Chapitre 1

Projet CPS : Spécifications de River City Ransom

Béatrice CARRE et Steven VAROUMAS

1.1 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**]

$$\text{est\_vaincu}(P) \overset{min}{=} \text{points\_de\_vie}(P) \leq 0$$
$$\text{est\_equipe\_perso}(P) \overset{min}{=} \text{perso\_equipe}(P) \neq \text{null}$$
$$\text{est\_equipe\_objet}(P) \overset{min}{=} \text{objet\_equipe}(P) \neq \text{null}$$

[**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
est_visible(P) = true
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)) = 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)) =
$\begin{cases}$ force(P) - Objet : :bonus_force(objet_equipe(P)) si est_equipe_objet(P)
$\end{cases}$ force(P) sinon
objet_equipe(jeter(P)) = null

1.2 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)

1.3 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
```

1.4 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) =^min bonus_force > 0
      est_de_valeur(O) =^min valeur_marchande > 0
      est_equipable(O) =^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
```

1.5 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 $0 \leq x \leq$ largeur $\wedge\ 0 \leq y \leq$ profondeur

Constructors :

init : int × int × int → [Terrain]
pre init(largeur, hauteur, prof) require largeur > 50 ∧ hauteur > 10 ∧ prof > 50 ∧ largeur%50=0 ∧ profondeur%50=0

Operators :

modifier_bloc : [Terrain] × int × int × Bloc → [Terrain]
pre bloc(T, x, y, b) require $0 \leq x \leq$ largeur $\wedge\ 0 \leq y \leq$ profondeur $\wedge$ b $\neq$ 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

1.6 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
Constructors :
init : ∅ → [MoteurJeu]
Operators :
pasJeu : [MoteurJeu] × COMMANDE × COMMANDE → [MoteurJeu]
pre pasJeu(M,comAlex,comRyan) require : ¬estFini(M)
Observations :
[**Invariants**]

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

$$\text{resultat(M)} \overset{min}{=} \begin{cases} \text{ALEXGAGNANT} & \text{si } \neg\text{Personnage : :estVaincu(GestionCombat : :alex(combat(M)))} \\ & \wedge\ \text{Gangster : :estVaincu(GestionCombat : :slick(combat(M)))} \\ & \wedge\ \text{Personnage : :estVaincu(GestionCombat : :ryan(combat(M)))} \\ \text{RYANGAGNANT} & \text{si } \neg\text{Personnage : :estVaincu(GestionCombat : :ryan(combat(M)))} \\ & \wedge\ \text{Gangster : :estVaincu(GestionCombat : :slick(combat(M)))} \\ & \wedge\ \text{Personnage : :estVaincu(GestionCombat : :alex(combat(M)))} \\ \text{DEUXGAGNANTS} & \text{si } \neg\text{Personnage : :estVaincu(GestionCombat : :ryan(combat(M)))} \\ & \wedge\ \text{Gangster : :estVaincu(GestionCombat : :slick(combat(M)))} \\ & \wedge\ \neg\text{Personnage : :estVaincu(GestionCombat : :alex(combat(M)))} \\ \text{SLICKGAGNANT} & \text{si Personnage :estVaincu(GestionCombat : :ryan(combat(M)))} \\ & \wedge\ \neg\text{Gangster : :estVaincu(GestionCombat : :slick(combat(M)))} \\ & \wedge\ \text{Personnage : :estVaincu(GestionCombat : :alex(combat(M)))} \\ \text{NULLE} & \text{sinon} \end{cases}$$

```
  [init]
combat(init()) = GestionCombat::init()
  [pasJeu]
combat(pasJeu(M,cA,cR)) = GestionCombat::gerer(combat(M), cA, cR)
```

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

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

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

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

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

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

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

$$actionGangster(G,g) = RIEN \text{ si } estGele(G,g) \vee estVaincu(G,g)$$

[init]

$$terrain(init()) = Terrain::init(1000,1000,1000)$$
$$alex(init()) = Personnage::init("Alex",10,10,10,100,100,0)$$
$$ryan(init()) = Personnage::init("Ryan",10,10,10,100,100,0)$$
$$slick(init()) = Gangster::init("Slick",10,10,10,100,100,0)$$
$$gangsters(init()) = \{g = Personnage::init("noname",10,10,10,100,100,0)\},\ \forall\ g \in gangsters(G)$$
$$actionGangster(G,g) = RIEN\ \forall\ g \in gangsters(G)$$
$$estGele(init(),\ s) = false$$
$$collisionGauche(init(),p1,p2) = false$$
$$collisionDroite(init(),p1,p2) = false$$
$$collisionDevant(init(),p1,p2) = false$$
$$collisionDerriere(init(),p1,p2) = false$$
$$collisionDessous(init(),p1,p2) = false$$
$$collisionDessus(init(),p1,p2) = false$$
$$collision(init(),p1,p2) = false$$
$$estFrappe(init(),\ s) = false$$
$$posX(init(),alex(G)) < 50$$
$$posX(init(),slick(G)) > Terrain::largeur(terrain(G))-50$$
$$posX(init(),ryan(G)) < 50$$
$$posZ(init(),p) = 0$$

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

Bloc::type(Terrain:bloc(terrain(G),posX(init(),alex(G)),posY(init(),alex(G)),posZ(init(),alex(G))))≠FOSSE
Bloc::type(Terrain:bloc(terrain(G),posX(init(),ryan(G)),posY(init(),ryan(G)),posZ(init(),ryan(G))))≠FOSSE

[**gerer**]
posX(gerer(G,cA,cR),alex(G)) =
{ min(posX(G,alex(G)) + 10,Terrain :largeur(terrain(G)))
si cA = DROITE∨ cA = SAUTDROITE∧ ¬Personnage : :estVaincu(alex(G)) ∧ ¬estGele(G,alex(G))
∧ ¬collisionDroite(alex(G),p) ∀ p ∈ gangster ∨ p = slick(G)
max(posX(G,alex(G)) - 10,0)
si cA = GAUCHE ∨ cA = SAUTGAUCHE ∧ ¬Personnage : :estVaincu(alex(G)) ∧ ¬estGele(G,alex(G))
∧ ¬collisionGauche(alex(G),p) ∀ p ∈ gangster ∨ p = slick(G)
posX(G,alex(G)) sinon

posY(gerer(G,cA,cR),alex(G)) =
{ min(posY(G,alex(G)) + 10,Terrain :profondeur(terrain(G)))
si cA = HAUT ∨ cA = SAUTHAUT ∧ ¬Personnage : :estVaincu(alex(G)) ∧ ¬estGele(G,alex(G)) ∧ ¬collisionDerriere(alex(G),p) ∀ p ∈ gangster ∨ p = slick(G)
max(posY(G,alex(G)) - 10,0)
si cA = BAS ∨ cA = SAUTBAS ∧ ¬Personnage : :estVaincu(alex(G)) ∧ ¬estGele(G,alex(G)) ∧ ¬collisionDevant(alex(G),p) ∀ p ∈ gangster ∨ p = slick(G)
posY(G,alex(G)) sinon

posZ(gerer(G,cA,cR),alex(G)) =
{ 10 si cA = SAUT ∨ cA = SAUTBAS ∨ cA = SAUTHAUT ∨ cA = SAUTDROITE∨ cA = SAUTGAUCHE
∧ ¬Personnage : :estVaincu(alex(G)) ∧ ¬estGele(G,alex(G)) ∧ ¬collisionDessus(alex(G),p) ∀ p ∈ gangster ∨ p = slick(G)
pos(G,alex(G)) si estGele(G,alex(G))
0 Sinon

alex(gerer(G,cA,cR)) =
{ - Personnage : :jeter(alex(G))
si cA = JETER ∧ ¬Personnage : :estVaincu(alex(G)) ∧ ¬estGele(G,alex(G)) ∧ Bloc : :type(Terrain : :bloc(terrain(G),posX(G,alex(G)),posY(G,alex(G))))=VIDE
- Personnage : :ramasser_objet(alex(G), Bloc : :objet(Terrain : :bloc(terrain(G), posX(alex(G)),posY(alex(G)))))
si cA = RAMASSER ∧ posZ(alex(G))=0 ∧ ¬Personnage : :estVaincu(alex(G)) ∧ ¬estGele(G,alex(G))
- Personnage : :ramasser_perso(alex(G), p)
si collision(G,alex(G), p) ∧ cA = RAMASSER ∧ ¬Personnage : :estVaincu(alex(G)) ∧ ¬estGele(G,alex(G))
- Personnage : :ramasser_argent(alex(G), Bloc : :objet(Terrain : :bloc(terrain(G), posX(alex(G)),posY(alex(G)))))
si cA = RAMASSER ∧ posZ(alex(G))=0 ∧ ¬Personnage : :estVaincu(alex(G)) ∧ ¬estGele(G,alex(G))
- Personnage : :retrait_vie(alex(G), Personnage : :force(p))
si collision(G,alex(G),p) ∧ actionGangster(G,p) = FRAPPER ∧ ¬Personnage : :estVaincu(alex(G)) ∧ ¬estGele(G,alex(G))
- Personnage : :retrait_vie(alex(G), Personnage : :points_de_vie(alex(G))
si Bloc : :type(Terrain : :bloc(terrain(G), posX(alex(G)), posY(alex(G)),posZ(alex(G)))) = FOSSE ∧ ¬Personnage : :estVaincu(alex(G)) ∧ ¬estGele(G,alex(G))
- alex(G) Sinon

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

$$posX(gerer(G,cA,cR),p) = posX(G,alex(G)) \text{ si } cA = JETER \wedge Personnage::perso\_equipe(alex(G)) = p \wedge \neg Personnage::estVaincu(alex(G)) \text{ } posX(G,p) \text{ sinon}$$

$$posY(gerer(G,cA,cR),p) = posY(G,alex(G)) \text{ si } cA = JETER \wedge Personnage::perso\_equipe(alex(G)) = p \wedge \neg Personnage::estVaincu(alex(G)) \text{ } posY(G,p) \text{ sinon}$$

$$posZ(gerer(G,cA,cR),p) = 0 \text{ si } cA = JETER \wedge Personnage::perso\_equipe(alex(G)) = p \wedge \neg Personnage::estVaincu(alex(G)) \text{ } posZ(G,p) \text{ sinon}$$

TODO SLICK AND GANGSTERS AND RYAN

$$terrain(gerer(G,cA,cR)) =$$
- $Bloc::retirerObjet(Terrain::bloc(terrain(G), posX(alex(G)), posY(alex(G)))) \text{ si } cA = RAMASSER \wedge \neg Personnage::estVaincu(alex(G))$
- $Bloc::poserObjet(Terrain::bloc(terrain(G), posX(alex(G)), posY(alex(G)))), Personnage:objet\_equipe(alex(G))) \text{ si } cA = JETER \wedge Personnage::est\_equipe\_objet(alex(G)) = true \wedge \neg Personnage::estVaincu(alex(G))$
- $Bloc::retirerObjet(Terrain::bloc(terrain(G), posX(ryan(G)), posY(ryan(G)))) \text{ si } cR = RAMASSER \wedge \neg Personnage::estVaincu(ryan(G))$
- $Bloc::poserObjet(Terrain::bloc(terrain(G), posX(ryan(G)), posY(ryan(G)))), Personnage:objet\_equipe(ryan(G))) \text{ si } cR = JETER \wedge Personnage::est\_equipe\_objet(ryan(G)) = true \wedge \neg Personnage::estVaincu(ryan(G))$
- $terrain(G)$ sinon