele(G,ryan(G))} \\
\text{- ryan(G) Sinon}
\end{cases} \\
&gangsters(gerer(G, cA, cR)) = \{\ g = \\
&\begin{cases}
\text{- Gangster : :jeter(g)} \\
\text{si actionGangster(G,g) = JETER} \wedge \neg\text{Personnage : :est\_vaincu(g)} \wedge \neg\text{estGele(G,g)} \wedge \text{Bloc : :type(Terrain : :bloc(terrain(G),posX(G,g),posY(G,g)))=VIDE} \\
\text{- Gangster : :ramasser\_objet(g, Bloc : :objet(Terrain : :bloc(terrain(G), posX(G,g),posY(G,g))))} \\
\text{si actionGangster(G,g) = RAMASSER} \wedge \text{posZ(G,g)=0} \wedge \neg\text{Gangster : :est\_vaincu(g)} \wedge \neg\text{estGele(G,g)} \\
\text{- Gangster : :retrait\_vie(g, Personnage : :force(p))} \\
\text{si collision(G,alex(G),g)} \wedge cA = \text{FRAPPER} \wedge \neg\text{Gangster : :est\_vaincu(g)} \wedge \neg\text{estGele(G,g)} \\
\text{- Gangster : :retrait\_vie(g, Personnage : :force(p))} \\
\text{si collision(G,ryan(G),g)} \wedge cR = \text{FRAPPER} \wedge \neg\text{Gangster : :est\_vaincu(g)} \wedge \neg\text{estGele(G,g)} \\
\text{- Gangster : :retrait\_vie(g, Gangster : :points\_de\_vie(g)} \\
\text{si Bloc : :type(Terrain : :bloc(terrain(G), posX(G,g), posY(G,g))) = FOSSE} \wedge \neg\text{Gangster : :est\_vaincu(g)} \wedge \neg\text{estGele(G,g)} \\
\text{- g Sinon}
\end{cases} \\
&\}\quad \forall\ g \in gangsters(G) \\
&slick(gerer(G, cA, cR)) = \\
&\begin{cases}
\text{- Gangster : :jeter(slick(G))} \\
\text{si actionGangster(G,slick(G)) = JETER} \wedge \neg\text{Personnage : :est\_vaincu(slick(G))} \wedge \neg\text{estGele(G,slick(G))} \wedge \\
\text{Bloc : :type(Terrain : :bloc(terrain(G),posX(G,slick(G)),posY(G,slick(G))))=VIDE} \\
\text{- Gangster : :ramasser\_objet(g, Bloc : :objet(Terrain : :bloc(terrain(G), posX(G,slick(G)),posY(G,slick(G)))))} \\
\text{si actionGangster(G,g) = RAMASSER} \wedge \text{posZ(G,slick(G))=0} \wedge \neg\text{Gangster : :est\_vaincu(slick(G))} \wedge \neg\text{estGele(G,slick(G))} \\
\text{- Gangster : :retrait\_vie(slick(G), Personnage : :force(p))} \\
\text{si collision(G,alex(G),slick(G))} \wedge cA = \text{FRAPPER} \wedge \neg\text{Gangster : :est\_vaincu(slick(G))} \wedge \neg\text{estGele(G,g)} \\
\text{- Gangster : :retrait\_vie(slick(G), Personnage : :force(p))} \\
\text{si collision(G,ryan(G),slick(G))} \wedge cR = \text{FRAPPER} \wedge \neg\text{Gangster : :est\_vaincu(slick(G))} \wedge \neg\text{estGele(G,slick(G))} \\
\text{- Gangster : :retrait\_vie(slick(G), Gangster : :points\_de\_vie(slick(G))} \\
\text{si Bloc : :type(Terrain : :bloc(terrain(G), posX(G,slick(G)), posY(G,slick(G)))) = FOSSE} \wedge \neg\text{Gangster : :est\_vaincu(slick(G))} \wedge \neg\text{estGele(G,slick(G))} \\
\text{- slick(G) Sinon}
\end{cases}
\end{aligned}
$$

$$estVisible(gerer(G,cA,cR),alex(G)) = \begin{cases} true & \text{si } \neg \text{Personnage : :est\_vaincu(alex(G))} \\ false & \text{sinon} \end{cases}$$

$$estVisible(gerer(G,cA,cR),ryan(G)) = \begin{cases} true & \text{si } \neg \text{Personnage : :est\_vaincu(ryan(G))} \\ false & \text{sinon} \end{cases}$$

$$estVisible(gerer(G,cA,cR),g)) = \left\{ \begin{array}{ll} true & \text{si } \neg \text{Gangster : :est\_vaincu(g)} \\ true & \text{si Personnage :perso\_equipe(alex(G)) = g} \wedge cA = \text{JETER} \\ false & \text{si collision(G,alex(G),g)} \wedge cA = \text{RAMASSER} \\ false & \text{sinon} \end{array} \right\} \forall\ g \in gangsters(G)$$

$$estVisible(gerer(G,cA,cR),slick(G))) = \begin{cases} true & \text{si } \neg \text{Gangster : :est\_vaincu(slick(G))} \\ true & \text{si Personnage :perso\_equipe(alex(G)) = slick(G)} \wedge cA = \text{JETER} \\ false & \text{si collision(G,alex(G),slick(G))} \wedge cA = \text{RAMASSER} \\ false & \text{sinon} \end{cases}$$

$$posX(G, gerer(G,cA,cR),p) = \begin{cases} posX(G,alex(G)) & \text{si } cA = \text{JETER} \wedge \text{Personnage : :perso\_equipe(alex(G)) = p} \wedge \neg\ \text{Personnage : :est\_vaincu(alex(G))} \\ posX(G,p) & \text{sinon} \end{cases}$$

$$posY(G, gerer(G,cA,cR),p) = \begin{cases} posY(G,alex(G)) & \text{si } cA = \text{JETER} \wedge \text{Personnage : :perso\_equipe(alex(G)) = p} \wedge \neg\ \text{Personnage : :est\_vaincu(alex(G))} \\ posY(G,p) & \text{sinon} \end{cases}$$

$$posZ(gerer(G,cA,cR),p) = \begin{cases} 0 & \text{si } cA = \text{JETER} \wedge \text{Personnage : :perso\_equipe(alex(G)) = p} \wedge \neg\ \text{Personnage : :est\_vaincu(alex(G))} \\ posZ(G,p) & \text{sinon} \end{cases}$$

$$posX(G, gerer(G,cA,cR),p) = \begin{cases} posX(G,ryan(G)) & \text{si } cA = \text{JETER} \wedge \text{Personnage : :perso\_equipe(ryan(G)) = p} \wedge \neg\ \text{Personnage : :est\_vaincu(ryan(G))} \\ posX(G,p) & \text{sinon} \end{cases}$$

$$posY(G, gerer(G,cA,cR),p) = \begin{cases} posY(G,ryan(G)) & \text{si } cA = \text{JETER} \wedge \text{Personnage : :perso\_equipe(ryan(G)) = p} \wedge \neg\ \text{Personnage : :est\_vaincu(ryan(G))} \\ posY(G,p) & \text{sinon} \end{cases}$$

$$posZ(gerer(G,cA,cR),p) = \begin{cases} 0 & \text{si } cA = \text{JETER} \wedge \text{Personnage : :perso\_equipe(ryan(G)) = p} \wedge \neg\ \text{Personnage : :est\_vaincu(ryan(G))} \\ posZ(G,p) & \text{sinon} \end{cases}$$

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

$$\begin{cases} \text{- Bloc : :retirerObjet(Terrain : :bloc(terrain(G), posX(G,alex(G)), posY(G,alex(G))))} \\ \text{si cA} = \text{RAMASSER} \wedge \neg \text{ Personnage : :est\_vaincu(alex(G))} \wedge \neg\text{estGele(G,alex(G))} \\ \text{- Bloc : :poserObjet(Terrain : :bloc(terrain(G), posX(G,alex(G)), posY(G,alex(G)))), Personnage :objet\_equipe(alex(G))} \\ \text{si cA} = \text{JETER} \wedge \text{Personnage : :est\_equipe\_objet(alex(G))} = \text{true} \wedge \neg \text{ Personnage : :est\_vaincu(alex(G))} \wedge \neg\text{estGele(G,alex(G))} \\ \text{- Terrain : :bloc(terrain(G), posX(G,alex(G)), posY(G,alex(G))) Sinon} \end{cases}$$

$$\text{Terrain : : bloc(terrain(gerer(G, cA, cR)), posX(G, ryan(G)), posY(G, ryan(G))) =}$$
$$\begin{cases} \text{- Bloc : :retirerObjet(Terrain : :bloc(terrain(G), posX(G,ryan(G)), posY(G,ryan(G))))} \\ \text{si cR} = \text{RAMASSER} \wedge \neg \text{ Personnage : :est\_vaincu(ryan(G))} \wedge \neg\text{estGele(G,ryan(G))} \\ \text{- Bloc : :poserObjet(Terrain : :bloc(terrain(G), posX(G,ryan(G)), posY(G,ryan(G)))), Personnage :objet\_equipe(ryan(G))} \\ \text{si cR} = \text{JETER} \wedge \text{Personnage : :est\_equipe\_objet(ryan(G))} = \text{true} \wedge \neg \text{ Personnage : :est\_vaincu(ryan(G))} \wedge \neg\text{estGele(G,ryan(G))} \\ \text{- Terrain : :bloc(terrain(G), posX(G,ryan(G)), posY(G,ryan(G))) Sinon} \end{cases}$$

$$\text{Terrain : : bloc(terrain(gerer(G, cA, cR)), posX(G, slickG)), posY(G, slick(G))) =}$$
$$\begin{cases} \text{- Bloc : :retirerObjet(Terrain : :bloc(terrain(G), posX(G,slick(G)), posY(G,slick(G))))} \\ \text{si actionGangster(G,slick(G))} = \text{RAMASSER} \wedge \neg \text{ Personnage : :est\_vaincu(slick(G))} \wedge \neg\text{estGele(G,slick(G))} \\ \text{- Bloc : :poserObjet(Terrain : :bloc(terrain(G), posX(G,slick(G)), posY(G,slick(G)))), Personnage :objet\_equipe(slick(G))} \\ \text{si actionGangster(G,slick(G))} = \text{JETER} \wedge \text{Personnage : :est\_equipe\_objet(slick(G))} = \text{true} \wedge \neg \text{ Personnage : :est\_vaincu(slick(G))} \wedge \neg\text{estGele(G,slick(G))} \\ \text{- Terrain : :bloc(terrain(G), posX(G,slick(G)), posY(G,slick(G))) Sinon} \end{cases}$$

$$\text{Terrain : : bloc(terrain(gerer(G, cA, cR)), posX(G, g), posY(G, g)) =}$$
$$\begin{cases} \text{- Bloc : :retirerObjet(Terrain : :bloc(terrain(G), posX(G,g), posY(G,g)))} \\ \text{si actionGangster(G,g)} = \text{RAMASSER} \wedge \neg \text{ Personnage : :est\_vaincu(g)} \wedge \neg\text{estGele(G,g)} \\ \text{- Bloc : :poserObjet(Terrain : :bloc(terrain(G), posX(G,g), posY(G,g))), Personnage :objet\_equipe(g)} \\ \text{si actionGangster(G,g)} = \text{JETER} \wedge \text{Personnage : :est\_equipe\_objet(g)} = \text{true} \wedge \neg \text{ Personnage : :est\_vaincu(g)} \wedge \neg\text{estGele(G,g)} \quad \forall\ g \in \text{gangsters(G)} \\ \text{- Bloc : :poserObjet(Terrain : :bloc(terrain(G), posX(G,g),posY(G,g)), Objet : :init("Recompense",0,1000))} \\ \text{si Gangster : :est\_vaincu(g)} \\ \text{- Terrain : :bloc(terrain(G), posX(G,g), posY(G,g)) Sinon} \end{cases}$$

Tests

0.1 Le service Personnage

```
Cas de test : Personnage::testInitWorking
CI : nom = "Ryan" ∧ l = 30 ∧ h = 30 ∧ p = 30 ∧ a = 10 ∧ v = 100 ∧ f = 100
Operation : P0 =def init(nom,l,h,p,f,v,a)
Oracle :
   nom(P0) == "Ryan"
   largeur(P0) =  30
   profondeur(P0) = 30
   hauteur(P0) = 30
   pointsDeVie(P0) = 100
   force(P0) = 100
   argent(P0) = 10


Cas de test : Personnage::testInitFailing
CI : nom = "Joe" ∧ l = 30 ∧ h = 30 ∧ p = 30 ∧ a = 10 ∧ v = 100 ∧ f = 100
Operation : P0 =def init(nom,l,h,p,f,v,a)
Oracle :
   nom ≠ "Alex" ∧ nom ≠ "Ryan"
   Une exception est levee

Cas de test : Personnage::testInitFailing
CI : nom = "Alex" ∧ l = −5 ∧ h = 30 ∧ p = 30 ∧ a = 10 ∧ v = 100 ∧ f = 100
Operation : P0 =def init(nom,l,h,p,f,v,a)
Oracle :
   l ≤ 0
   Une exception est levee

Cas de test : Personnage::testInitFailing
CI : nom = "Alex" ∧ l = 30 ∧ h = −5 ∧ p = 30 ∧ a = 10 ∧ v = 100 ∧ f = 100
Operation : P0 =def init(nom,l,h,p,f,v,a)
Oracle :
   h ≤ 0
   Une exception est levee

Cas de test : Personnage::testInitFailing
CI : nom = "Alex" ∧ l = 30 ∧ h = 30 ∧ p = −5 ∧ a = 10 ∧ v = 100 ∧ f = 100
Operation : P0 =def init(nom,l,h,p,f,v,a)
Oracle :
   p ≤ 0
   Une exception est levee

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

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

Cas de test : Personnage::testInitFailing
CI : nom = "Alex" ∧ l = 30 ∧ h = 30 ∧ p
= 30 ∧ a = 10 ∧ v = 100 ∧ f = −8
Operation : P0 =def init(nom,l,h,p,f,v,a)
Oracle :
   f ≤ 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