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
```

```
depot_vie : [Personnage] × int → [Personnage]
   pre depot_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 // pour ne pas avoir une somme negative
depot_argent : [Personnage] × int → [Personnage]
   pre depot_argent(P,s) require ¬est_vaincu(P) ∧ s>0
ramasser_objet : [Personnage] × Object → [Personnage]
   pre ramasser_objet(P,o) require ¬est_vaincu(P) ∧ ¬est_equipe_objet(P)
       ∧  ¬est_equipe_perso(P)
     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]

$$\texttt{est\_vaincu(P)} \overset{min}{=} \texttt{points\_de\_vie(P)} \leq 0$$

$$\texttt{est\_equipe\_perso(P)} \overset{min}{=} \texttt{perso\_equipe(P)} \neq \texttt{null}$$

$$\texttt{est\_equipe\_objet(P)} \overset{min}{=} \texttt{objet\_equipe(P)} \neq \texttt{null}$$

```
[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)) = points_de_vie(P) - s

[depot_vie]
   points_de_vie(depot_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) si Objet::est_equipable(objet)
        { force(P) sinon

    somme_d_argent(ramasser_objet(P,objet)) =
        { somme_d_argent(P) + Objet::valeur_marchande(objet) si Objet::est_de_valeur(objet)
        { somme_d_argent(P) sinon

[ramasser_perso]
    perso_equipe(ramasser_perso(P,perso)) = perso

[jeter]
    perso_equipe(jeter(P)) = null
    force(jeter(P)) =
        { force(P) - Objet::bonus_force(objet_equipe(P))
        {  si est_equipe_objet(P) ∧ Objet::est_equipable(objet_equipe(P))
        { force(P) sinon
    objet_equipe(jeter(P)) = null
```

1.2 Gangster

```
service: Gangster
Refine : Personnage

Constructors :

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

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=OBJ∧ ∧ o≠null )
Operators :
      retirerObjet : [Bloc] → [Bloc]
            pre retirerObjet(B) require type(B)=OBJ∧
      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 × int → Bloc
      pre bloc( T, i, j, k) require 0 ≤ i ≤ largeur ∧ 0 ≤ j ≤ hauteur∧ 0
         ≤ k ≤ profondeur

Constructors :

   init : int × int× int → [Terrain]
      pre init(largeur, hauteur, prof) require largeur > 0 ∧ hauteur > 0
         ∧ prof > 0

Operators :

   modifier_bloc : [Terrain] × int × int × int × Bloc → [Terrain]
      pre bloc( T, i, j, k, b) require 0 ≤ i ≤ largeur ∧ 0 ≤ j ≤ hauteur
         ∧ 0 ≤ k ≤ 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, z) ≠ NULL

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

$$\texttt{estFini(M)} \overset{min}{=} \begin{cases} \texttt{(Personnage:: estVaincu(GestionCombat::alex(combat(M})))} \\ \texttt{} \wedge \texttt{ Personnage::estVaincu(GestionCombat::ryan(combat(M))))} \\ \texttt{} \vee \texttt{ Gangster::estVaincu(GestionCombat::slick(combat(M))} \end{cases}$$

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

```
[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,
   FRAPPE, SAUT, SAUTHAUT, SAUTDROIT, SAUTGAUCHE, SAUTBAS,  RAMASSER, JETER
   }

Observators :
      terrain : [GestionCombat] → Terrain
      alex : [GestionCombat] → Personnage
      ryan : [GestionCombat] → Personnage
      slick : [GestionCombat} → Gangster
      gangsters :  [GestionCombat] → Set<Gangster>
      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)
```

```
    positionX : [GestionCombat] × Personnage → int
          pre positionX(G, perso) require perso = alex(G) ∨ perso = ryan(
             G) ∨ perso = slick(G)  ∨ perso ∈ gangsters(G)
    positionY : [GestionCombat] × Personnage → int
          pre positionY(G, perso) require perso = alex(G) ∨ perso = ryan(
             G) ∨ perso = slick(G)  ∨ perso ∈ gangsters(G)
    positionZ : [GestionCombat] × Personnage → int
          pre positionZ(G, perso) require perso = alex(G) ∨ perso = ryan(
             G) ∨ perso = slick(G)  ∨ perso ∈ gangsters(G)
    collision : [GestionCombat] × Personnage × Personnage → boolean
          pre collision(G, perso1, perso2) require
          (perso1 = alex(G) ∧ perso2 = ryan(G))
          ∨ (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 <= positionX(G,s) <= Terrain::largeur(terrain)
       0 <= positionY(G,s) <= Terrain::profondeur(terrain)
       0 <= positionZ(G,s) <= Terrain::hauteur(terrain)
       collision(G,perso1,perso2) =min= collision(G,perso2,perso1)
       collision(G,p1,p2) =min= A FAIRE

    [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("???",10,10,10,100,100,0)}, ∀
     g ∈ gangsters(G)
  estGele(init(), s) = false
  collision(p1,p2) = false
  estFrappe(init(), s) = false
  positionX(init(),alex(G)) < 50
  positionX(init(),slick(G)) > Terrain::largeur(terrain(G))-50
  positionX(init(),ryan(G)) < 50
  positionZ(init(),p) = 0

  Bloc::type(Terrain:bloc(terrain(G),positionX(init(),g),positionY(init(),
     g),positionZ(init(),g))) = VIDE ∀ g ∈ gangsters(G)
```

```
Bloc::type(Terrain:bloc(terrain(G),positionX(init(),slick(G)),positionY(
   init(),slick(G)),positionZ(init(),slick(G)))) = VIDE
Bloc::type(Terrain:bloc(terrain(G),positionX(init(),alex(G)),positionY(
   init(),alex(G)),positionZ(init(),alex(G))))≠FOSSE
Bloc::type(Terrain:bloc(terrain(G),positionX(init(),ryan(G)),positionY(
   init(),ryan(G)),positionZ(init(),ryan(G))))≠FOSSE


 [gerer]
 positionX(gerer(G,cA,cR),alex(G)) =
    { positionX(G,alex(G)) + 10   si cA = DROIT ∨ cA = SAUTDROIT
    { positionX(G,alex(G)) - 10   si cA = GAUCHE ∨ cA = SAUTGAUCHE
    { positionX(G,alex(G))        sinon

 positionY(gerer(G,cA,cR),alex(G)) =
    { positionY(G,alex(G)) + 10   si cA = HAUT ∨ cA = SAUTHAUT
    { positionY(G,alex(G)) - 10   si cA = BAS ∨ cA = SAUTBAS
    { positionY(G,alex(G))        sinon

positionZ(gerer(G,cA,cR),alex(G)) =
    { 10  si cA = SAUT ∨ cA = SAUTBAS ∨ cA = SAUTHAUT ∨ cA = SAUTDROIT ∨ cA = SAUTGAUCHE
    { 0   Sinon

 positionX(gerer(G,cA,cR),ryan(G)) =
    { positionX(G,ryan(G)) + 10   si cR = DROIT ∨ cR = SAUTDROIT
    { positionX(G,ryan(G)) - 10   si cR = GAUCHE ∨ cR = SAUTGAUCHE
    { positionX(G,ryan(G))        sinon


 positionY(gerer(G,cA,cR),ryan(G)) =
    { positionY(G,ryan(G)) + 10   si cR = HAUT ∨ cR = SAUTHAUT
    { positionY(G,ryan(G)) - 10   si cR = BAS ∨ cR = SAUTBAS
    { positionY(G,ryan(G))        sinon

positionZ(gerer(G,cA,cR),ryan(G)) =
    { 10  si cR = SAUT ∨ cR = SAUTBAS ∨ cR = SAUTHAUT ∨ cR = SAUTDROIT ∨ cR = SAUTGAUCHE
    { 0   Sinon


alex(gerer(G,cA,cR)) =
{ - Personnage::jeter(alex(G)) si cA = JETER
{ - Personnage::ramasser_objet(alex(G), Bloc::objet(Terrain::bloc(terrain(G), positionX(alex(G)),
{ positionY(alex(G)),positionZ(alex(G)))) si cA = RAMASSER
{ - Personnage::ramasser_perso(alex(G), p) si collision(alex(G), p) ∧ cA = RAMASSER
{ - alex(G) Sinon

positionX(gerer(G,cA,cR),p) =
   positionX(G,alex(G))+10 si cA = JETER  ∧ Personnage::perso_equipe(
      alex()) = p
```

```
    positionX(G,p) sinon
positionY(gerer(G,cA,cR),p) =
    positionY(G,alex(G)) si cA = JETER ∧ Personnage::perso_equipe(alex())
        = p
    positionY(G,p) sinon
positionZ(gerer(G,cA,cR),p) =
    0 si cA = JETER ∧ Personnage::perso_equipe(alex()) = p
    positionZ(G,p) sinon

ryan(gerer(G,cA,cR)) =
- Personnage::jeter(ryan(G)) si cR = JETER
- Personnage::ramasser_objet(ryan(G), Bloc::objet(Terrain::bloc(terrain(G), positionX(ryan(G)),
positionY(ryan(G)),positionZ(ryan(G)))) si cR = RAMASSER
- Personnage::ramasser_perso(ryan(G), p) si collision(ryan(G), p) ∧ cR = RAMASSER
- ryan(G) Sinon
positionX(gerer(G,cA,cR),p) =
    positionX(ryan(G))+10 si cR = JETER  ∧ Personnage::perso_equipe(ryan
       ()) = p
    positionX(G,p) sinon

positionY(gerer(G,cA,cR),p) =
    positionY(ryan(G)) si cR = JETER ∧ Personnage::perso_equipe(ryan()) =
        p
    positionY(G,p) sinon

positionZ(gerer(G,cA,cR),p) =
    0 si cR = JETER ∧ Personnage::perso_equipe(ryan()) = p
    positionZ(G,p) sinon


terrain(gerer(G,cA,cR)) =
- Bloc::retirerObjet(Terrain::bloc(terrain(G), positionX(alex(G)),
    positionY(alex(G)),positionZ(alex(G))) si cA = RAMASSER
- Bloc::poserObjet(Terrain::bloc(terrain(G), positionX(alex(G)),
    positionY(alex(G)),positionZ(alex(G))), Personnage:objet_equipe(alex
    ()) si cA = JETER ∧ Personnage::est_equipe_objet(alex()) = true
- Bloc::retirerObjet(Terrain::bloc(terrain(G), positionX(ryan(G)),
    positionY(ryan(G)),positionZ(ryan(G))) si cR = RAMASSER
- Bloc::poserObjet(Terrain::bloc(terrain(G), positionX(ryan(G)),
    positionY(ryan(G)),positionZ(ryan(G))), Personnage:objet_equipe(ryan
    ()) si cR = JETER ∧ Personnage::est_equipe_objet(ryan()) = true
- terrain(G) sinon // Faut il faire aussi les deux ?????
```