Chapitre 1

Projet CPS : Spécifications de River City Ransom

Béatrice CARRE et Steven VAROUMAS

1.1 Le service Personnage

```
service: Personnage
use : Objet, Bloc
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 ≠ "" ∧ 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]
      est_vaincu(P) =^min points_de_vie(P) ≤ 0
      est_equipe_perso(P) =^min perso_equipee(P) ≠ null
      est_equipe_objet(P) =^min objet_equipee(P) ≠ null

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

   [retrait_vie]
```

```
      points_de_vie(retrait_vie(P,s)) = 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

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

   [jeter]
      perso_equipe(jeter(P)) = null
      objet_equipe(jeter(P)) = null
```

1.2 Gangster

```
service: Gangster
Refine : Personnage
```

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 × 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]
estFini(M) =min {  (Personnage:: estVaincu(GestionCombat::alex(combat(M)))
                ∧ Personnage::estVaincu(GestionCombat::ryan(combat(M))))
                ∨ Gangster::estVaincu(GestionCombat::slick(combat(M))

resultat(M)
   =min {  ALEXGAGNANT   si Personnage::!estVaincu(GestionCombat::alex(combat(M)))
                         ∧ Gangster::estVaincu(GestionCombat::slick(combat(M)))
                         ∧ Personnage::estVaincu(GestionCombat::ryan(combat(M)))

           RYANGAGNANT   si Personnage::!estVaincu(GestionCombat::ryan(combat(M)))
                         ∧ Gangster::estVaincu(GestionCombat::slick(combat(M)))
                         ∧ Personnage::estVaincu(GestionCombat::alex(combat(M)))

           DEUXGAGNANTS  si Personnage::!estVaincu(GestionCombat::ryan(combat(M)))
                         ∧ Gangster::estVaincu(GestionCombat::slick(combat(M)))
                         ∧ Personnage::!estVaincu(GestionCombat::alex(combat(M)))

           SLICKGAGNANT  si Personnage: estVaincu(GestionCombat::ryan(combat(M)))
                         ∧ Gangster::!estVaincu(GestionCombat::slick(combat(M)))
                         ∧ Personnage::estVaincu(GestionCombat::alex(combat(M)))

           NULLE         sinon

      [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
      gangsters :  [GestionCombat] → Set<Gangster>
      gangstersNoms : [GestionCombat] → Set<String>
      estGele : [GestionCombat] × String → boolean
            pre estGele(G, id) require id = "alex" ∨ id = "ryan"  ∨ id
               ∈ gangstersNoms(G)
      estFrappe : [GestionCombat] × String → boolean
            pre estFrappe(G, id) require id = "alex" ∨ id = "ryan"  ∨
               id ∈ gangstersNoms(G)
      positionX : [GestionCombat] × String → int
            pre positionX(G, id) require id = "alex" ∨ id = "ryan"  ∨
               id ∈ gangstersNoms(G)
      positionY : [GestionCombat] × String → int
            pre positionY(G, id) require id = "alex" ∨ id = "ryan"  ∨
               id ∈ gangstersNoms(G)
      positionZ : [GestionCombat] × String → int
            pre positionZ(G, id) require id = "alex" ∨ id = "ryan"  ∨
               id ∈ gangstersNoms(G)
      collision : [GestionCombat] × String × String → boolean
            pre collision(G, id1, id2) require
            (id = "alex" ∧ id = "ryan")
            ∨ (id1 = "alex" ∧ id2 = "slick")
            ∨ (id1 = "ryan" ∧ id2 = "slick")
            ∨ (id1 = "alex"∧ id2 ∈ gangstersNoms(G))
            ∨ (id1 = "ryan"∧ id2 ∈ gangstersNoms(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 = ?

      [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)
   gangsters(init()) = {Personnage::init(g,10,10,10,100,100,0) , ∀ g ∈
      gangstersNoms(G)}
   estGele(init(), s) = false ∀ s ∈ String
   estFrappe(init(), s) = false ∀ s ∈ String
   positionX(init(),"Alex") = 10
   positionY(init(), "Alex") = 10
   positionZ(init(), "Alex") = 0
   positionX(init(),"Ryan") = 10
   positionY(init(), "Ryan") = 30
   positionZ(init(), "Ryan") = 0
   positionX(init(),g) = random
   positionY(init(), g) = random
   positionZ(init(), g) = 0, ∀ g ∈ gangstersNoms(G)


    [gerer]
    positionX(gerer(G,cA,cR),"Alex") =
      positionX(G,"Alex") + 10 si cA = DROIT ∨ cA = SAUTDROIT
      positionX(G,"Alex") - 10 si cA = GAUCHE ∨ cA = SAUTGAUCHE
      positionX(G,"Alex") Sinon
    positionY(gerer(G,cA,cR),"Alex") =
      positionY(G,"Alex") + 10 si cA = HAUT ∨ cA = SAUTHAUT
      positionY(G,"Alex") - 10 si cA = BAS ∨ cA = SAUTBAS
      positionY(G,"Alex") Sinon
    positionZ(gerer(G,cA,cR),"Alex") =
      10 si cA = SAUT ∨ cA = SAUTBAS ∨ cA = SAUTHAUT ∨ cA = SAUTDROIT
         ∨ cA = SAUTGAUCHE
      0 Sinon

   positionX(gerer(G,cA,cR),"Ryan") =
      positionX(G,"Ryan") + 10 si cR = DROIT ∨ cR = SAUTDROIT
      positionX(G,"Ryan") - 10 si cR = GAUCHE ∨ cR = SAUTGAUCHE
      positionX(G,"Ryan") Sinon
```

```
 positionY(gerer(G,cA,cR),"Ryan") =
   positionY(G,"Ryan") + 10 si cR = HAUT ∨ cR = SAUTHAUT
   positionY(G,"Ryan") - 10 si cR = BAS ∨ cR = SAUTBAS
   positionY(G,"Ryan") Sinon
 positionZ(gerer(G,cA,cR),"Ryan") =
   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(T,
       positionX("Alex"), positionY("Alex"),positionZ("Alex"))) si
      cA = RAMASSER
   Personnage::ramasser_perso(alex(G), p) si collision("Alex", nom)
      et Personnage::nom(p) == nom
   alex(G) Sinon
```