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**]

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

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

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

[**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**]

$$objet\_equipe(ramasser\_objet(P,objet)) = objet$$

$$force(ramasser\_objet(P,objet)) = \begin{cases} force(P) + Objet::bonus\_force(objet) \text{ si } Objet::est\_equipable(objet) \\ force(P) \text{ sinon} \end{cases}$$

$$somme\_d\_argent(ramasser\_objet(P,objet)) = \begin{cases} somme\_d\_argent(P) + Objet::valeur\_marchande(objet) \text{ si } Objet::est\_de\_valeur(objet) \\ somme\_d\_argent(P) \text{ sinon} \end{cases}$$

[**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
use : enum ACTION{RIEN,FRAPPE,SAUTE,HAUT,BAS,GAUCHE,DROITE}
Observators :
  action : [Gangster]  → ACTION
    pre action(G) require ¬estVaincu(G)


Constructors :

  init : String × 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

Observations :
  [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
    action(init(n,l,h,p,f,v,a)) = RIEN
```

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]
```

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

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

[**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, 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)
      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)
      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))
```

$\vee$ (perso1 = ryan(G) $\wedge$ perso2 = slick(G))

Constructors:

init : $\emptyset \rightarrow$ [GestionCombat]

Operators :

gerer : [GestionCombat] $\times$ COMMANDE $\times$ COMMANDE $\rightarrow$ [GestionCombat]

Observations :

[**Invariants**]

0 <= posX(G,s) <= Terrain::largeur(terrain)
0 <= posY(G,s) <= Terrain::profondeur(terrain)
0 <= posZ(G,s) <= Terrain::hauteur(terrain)
collision(G,p1,p2) $\stackrel{min}{=}$ 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)}, $\forall$ g $\in$ gangsters(G)
estGele(init(), s) = false
collision(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))))$\neq$FOSSE
Bloc::type(Terrain:bloc(terrain(G),posX(init(),ryan(G)),posY(init(),ryan(G)),posZ(init(),ryan(G))))$\neq$FOSSE

[**gerer**]

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

$$\begin{cases} \text{posX(G,alex(G))} + 10 & \text{si cA} = \text{DROIT} \vee \text{cA} = \text{SAUTDROIT} \\ \text{posX(G,alex(G))} - 10 & \text{si cA} = \text{GAUCHE} \vee \text{cA} = \text{SAUTGAUCHE} \\ \text{posX(G,alex(G))} & \text{sinon} \end{cases}$$

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

$$\begin{cases} \text{posY(G,alex(G))} + 10 & \text{si cA} = \text{HAUT} \vee \text{cA} = \text{SAUTHAUT} \\ \text{posY(G,alex(G))} - 10 & \text{si cA} = \text{BAS} \vee \text{cA} = \text{SAUTBAS} \\ \text{posY(G,alex(G))} & \text{sinon} \end{cases}$$

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

$$\begin{cases} 10 & \text{si cA} = \text{SAUT} \vee \text{cA} = \text{SAUTBAS} \vee \text{cA} = \text{SAUTHAUT} \vee \text{cA} = \text{SAUTDROIT} \vee \text{cA} = \text{SAUTGAUCHE} \\ 0 & \text{Sinon} \end{cases}$$

$$posX(gerer(G,cA,cR),ryan(G)) = \begin{cases} posX(G,ryan(G)) + 10 & \text{si } cR = DROIT \vee cR = SAUTDROIT \\ posX(G,ryan(G)) - 10 & \text{si } cR = GAUCHE \vee cR = SAUTGAUCHE \\ posX(G,ryan(G)) & \text{sinon} \end{cases}$$

$$posY(gerer(G,cA,cR),ryan(G)) = \begin{cases} posY(G,ryan(G)) + 10 & \text{si } cR = HAUT \vee cR = SAUTHAUT \\ posY(G,ryan(G)) - 10 & \text{si } cR = BAS \vee cR = SAUTBAS \\ posY(G,ryan(G)) & \text{sinon} \end{cases}$$

$$posZ(gerer(G,cA,cR),ryan(G)) = \begin{cases} 10 & \text{si } cR = SAUT \vee cR = SAUTBAS \vee cR = SAUTHAUT \vee cR = SAUTDROIT \vee cR = SAUTGAUCHE \\ 0 & \text{Sinon} \end{cases}$$

$$alex(gerer(G,cA,cR)) = \begin{cases} \text{- Personnage : :jeter(alex(G)) si } cA = JETER \\ \text{- Personnage : :ramasser\_objet(alex(G), Bloc : :objet(Terrain : :bloc(terrain(G), posX(alex(G)),} \\ \text{posY(alex(G)),posZ(alex(G)))) si } cA = RAMASSER \\ \text{- Personnage : :ramasser\_perso(alex(G), p) si collision(alex(G), p)} \wedge cA = RAMASSER \\ \text{- Personnage : :retrait\_vie(alex(G), Personnage : :force(p)) si collision(alex(G),p)} \wedge \text{Gangster : :action(p)} = FRAPPER \\ \text{- alex(G) Sinon} \end{cases}$$

$$posX(gerer(G,cA,cR),p) = posX(G,alex(G))+10 \text{ si } cA = JETER \wedge Personnage::perso\_equipe(alex()) = p \; posX(G,p) \text{ sinon}$$
$$posY(gerer(G,cA,cR),p) = posY(G,alex(G)) \text{ si } cA = JETER \wedge Personnage::perso\_equipe(alex()) = p \; posY(G,p) \text{ sinon}$$
$$posZ(gerer(G,cA,cR),p) = 0 \text{ si } cA = JETER \wedge Personnage::perso\_equipe(alex()) = p \; posZ(G,p) \text{ sinon}$$

$$slick(gerer(G,cA,cR)) = \begin{cases} \text{- Gangster : :retrait\_vie(slick(G),Personnage : :force(h)) si collison(alex(G),slick(G))} \wedge cA = FRAPPER \\ \text{- Gangster : :retrait\_vie(slick(G),Personnage : :force(h)) si collison(ryan(G),slick(G))} \wedge cR = FRAPPER \\ \text{- slick(G) sinon} \end{cases}$$

$$ryan(gerer(G,cA,cR)) = \begin{cases} \text{- Personnage : :jeter(ryan(G)) si } cR = JETER \\ \text{- Personnage : :ramasser\_objet(ryan(G), Bloc : :objet(Terrain : :bloc(terrain(G), posX(ryan(G)),} \\ \text{posY(ryan(G)),posZ(ryan(G)))) si } cR = RAMASSER \\ \text{- Personnage : :ramasser\_perso(ryan(G), p) si collision(ryan(G), p)} \wedge cR = RAMASSER \\ \text{- Personnage : :retrait\_vie(ryan(G), Personnage : :force(p)) si collision(ryan(G),p)} \wedge \text{Gangster : :action(p)} = FRAPPER \\ \text{- ryan(G) Sinon} \end{cases}$$

$$posX(gerer(G,cA,cR),p) = posX(ryan(G))+10 \text{ si } cR = JETER \wedge Personnage::perso\_equipe(ryan()) = p \; posX(G,p) \text{ sinon}$$
$$posY(gerer(G,cA,cR),p) = posY(ryan(G)) \text{ si } cR = JETER \wedge Personnage::perso\_equipe(ryan()) = p \; posY(G,p) \text{ sinon}$$

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


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