Projet CPS : Street Fighter

Couassi-blé Yannick-Alain
Hourcade Hugo

Rapport

Master Informatique

UPMC SORBONNE UNIVERSITÉS

Science et Technologie du Logiciel
Université Pierre & Marie Curie
Paris

Mai 2017

Table des matières

Chapitre 1

Introduction

Dans ce rapport, nous exposons les différentes spécifications formelles des services que nous avons utilisés. Un manuel d'utilisation de notre jeu et finalement des tests MBT et quelques explications de nos choix d'implémentations. Un build XML est fournie avec les cibles ant suivantes :

1. **compile** pour compiler les sources
2. **run** pour lancer le jeu
3. **test1** pour effectuer les tests junit MBT pour la hitbox
4. **bug** pour lancer une implementation bugger du moteur de jeu

Chapitre 2

Manuel d'utilisation

Notre jeu comporte 2 personnages voir figure 2.1, jackie et elsa avec des techniques différentes. Ils ont des sprites pour rendre le jeu plus interactif, mais sont dans 2 hitbox de couleurs rouge et bleu respectives. IL est jouable à 2 sur un clavier.

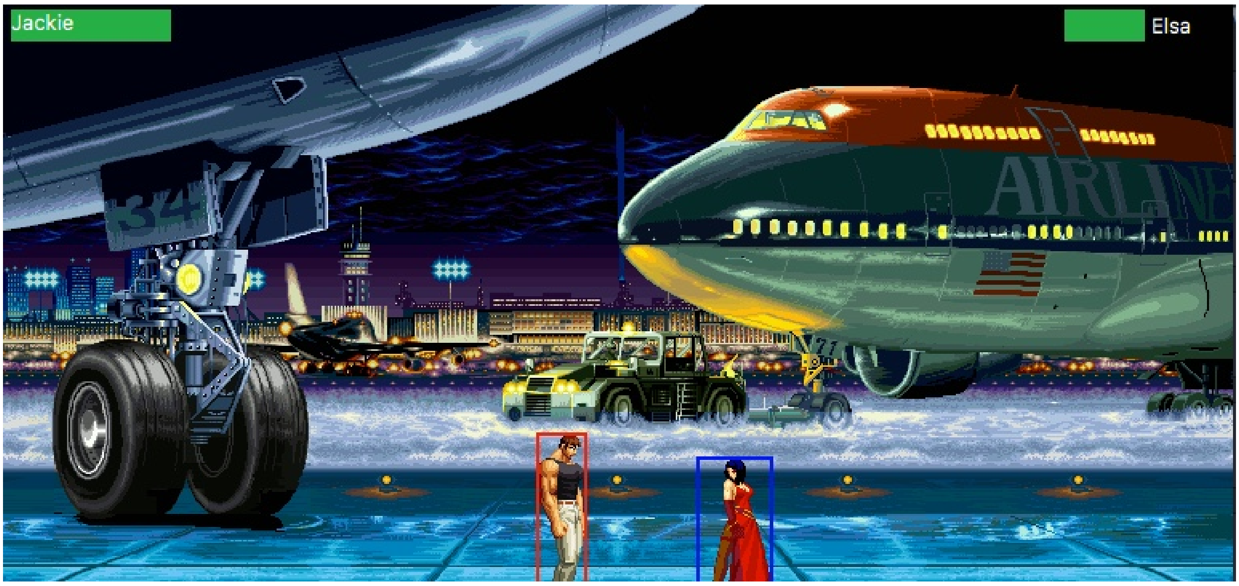

Figure 2.1 – Debut d'une partie

`[fig-init]`

Voici la liste des touches clavier pour contrôler jackie et elsa.

Jackie :

1. RIGHT : direction droite pour un déplacement vers la droite

2. LEFT : direction gauche pour un déplacement vers la gauche
3. DOWN : direction vers le bas pour un accroupissement
4. UP : direction vers le haut pour un saut
5. NUMPAD1 : la touche 1 du pavé numérique pour se protéger d'une attaque
6. NUMPAD2 : la touche 2 du pavé numérique pour effectuer la première technique d'attaque (coup de poing)
7. NUMPAD3 : la touche 3 du pavé numérique pour etfectuer la deuxième attaque (coup de pied)
8. NUMPAD5 : la touche 5 du pavé numérique pour effectuer la troisième attaque spéciale (uppercut)

Elsa

1. D : pour un déplacement vers la droite
2. Q : pour un déplacement vers la gauche
3. S : pour un accroupissement
4. Z : pour un saut
5. H : pour se protéger d'une attaque
6. J : pour effectuer la première technique d'attaque (coup de poing)
7. K : pour effectuer la deuxième attaque (coup de pied)
8. I : pour effectuer la troisième attaque spéciale

PS : Pour une adaptation des touches a votre clavier en cas d'incompatibilité, se rendre dans la classe MainGame et dans la méthode moveRecOnKeyPress() afin de changer les réglages du keyEvent.

Chapitre 3

Spécifications formelles

3.1 HitBox

```
Service:  Hitbox
Types:   bool, int
Observators:  PositionX: [Hitbox] -> int
                             PositionY: [Hitbox] -> int
                             Length: [HitBox] -> int
                             Height: [HitBox] -> int
                             BelongsTo: [Hitbox]  int  int -> bool
                             CollidesWith: [Hitbox]  Hitbox -> bool
                             EqualsTo: [Hitbox]  Hitbox -> bool
Constructors:  init: int x int x int x int -> [HitBox]
               pre init(x,y,h,l) requires h>=0 && l>=0
Operators:  MoveTo: [Hitbox]  int  int -> [Hitbox]
              SetHeight: [HitBox] x int -> [HitBox]
            SetLength: [HitBox] x int -> [HitBox]
Observations:
               [invariant]:
                             ( PositionX(H)<PositionX(H1) ) ^ ( PositionY(H)<PositionY(H1) ) =>
                                  CollidesWith(H,H1) = ( PositionX(H1)-PositionX(H) < Length(H) ) ^ (
                                  PositionY(H1)-PositionY(H) <Height(H) )

                             ( PositionX(H)<PositionX(H1) ) ^ ( PositionY(H)>PositionY(H1) ) =>
                                  CollidesWith(H,H1) = ( PositionX(H1)-PositionX(H) < Length(H) ) ^ (
                                  PositionY(H)-PositionY(H1) <Height(H1) )

                             ( PositionX(H)>PositionX(H1) ) ^ ( PositionY(H)<PositionY(H1) ) =>
                                  CollidesWith(H,H1) = ( PositionX(H)-PositionX(H1) < Length(H1) ) ^ (
                                  PositionY(H1)-PositionY(H) <Height(H) )
```

```
                  ( PositionX(H)>PositionX(H1) ) ^ ( PositionY(H)>PositionY(H1) ) =>
                      CollidesWith(H,H1) = ( PositionX(H)-PositionX(H1) < Length(H1) ) ^ (
                      PositionY(H)-PositionY(H1) <Height(H1) )

                  EqualsTo(H,H1) = (PositionX(H)=PositionX(H1)) ^ (PositionY(H)=PositionY(H1
                      )) ^ (Height(H)=Height(H1)) ^ (Length(H)=Length(H1))

                  BelongsTo(H,x,y) = (x-PositionX(H)>0)&&(x-PositionX(H)<Length(H))&&(y-
                      PositionY(H)>0)&&(y-PositionY(H)<Height(H))
[init]:
                  PositionX(init(x,y)) = x
                  PositionY(init(x,y)) = y
                  Length(init(x,y,l,h)) = l
                  Height(init(x,y,l,h)) = h
[MoveTo]:
                  PositionX(MoveTo(H,x,y)) = x
                  PositionY(MoveTo(H,x,y)) = y
                  forAll u,v:int  int, BelongsTo(MoveTo(H,x,y),u,v) = Belongsto(H,u-(x-PositionX
                      (H)),v-(y-PositionY(H))
                  Length(MoveTo(H,x,y)) = Length(H)
                  Height(MoveTo(H,x,y)) = Height(H)
[SetHeight]:
                  Height(SetHeight(H,h)) = h
                  Length(SetHeight(H,h)) = Length(H)
                  PositionX(SetHeight(H,h)) = PositionX(H)
                  PositionY(SetHeight(H,h)) = PositionY (H)
[SetLength]:
                  Length(SetLength(H,l)) = l
                  Height(SetLength(H,l)) = Height(H)
                  PositionX(SetLength(H,l)) = PositionX(H)
                  PositionY(SetLength(H,l)) = PositionY(H)
```

3.2 Character

```
Service:   Character
Types:     bool, int, Commande
Observators:  positionX: [Character] −> int
                        positionY: [Character] −> int
                        engine: [Character] −> Engine
                        charBox: [Character] −> HitBox
                        life: [Character] −> int
                        const speed: [Character] −> int
                        faceRight: [Character] −> bool
                        dead: [Character] −> bool
Constructors:  init: int  int  bool  Engine x Hitbox −> [Character]
                              pre init(l,s,f,e,h) requires l > 0 && s > 0
Operators:  moveLeft: [Character] −> [Character]
                        moveRight: [Character] −> [Character]
                        switchSide: [Character] −> [Character]
                        step: [Character]  Commande −> [Character]
                                  pre step() requires !dead
Observations:
            [invariant]:
                        positionX(C) > 0 && positionX(C) < Engine:: width(engine) − HitBox
                            :: width(charBox)

                        positionY(C) > 0 && positionY(C) < Engine:: height(engine) −
                            HitBox:: height(charBox)

                        dead(C) = !(life(C) > 0)
            [init]:
                        life(init(l, s, f, e)) = l
                        speed(init(l, s, f, e)) = s
                        faceRight(init(l, s, f, e)) = f
                        engine(init(l, s, f, e)) = e
                        charbox(init(l, s, f, e)) = h
            [moveLeft]:
                        (exists i, player(engine(C), i) != C => collisionwith(charBox(
                            moveLeft(C)), charBox(player(engine(C), i)))) => positionX(
                            moveLeft(C)) = positionX(C)
                        positionX(C) >= speed(C) && (forAll i, player(engine(C), i) != C =>
                             !collisionwith(charBox(moveLeft(C)), charBox(player(engine(C),
                            i)))) => positionX(moveLeft(C)) = positionX(C) − speed(C)
                        positionX(C) < speed(C) && (forAll i, player(engine(C), i) != C => !
                            collisionwith(charBox(moveLeft(C)), charBox(player(engine(C), i
                            )))) => positionX(moveLeft(C)) = 0
                        faceRight(moveLeft(C)) = faceRight(C)
                        life(moveLeft(C)) = life(C)
```

positionY(moveLeft(C)) = positionY(C)

[moveRight]:

(exists i, player(engine(C), i) != C => collisionwith(charBox(moveRight(C)), charBox(player(engine(C), i)))) => positionX(moveRight(C)) = positionX(C)

Engine::width(engine) − Hitbox:: Length(charBox(C)) − positionX(C) >= speed(C) && (forAll i, player(engine(C), i) != C => ! collisionwith(charBox(moveRight(C)), charBox(player(engine(C), i)))) => positionX(moveRight(C)) = positionX(C) + speed(C)

Engine::width(engine) − HitBox:: Length(charBox(C)) − positionX(C) < speed(C) && (forAll i, player(engine(C), i) != C =>! collisionwith(charBox(moveRight(C)), charBox(player(engine(C), i)))) => positionX(moveRight(C)) = Engine::width(engine) − HitBox:: Length(charBox(C))

faceRight(moveRight(C)) = faceRight(C)

life(moveRight(C)) = life(C)

positionY(moveRight(C)) = positionY(C)

[switchSide]:

faceRight(switchSide(C)) != faceRight(C)

positionX(switchSide(C)) = positionX(C)

positionY(switchSide(C)) = positionY(C)

life(switchSide(C)) = life(C)

[step]:

step(C, LEFT) = moveLeft(C)

step(C, RIGHT) = moveRight(C)

step(C, NEUTRAL) =null

3.3 Engine

```
Service:  Engine
Types:  bool, int, Commande

Observators:  const height: [Engine] −> int
                                           const width: [Engine]−> int
                                           char: [Engine] x int−> Character
                                                  pre char(E,i) requires i \in {1,2}
                                           player: [Engine] int!Player
                                                  pre player(E,i) requires i \in {1,2}
                                           gameOver: [Engine]−>bool

Constructors:  init: int x int x int x Player x Player−>[Engine]
                                                  pre init(h,w,s,p1,p2) requires h > 0 ^ s > 0 ^ w > s ^ p1 != p2

Operators:  step: [Engine] x Commande x Commande −> [Engine]
                                                  pre step(E) requires : ! gameOver(E)

Observations:
               [invariant]:
                                           gameOver(E) = \exist i  {1,2} Character ::dead(player(E; i))
               [init]:
                                           height(init(h; w; s; p1; p2)) = h
                                           width(init(h; w; s; p1; p2)) = w
                                           player(init(h; w; s; p1; p2); 1) = p1
                                           player(init(h; w; s; p1; p2); 2) = p2
                                           Character ::positionX(char(init(h; w; s; p1; p2); 1)) = (w/2 −s/2)
                                           Character ::positionX(char(init(h; w; s; p1; p2); 2)) = (w/2 + s/2)
                                           Character ::positionY(char(init(h; w; s; p1; p2); 1)) = 0
                                           Character ::positionY(char(init(h; w; s; p1; p2); 2)) = 0
                                           Character ::faceRight(char(init(h; w; s; p1; p2); 1))
                                           Character :: !faceRight(char(init(h; w; s; p1; p2); 2))
               [step]:
                                           char(step(E; C1; C2); 1) = step(char(E; 1); C1)
                                           char(step(E; C1; C2); 2) = step(char(E; 2); C2)
```

3.4 FightChar

```
Service: FightChar refines Character

Observators: isBlocking [FightChar] −> bool
                   isBlockstunned: [FightChar] −> bool
                   isHitstunned: [FightChar] −> bool
                   isTeching: [FightChar] −>bool
                   tech: [FightChar] −> Tech
                      pre tech(C) requires isTeching(C)
                   techFrame: [FightChar] −> bool
                      pre techFrame(C) requires isTeching(C)
                   techHasAlreadyHit: [FightChar] −> bool
                      pre techHasAlreadyHit(C) requires isTeching(C)

                   //on recupere les memes specification du service Character mais on ajoute:
constructors: init: int x int x bool x [Engine] x [HitBox] x [Tech] −> [FigthChar]

Operators: startTech: [FightChar] x Tech −> [FightChar]
                      pre startTech(C,T) requires !isTeching() ˆ !isHitstunned() ˆ !isBlockStunned()

                   step: [FightChar] x Commande −> [FightChar]
                      pre: step(c) requires !isHitStunned ˆ !isBlockStunned ˆ !isTeching
                      // Specification supplementaire pour step
Observation:
           [Invariant]:
                     isBlocking == !isTeching
                     isTeching == !isHitstunned ==!isBlockstunned

           [init]:
                     tech( init(l,s,f,e,h,technique))==technique
                     !isTeching(init(l,s,f,e,h,technique))
                     !techFrame(init(l,s,f,e,h,technique))
                     !techHasAlreadyHit(init(l,s,f,e,h,technique))

           [startTech]:
                     isTeching(starTech(technique))

           [step]:
              // observation supplementaire pour step
              step(c)
                 if (c=BLOCK) => isBlocking
                 if (c=PUNCH) => starTech(punch)
```

3.5 Player

```
Service:  Player
Types:  bool, int
Observators:  getChar: [Player] −> Character
Constructors:  init: −> [Player]
Operators:  setChar: [Player] x Character −> [Player]
Observations:
                      [init]
                      [setChar]
                             getChar(setChar(P,C)) = C
```

Chapitre 4

Tests et Choix d'implémentations

Nous avons choisi d'implémenter directement une hitbox rectangle au lieu de faire une hitbox abstraite qui représente un point du plan et ensuite utiliser un raffinement.

Nous avons eu du mal à faire marcher les tests MBT junit avec notre build XMl . Chose qui à été réussi très tardivement.Cependant, on montre un test de collision entre 2 hitbox et d'appartenance d'un point dans la hitbox . Donc nous basons tous nos tests majoritairement sur les contrats qui sont des tests en ligne.