Spécification

Engine

Service: Engine
Types: bool, int, Commande
Observators:
    const height: [Engine] → int
    const width: [Engine] → int
    char: [Engine] × int → Character
        pre char(E,i) **requires** i 2 f1, 2g
    player: [Engine] × int →Player
        pre player(E,i) **requires** i 2 f1, 2g
    gameOver: [Engine] →bool
Constructors:
    init: int × int × int × Player × Player → [Engine]
        pre init(h,w,s,p1,p2) **requires** h > 0 && s > 0 && w > s && p1 != p2
Operators:
    step: [Engine] × Commande × Commande → [Engine]
        pre step(E) **requires** : gameOver(E)
Observations:
[**invariant**]:
    gameOver(E) (∃i, Character(engine(C), i) = **Character** ::dead(player(E, i))
[**init**]:
    height(init(h, w, s, p1, p2)) = h
    width(init(h, w, s, p1, p2)) = w
    space(init(h, w, s, p1, p2)) = s
    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)

Hitbox

Service: Hitbox
Types: bool, int
Observators:
    PositionX: [Hitbox] → int
    PositionY: [Hitbox] → int
    Hauteur: [Hitbox] → int
    Longueur: [Hitbox] → int
    BelongsTo: [Hitbox] × int × int → bool
    CollidesWith: [Hitbox] × Hitbox → bool

```
      EqualsTo: [Hitbox] × Hitbox → bool
Constructors:
      init: int × int x int x int → [Hitbox]
            pre init(x, y, h, l) requires h > 0 && l > 0
Operators:
      MoveTo: [Hitbox] × int × int → [Hitbox]
            pre moveTo(h, w) requires h > 0 && w > 0
      resize: [Hitbox] × int × int → [Hitbox]
Observations:
[invariant]:
      CollidesWith(H,H1)  =  9  x,y:int  ×  int,  BelongsTo(H,x,y)  &&
      BelongsTo(H1,x,y)
      EqualsTo(H,H1) = 8 x,y:int × int, BelongsTo(H,x,y) = BelongsTo(H1,x,y)
[init]:
      PositionX(init(x, y, h, l)) = x
      PositionY(init(x, y, h, l)) = y
      Hauteur(init(x, y, h, l)) = h
      Largeur(init(x, y, h, l)) = l

[MoveTo]:
      PositionX(MoveTo(H,x,y)) = x
      PositionY(MoveTo(H,x,y)) = y
[Resize]:
      Hauteur (resize (H,h,w)) = h
      Largeur (resize (H,h,w)) = w
```

Combo

```
Service: Combo
Types : int
Observators:
      Combo : [Combo] → int
      frameRestantes : [Combo] → int
Constructors:
      init: int × int → [Combo]
Operators:
      reset: [Combo]  → [Combo]
      addCombo: [Combo]  → [Combo]
            pre addCombo() requires frameRestante() > 0
      step: [Combo]  → Boolean → [Combo]
Observations:
[invariant]:
      Combo() >= 0
[init]:
      Combo(init) = 0
      frameRestantes = 96
[Reset]:
      Combo() =0
[addCombo]:
      combo(addCombo) = Combo + 1
[step]:
[removeFrame]:
      frameRestante(removeFrame()) = frameRestante -1
```

[**comboPossible**]:
	comboPossible == frameRestante > 0

Character

Service: Character
Types: bool, int, Commande, Personnage
Observators:
	positionX: [Character] → int
	positionY: [Character] → int
	hauteur: [Character] → int
	longueur: [Character] → int
	personnage :[Character] → Personnage
	nom :[Character] → String
	state: [Character] → State
	engine: [Character] → Engine
	charBox: [Character] → Hitbox
	jump: [Character] → Jump
	life: [Character] → int
	const speed: [Character] → int
	faceRight: [Character] → bool
	dead: [Character] → bool
	id: [Character] → int
Constructors:
	init: Personnage x int × int × bool × Engine → [Character]
		pre init(p, l,s,f,e) **requires** l > 0 && s > 0

Operators:
	moveLeft: [Character] → [Character]
	moveRight: [Character] → [Character]
	switchSide: [Character] → [Character]
	step: [Character] × Commande → [Character]
		pre step() **requires** :!dead
	bindJump: [Character] x Jump → [Character]
Observations:
[**invariant**]:
	positionX(C) > 0 && positionX(C) < **Engine**:: width(engine)
	positionY(C) > 0 && positionY(C) < **Engine**:: height(engine)
	dead(C) = !(life > 0)
[**init**]:
	life(init(p, l, s, f, e)) = l && speed(init(p, l, s, f, e) = s &&
faceRight(init(p, l, s, f, e)) = f
	&& engine(init(p, l, s, f, e)) = e && personage = p
	∃h :Hitbox, charbox(init(p, l, s, f, e) = h
	∃j :Jump, jump(init(p, l, s, f, e) = j

[**moveX**] : #Tous les mouvements

faceRight(moveLeft(C)) = faceRight(C) ∧ life(moveLeft(C)) = life(C)

[**moveLeft**]:

(∃i, player(engine(C), i) != C ∧ collisionwith(hitbox(moveLeft(C)), hitbox(player(engine(C), i)))) ⇒ positionX(moveLeft(C)) = positionX(C)

positionX(C) ≤ speed(C) ∧ (∀i, player(engine(C), i) != C ⇒ ¬collisionwith(hitbox(moveLeft(C)),

hitbox(player(engine(C), i)))) ⇒ positionX(moveLeft(C)) = positionX(C) – speed(C)
positionX(C) > speed(C) ∧(∀i, player(engine(C), i) != C ⇒ ¬collisionwith(hitbox(moveLeft(C)),

hitbox(player(engine(C), i)))) ⇒ positionX(moveLeft(C)) = 0
positionY(moveLeft(C)) = positionY(C)
HitboxState(moveLeft(C)) = HitboxState::STANDING

[**moveRight**]:
(∃i, player(engine(C), i) != C ∧ collisionwith(hitbox(moveRight(C)), hitbox(player(engine(C), i)))) ⇒ positionX(moveRight(C)) = positionX(C)
positionX(C) <= speed(C) ∧ (∀i, player(engine(C), i) != C ⇒

¬collisionwith(hitbox(moveRight(C)), hitbox(player(engine(C), i)))) ⇒ positionX(moveRight(C)) = positionX(C) + speed(C)

positionX(C) ≤ speed(C) ∧(∀i, player(engine(C), i) != C ⇒ ¬collisionwith(hitbox(moveRight(C)), hitbox(player(engine(C), i)))) ⇒
positionY(moveLeft(C)) = positionY(C)
HitboxState(moveLeft(C)) = HitboxState::STANDING

[moveDown]:
HitboxState(moveLeft(C)) = HitboxState::CROUCHING
[moveDownLeft]
HitboxState(moveLeft(C)) = HitboxState::CROUCHING

[moveDownRight]
HitboxState(moveLeft(C)) = HitboxState::CROUCHING

[moveUpRight]
[moveUpLeft]
[moveUpNeutral]
[moveDownRight]
POST BAISSEMENT HAUTEUR HITBOX
[moveDownLeft]
[**switchSide**]:
faceRight(switchSide(C))! = faceRight(C)
positionX(switchSide(C)) = positionX(C)
[**step**]:
step(C, LEFT) = moveLeft(C)
step(C, RIGHT) = moveRight(C)
step(C, NEUTRAL) = neutral(C)
step(C, UPRIGHT) = moveUpRight(C)
step(C, UPLEFT) = moveUpLeft(C)
step(C, UPNEUTRAL) = moveUpNeutral(C)
step(C, DOWNRIGHT) = moveDownRight(C)

```
      step(C, DOWNLEFT) = moveDownLeft(C)
      step(C, DOWNNEUTRAL) = moveDownNeutral(C)
[bindHitbox]:
      Charbox(bindHitbox(h)) = h
[bindJump]:
      Jump(bindJump(j)) = j
```

FighterCharacter

```
Service: FighterCharacter refines Character
Observators:
      isBlocking: [FightChar] → bool
      isBlockstunned: [FightChar] → bool
      isHitstunned: [FightChar] → bool
      isTeching: [FightChar] → bool
      techniqueCourante: [FightChar] → Tech
            pre tech(C) requires isTeching(C)
      getComboService → Combo
      getCombo → int
      isCombo → bool
      getCombo → int

Operators:
      startTech(): [FightChar] x Technique → void
            pre startTech  requires isTeching()
      endTechnique(): [FightChar] → void
            pre startTech  requires isTeching()
      takeAttack(): [FightChar] x damage x hstun x bstun → void
            pre takeAttack requires damage > 0 && hstun > 0 && bstun > 0 &&
!dead
      stepCombo(): [FightChar] → void

Observations:

[invariant]:
      isTeching() → techniqueCourrante() != null
      isTeching → ¬isBlocking
      isHitStunned → (frameHitStun() > 0)
      isBlocking → ¬(isHitStunned || isBlockStun) = false
      isBlockStunned → (frameBlockStun() > 0)

Observators

[startTech]:
      TechniqueCourante(startTech()) != null && isTeching(startTech(t)) =
true
[stepCombo]:
      (h      &&      Combo::isComboPossible(stepCombo(h)))            =>
(getCombo(stepCombo(h))) = getCombo() + 1
      (h     &&     ¬     Combo::isComboPossible(stepCombo(h)))         =>
(getCombo(stepCombo(h))) = 1
[init]:
      ∃t :Technique, techniques(init(p, l, s, f, e)) = t
```

```
      ∃c :Technique, combo(init(p, l, s, f, e)) = c
[moveX] : #Tous les mouvements
 (isTeaching() || isBlocking() || isHitStunned() || isBlockstunned()) =
false
combo(moveX(C)) = combo(c)
```

Techniques

Service: Technique

Type : Hitbox

Observators:

```
      damage: [Technique] → int
      hitstun: [Technique] → int
      blockstun: [Technique] → int
      startuptime: [Technique] → int
      hittime: [Technique] → int
      recoverytime: [Technique] → int
      box: [Technique] -> Hitbox
      frame: [Technique] → int
```

Operators:

```
      init : int x int x int x int x int x int
      step : [Technique] -> [Technique]
            pre step(T) requires frame < startuptime + hittime + recoverytime
      launchTechnique: [Technique] -> [Technique]
```

Observation:

```
      [init] :
            damage(init(d,hs,bs,s,ht,r,bo)) = d
            hitstun(init(d,hs,bs,s,ht,r,bo)) = hs
            blockstun(init(d,hs,bs,s,ht,r,bo)) = bs
            startuptime(init(d,hs,bs,s,ht,r,bo)) = s
            hittime(init(d,hs,bs,s,ht,r,bo)) = ht
            recoverytime(init(d,hs,bs,s,ht,r,bo)) = r
            box(init(d,hs,bs,s,ht,r,bo)) = bo
      [step]:
            frame(step(T)) = frame(T)+1
      [launchTechnique] :
            frame = 0
```