Chapitre 1

Projet CPS : Spécifications de River City Ransom

Béatrice CARRE
Steven VAROUMAS

Introduction

Lien vers l'énoncé du projet : lien.

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 ≠ "" ∧ 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(P,chose) require ¬est_vaincu(P) ∧
            ¬est_equipe_objet(P) ∧ ¬est_equipe_perso(P)
    ramasser_perso : [Personnage] × Personnage → [Personnage]
        pre ramasser(P,chose) 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 // a
            verifier si ca ne "boucle" pas avec la precondition de
            la_chose_equipee ...
        est_equipe_objet(P) =^min objet_equipee(P) ≠ null // a
            verifier si ca ne "boucle" pas avec la precondition de
            la_chose_equipee ... // TOASK
    [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

1.3 Bloc

```
service : Bloc
use : Objet
types : enum TYPE{VIDE,FOSSE, PLEIN },
Observators :
      const type : [Bloc] → TYPE
      const objet : [Bloc] → Objet
      const perso : [Bloc] → Perso
Constructors :
      init : TYPE × Objet × perso → [Bloc]
            pre init(t,o,p) require
            (( ty=PLEIN ∨ ty=FOSSE ) ∧ o=null ∧ p=null)
            ∨ (ty=VIDE ∧¬( o=null∨p=null) ) // qu'il n'y ait pas les deux
                en meme temps.
Operators :
      prendreObjet : [Bloc] → [Bloc]
            pre prendreObjet(B) require type(B)=PLEIN ∧ (o≠null∨p≠null)
      poserObjet : [BLOC] × Objet → [Bloc]
            pre poserObjet(B,o) require type(B)=VIDE ∧ p≠null ∧
            objet(B)=null ∧ perso(B)=null
      prendrePerso : [Bloc] → [Bloc]
            pre prendrePerso(B) require type(B)=PLEIN ∧ (o≠null∨p≠null)
      poserPerso : [BLOC] × Personnage → [Bloc]
            pre poserPerso(B,p) require type(B)=VIDE ∧ p≠null
            ∧ objet(B)=null ∧ perso(B)=null
Observations :
      [init]
            type(init(t,o)) = ty
            objet(init(t,o)) = o
      [prendreObjet]
            type(prendreObjet(B)) = VIDE
      [poserObjet]
            type(poserObjet(B,o)) = PLEIN
            objet(poserObjet(B,o)) = o
      [prendrePerso]
            type(prendrePerso(B)) = VIDE
      [poserPerso]
            type(poserPerso(B,p)) = PLEIN
            objet(poserPerso(B,p)) = p
```

1.4 Objet

```
service : Objet
types : String, boolean, int, enum TRESOR {UNDOLLAR,CINQUANTECENTIMES,
    CHAINEDEVELO,POUBELLEMETALLIQUE,...} // a completer ??
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) )
                // comme ca on ne peut pas etre de valeur ET equipable
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 // meh
                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{LESDEUXGAGNANTS, RYANGAGNANT,
    ALEXGAGNANT, SLICKGAGNANT, NULLE},
        enum COMMANDE{RIEN, GAUCHE, DROITE, BAS, HAUT, FRAPPE, SAUT,
            RAMASSER, JETER}
Observators :
      estFini : [MoteurJeu] → boolean
      resultatFinal : [MoteurJeu] → RESULTAT
            pre resultatFinal(M) require estFini(M)
      combat : [MoteurJeu] → GestionCombat
Constructors :
      init : ∅ → [MoteurJeu]
Operators :
      pasJeu : [MoteurJeu] × COMMANDE × COMMANDE → [MoteurJeu]
            pre pasJeu(M, comRyan, comAlex) require :estFini(M)
Observations :
```

[**Invariants**]

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

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

[**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, COMMANDE

Observators :
      terrain: [GestionCombat] → Terrain
      alex : [GestionCombat] → Personnage
      ryan : [GestionCombat] → Personnage
      slick : [GestionCombat] → Gangster
      gangster : [GestionCombat] → {Gangster*} // Une liste de gangsters
          (les autres)
      estGele(GestionCombat] × String → boolean
            pre estGele(G, id) require id = "alex" ∨ id = "ryan" ∨ id = "
                slick" // et pour les autres gangsters ?
      estFrappe(GestionCombat] × String → boolean
            pre estFrappe(G, id) require id = "alex" ∨ id = "ryan" ∨ id =
                "slick" // idem
      positionX : [GestionCombat] × String → int
            pre positionX(G, id) require id = "alex" ∨ id = "ryan" ∨ id =
                "slick" // idem
      positionY : [GestionCombat] × String → int
            pre positionY(G, id) require id = "alex" ∨ id = "ryan" ∨ id =
                "slick" // idem
      positionZ : [GestionCombat] × String → int
            pre positionZ(G, id) require id = "alex" ∨ id = "ryan" ∨ id =
                "slick" // idem
      collision : [GestionCombat] × String × String → boolean
            pre collision(G, id1, id2) require (id = "alex" ∧ id = "ryan")
            (∨ id = "slick" ∧ id = "alex")
            (∨ id = "ryan"
            ∧ id = "slick") // idem

      Constructors:
      init : /emptyset → [GestionCombat]

Operators :
      gerer : [GestionCombat] × COMMANDE × COMMANDE → [GestionCombat]

Observations :
      [Invariants]

      [init]
            terrain(init()) = Terrain::init(256,256,256)
            alex(init()) = Personnage::init()
```