

Rapport de projet

CPS : River City Ransom

Béatrice CARRÉ
Steven VAROUMAS

19 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 spécification

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.

Une grande partie des opérateurs d'un service requièrent la validité de plusieurs pré-conditions permettant leur appel. De la même façon, plusieurs post-conditions définissent quelle est l'action de l'opérateur sur le service en question.

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 $\Rightarrow (2 \text{ à } \sim 6)^n$ 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 ainsi qu'une valeur centrale.

Par exemple, l'observateur $bloc(T, x, y)$ du service *Terrain* possède ce qui semble être deux préconditions simples : $0 \leq x \leq \text{largeur}(T)$ et $0 \leq y \leq \text{profondeur}(T)$

Ce qui se traduit en réalité par 4 conditions :

- $0 \leq x$
- $x \leq \text{largeur}(T)$
- $0 \leq y$
- $y \leq \text{profondeur}(T)$

Pour avoir une étendue de test satisfaisante, il nous faudrait au moins tester pour chaque condition une situation qui rendrait la condition fausse et une situation qui rendrait la condition vraie.

On peut alors écrire une table de vérité afin d'avoir une idée des différents tests à réaliser, selon les cas :

$0 \leq x$	$x \leq \text{largeur}(T)$	$0 \leq y$	$y \leq \text{profondeur}(T)$
\top	\top	\top	\top
\top	\top	\top	\perp
\top	\top	\perp	\top
\top	\top	\perp	\perp
\top	\perp	\top	\top
\top	\perp	\top	\perp
\top	\perp	\perp	\top
\top	\perp	\perp	\perp
\perp	\top	\top	\top
\perp	\top	\top	\perp
\perp	\top	\perp	\perp
\perp	\perp	\top	\top
\perp	\perp	\top	\perp
\perp	\perp	\perp	\top
\perp	\perp	\perp	\perp

Soient $16(= 2^4)$ tests à réaliser.

Certains opérateurs (ou constructeurs) possèdent jusqu'à 8 préconditions, ce qui reviendrait à effectuer au moins $2^8 = 256$ 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. A DÉVELOPPER

1.3 Implémentation par contrat

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

2 Solution

petite intro ?

2.1 Specification

La spécification que nous avons réalisé contient les services suivants :

- **Personnage** qui décrit les comportements des personnages du jeu (Ryan et Alex).
- **Gangster** qui raffine les comportements des personnages pour les ennemis de Ryan et Alex.
- **Bloc**, qui représente un bloc situé sur sol du terrain de jeu.

- **Objet**, définissant les caractéristiques des objets éparpillés sur le sol et pouvant être ramassés par les personnages.
- **Terrain**, représentant le terrain (en trois dimensions) dans lequel les personnages évoluent.
- **MoteurJeu**, service gérant les tours de jeu ainsi que vérifiant l'état de la partie.
- **GestionCombat** qui est le service central de l'application, et qui s'occupe en particulier d'associer à chaque commande envoyée à un personnage des actions concrètes dans le jeu.

Ces services décrivent de nombreux comportements, tous précisés en annexe. Nous nous attacherons ici à préciser quels ont été nos décisions particulières lors de la conception de leur spécification :

- Dans le service *Personnage*, nous avons choisi de séparer l'action de ramasser (un objet ou un personnage) en trois opérateurs différents : *ramasser_objet*, *ramasser_argent*, et *ramasser_perso*. L'idée était ici de permettre à un personnage de ramasser de l'argent sans pour autant se retrouver dans l'impossibilité de ramasser un autre objet par la suite ou bien de continuer à ramasser de l'argent alors qu'il est équipé d'une arme (ou d'un personnage)
- Chaque *Bloc* possède un observateur *type* qui retourne si le bloc est *VIDE* (il ne contient aucun objet ou trésor), *FOSSE* (il représente un trou dans lequel les personnages peuvent tomber) ou *OBJET* (il contient un objet, qui peut être un trésor ou une arme).
- Nous avons décidé qu'il ne pourrait y avoir qu'un seul objet (au maximum) par bloc sur le sol du terrain de jeu. Ainsi, il est impossible pour un personnage de jeter un objet s'il ne se situe pas sur un bloc vide. De même, l'opération *poserObjet* du service *Bloc* n'est permise qu'à condition que le bloc soit de type *VIDE*
- Les *Gangsters* n'étant pas dirigés par les actions du joueur, nous avons créé un observateur *actionGangster* dans *GestionCombat* retournant ce que décide de faire chaque gangster donné à chaque tour de jeu.
- Lorsqu'un *Personnage* en ramasse un autre, nous considérons que le personnage ramassé est devient alors invisible sur le terrain de jeu jusqu'à ce qu'il soit jeté.
- Les *Objets* ne peuvent se situer qu'au sol. Ainsi, le *Terrain* possède un tableau à deux dimensions de *Blocs* représentant une grille se situant au sol et pouvant contenir des objets.
- Chaque *Bloc* fait 50 pixels de large. Le *Terrain* doit donc posséder une largeur et une profondeur multiples de 50.

- Un observateur *estVisible* est ajouté dans *GestionCombat*, permettant essentiellement à l'implémentation graphique du jeu d'afficher ou non un *Personnage* ou un *Gangster* pendant l'exécution du jeu.

2.2 Tests

Nous n'avons malheureusement pas fait tout les tests.

1 exemple ...

Idéalement : pour chaque méthodes, pour chaque valeur :

2.3 Implémentation par contrat

TODO :

Qu'est- ce qu'elle suit, comment elle se déroule

Quel est le squelette de l'implem par contrat (+ exemple?)

De quoi est constituée notre implem (1 version buguée, 1 version pas buguée, 1 version graphique un peu buguée :))

code en annexe

desc des options du fichiers build

+ petit manuel utilisateur

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,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(T) ∧ 0 ≤ y ≤ profondeur(T)

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]

$0 \leq \text{posX}(G, s) \leq \text{Terrain}::\text{largeur}(\text{terrain})$

$0 \leq \text{posY}(G, s) \leq \text{Terrain}::\text{profondeur}(\text{terrain})$

$0 \leq \text{posZ}(G, s) \leq \text{Terrain}::\text{hauteur}(\text{terrain})$

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

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

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

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

Personnage :: profondeur (p2)/2)

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

collisionDessus (G,p1,p2) $\stackrel{min}{=}$ (-d \leq posZ (G,p2) - posZ (G,p1) \leq d+1) \wedge (d = Personnage :: hauteur (p1)/2 + d = Personnage :: 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 = g \\ \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) =
 $\begin{cases} \text{true} & \text{si } \neg \text{Gangster} : \text{est_vaincu}(\text{g}) \\ \text{true} & \text{si Personnage :perso_equipe}(\text{alex}(\text{G})) = \text{g} \wedge \text{cA} = \text{JETER} \\ \text{false} & \text{si collision}(\text{G},\text{alex}(\text{G}),\text{g}) \wedge \text{cA} = \text{RAMASSER} \\ \text{false} & \text{sinon} \end{cases} \quad \forall \text{ g} \in \text{gangsters}(\text{G})$

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

13

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

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

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

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

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

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

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

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

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

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

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

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

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 $\stackrel{def}{=}$ 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 $\stackrel{def}{=}$ 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 $\stackrel{def}{=}$ 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 $\stackrel{def}{=}$ 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 $\stackrel{def}{=}$ 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 $\stackrel{def}{=}$ 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 $\stackrel{def}{=}$ 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 $\stackrel{def}{=}$ 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  $\stackrel{def}{=}$  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  $\stackrel{def}{=}$  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  $\stackrel{def}{=}$  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  $\stackrel{def}{=}$  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  $\stackrel{def}{=}$  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  $\stackrel{def}{=}$  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  $\stackrel{def}{=}$  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  $\stackrel{def}{=}$  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  $\stackrel{def}{=}$  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  $\stackrel{def}{=}$  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  $\stackrel{def}{=}$  jeter(personnage)
Oracle :
    estEquipeObjet(personnage) = false
    estEquipePerso(personnage) = false
    Une exception est levee

```

0.2 Gangster

```
Cas de test : Gangster::testInitWorking
CI : nom = "Slick"  $\wedge$  l = 10  $\wedge$  h = 10  $\wedge$  p = 10  $\wedge$  f = 10  $\wedge$  v = 10
Operation : G0  $\stackrel{def}{=}$  init(nom,l,h,p,f,v)
Oracle :
  nom(P0) = "Slick"
  largeur(P0) = 10
  hauteur(P0) = 10
  profondeur(P0) = 10
  pointsDeVie(P0) = 10
  force(P0) = 10
  argent(P0) = 0

Cas de test : Gangster::testInitFailing1
CI : nom = ""  $\wedge$  l = 10  $\wedge$  h = 10  $\wedge$  p = 10  $\wedge$  f = 10  $\wedge$  v = 10
Operation : G0  $\stackrel{def}{=}$  init(nom,l,h,p,f,v)
Oracle :
  nom(P0) == ""
  Une exception est levee

Cas de test : Gangster::testInitFailing2
CI : nom = "Slick"  $\wedge$  l = -1  $\wedge$  h = 10  $\wedge$  p = 10  $\wedge$  f = 10  $\wedge$  v = 10
Operation : G0  $\stackrel{def}{=}$  init(nom,l,h,p,f,v)
Oracle :
  l  $\leq$  0
  Une exception est levee

Cas de test : Gangster::testInitFailing3
CI : nom = "Slick"  $\wedge$  l = 10  $\wedge$  h = -1  $\wedge$  p = 10  $\wedge$  f = 10  $\wedge$  v = 10
Operation : G0  $\stackrel{def}{=}$  init(nom,l,h,p,f,v)
Oracle :
  h  $\leq$  0
  Une exception est levee

Cas de test : Gangster::testInitFailing4
CI : nom = "Slick"  $\wedge$  l = 10  $\wedge$  h = 10  $\wedge$  p = -1  $\wedge$  f = 10  $\wedge$  v = 10
Operation : G0  $\stackrel{def}{=}$  init(nom,l,h,p,f,v)
Oracle :
  p  $\leq$  0
  Une exception est levee

Cas de test : Gangster::testInitFailing5
CI : nom = "Slick"  $\wedge$  l = 10  $\wedge$  h = 10  $\wedge$  p = 10  $\wedge$  f = -1  $\wedge$  v = 10
Operation : G0  $\stackrel{def}{=}$  init(nom,l,h,p,f,v)
Oracle :
  f  $\leq$  0
  Une exception est levee

Cas de test : Gangster::testInitFailing6
CI : nom = "Slick"  $\wedge$  l = 10  $\wedge$  h = 10  $\wedge$  p = 10  $\wedge$  f = 10  $\wedge$  v = -1
Operation : G0  $\stackrel{def}{=}$  init(nom,l,h,p,f,v)
Oracle :
  v  $\leq$  0
  Une exception est levee
```

0.3 Bloc


```

Cas de test : Bloc::testInitWorking
CI : type = TYPE.VIDE  $\wedge$  o = null
Operation : B0  $\stackrel{def}{=}$  init(type,o)
Oracle :
    type(B0) = TYPE.VIDE
    objet(B0) = 0

Cas de test : Bloc::testInitFailing1
CI : type = TYPE.VIDE  $\wedge$  o = Objet::init("machin",100,0)
Operation : B0  $\stackrel{def}{=}$  init(type,o)
Oracle :
    type== TYPE.VIDE  $\wedge$  o  $\neq$  null
    Une exception est levee

Cas de test : Bloc::testInitFailing2
CI : type = TYPE.OBJET  $\wedge$  o = null
Operation : B0  $\stackrel{def}{=}$  init(type,o)
Oracle :
    type== TYPE.OBJET  $\wedge$  o == null
    Une exception est levee

Cas de test : Bloc::RetirerObjetWorking
CI : b = init(TYPE.OBJET, Objet::init("machin",100,0))
Operation : B0  $\stackrel{def}{=}$  retirerObjet(b)
Oracle :
    type(B0) = TYPE.VIDE
    objet(B0) = null

Cas de test : Bloc::RetirerObjetFailing
CI : b = init(TYPE.VIDE, null)
Operation : B0  $\stackrel{def}{=}$  retirerObjet(b)
Oracle :
    type(b)  $\neq$  TYPE.OBJET
    Une exception est levee

Cas de test : Bloc::PoserObjetWorking
CI : b = init(TYPE.VIDE, null)  $\wedge$  o = Objet::init("machin",100,0)
Operation : B0  $\stackrel{def}{=}$  poserObjet(b,o)
Oracle :
    type(B0) = TYPE.OBJET
    objet(B0) = o

Cas de test : Bloc::PoserObjetFailing
CI : b = init(TYPE.OBJET, Objet::init("machin",100,0))  $\wedge$  o = Objet::init("truc",10,0)
Operation : B0  $\stackrel{def}{=}$  poserObjet(b,o)
Oracle :
    type(b)  $\neq$  TYPE.VIDE
    Une exception est levee

```

0.4 Objet

0.5 Terrain

```

Cas de test : Terrain::testInitWorking
CI : p = 50, h = 100, l = 50
Operation : T0  $\stackrel{def}{=}$  init(l,h,p)

```

```

Oracle :
    largeur(T0) = l
    hauteur(T0) = h
    profondeur(T0) = p

Cas de test : Terrain::testInitFailing1
CI : p = 0, h = 100, l = 50
Operation : T0  $\stackrel{def}{=}$  init(l,h,p)
Oracle :
    p < 50
    Une exception est levee

Cas de test : Terrain::testInitFailing2
CI : p = 50, h = 100, l = 0
Operation : T0  $\stackrel{def}{=}$  init(l,h,p)
Oracle :
    l < 50
    Une exception est levee

Cas de test : Terrain::testInitFailing3
CI : p = 50, h = 10, l = 50
Operation : T0  $\stackrel{def}{=}$  init(l,h,p)
Oracle :
    h < 100
    Une exception est levee

Cas de test : Terrain::testInitFailing4
CI : p = 60, h = 100, l = 50
Operation : T0  $\stackrel{def}{=}$  init(l,h,p)
Oracle :
    (p%50) ≠ 0
    Une exception est levee

Cas de test : Terrain::testInitFailing5
CI : p = 50, h = 100, l = 60
Operation : T0  $\stackrel{def}{=}$  init(l,h,p)
Oracle :
    (l%50) ≠ 0
    Une exception est levee

Cas de test : Terrain::modifierBlocWorking
CI : t = init(500,100,500) ∧ b = Bloc::init(TYPE.VIDE, null)
Operation T0  $\stackrel{def}{=}$  modifierBloc(t,10,10, b)
Oracle :
    bloc(T0,10,10) = b

Cas de test : Terrain::modifierBlocFailing1
CI : t = init(500,100,500) ∧ b = Bloc::init(TYPE.VIDE, null)
Operation T0  $\stackrel{def}{=}$  modifierBloc(t,-1,10, b)
Oracle :
    x < 0
    Une exception est levee

Cas de test : Terrain::modifierBlocFailing2
CI : t = init(500,100,500) ∧ b = Bloc::init(TYPE.VIDE, null)
Operation T0  $\stackrel{def}{=}$  modifierBloc(t,1000,10, b)
Oracle :
    x > largeur(t)
    Une exception est levee

```

```

Cas de test : Terrain::modifierBlocFailing3
CI : t = init(500,100,500)  $\wedge$  b = Bloc::init(TYPE.VIDE, null)
Operation T0  $\stackrel{def}{=}$  modifierBloc(t, 10,-1, b)
Oracle :
    y < 0
    Une exception est levee

Cas de test : Terrain::modifierBlocFailing4
CI : t = init(500,100,500)  $\wedge$  b = Bloc::init(TYPE.VIDE, null)
Operation T0  $\stackrel{def}{=}$  modifierBloc(t, 10,1000, b)
Oracle :
    y > profondeur(t)
    Une exception est levee

Cas de test : Terrain::modifierBlocFailing5
CI : t = init(500,100,500)  $\wedge$  b = null
Operation T0  $\stackrel{def}{=}$  modifierBloc(t, 10,1000, b)
Oracle :
    b == null
    Une exception est levee

```

0.6 Moteur de jeu

0.7 GestionCombat