UPMC
SORBONNE UNIVERSITÉS

04/05/2017

Spécification du jeu Street Fighter

Loïc LAFONTAINE MAXIME LAVASTE

Table des matières

Introduction

Dans le cadre de l'enseignement de l'UE Composants, nous avons dû spécifier, créer des tests puis implémenter un Street Fighter.

Un Street Fighter est un jeu de combat 2D où le but est de tuer son adversaire.

Le but étant de fournir un programme respectant notre spécification grâce à la programmation par contrat ainsi que des tests MBT.

Nous avons utilisé le langage Java ainsi que Java FX pour créer facilement une interface Homme Machine respectant le modèle MVC.

Création des personnages

Nous avons créé un système simple pour créer et configurer un nouveau personnage. Nous nous servons de fichiers de configuration pour chaque personnage.

```
nom=Chun Li
vie=1900
vitesse=13

[STANDING]
standing_height=292
standing_width=185

[CROUCH]
crouching_height=251
crouching_width=185

[JUMP]
start_up=3
move_up=6
on_air=9
move_down=6
landing=5
vitesse_x=5
vitesse_y=35

[Attack - PUNCH]
punch_damage=55
punch_hit_stun=70
punch_block_stun=45
punch_hit_frame=6
punch_start_up_frame=3
punch_recovery_frame=12
punch_debut_y=70
punch_height=54
punch_width=210
punch_debut_x=180

[Attack - SUPERPUNCH]
super_punch_damage=25
super_punch_hit_stun=30
super_punch_block_stun=15
super_punch_hit_frame=6
super_punch_start_up_frame=3
super_punch_recovery_frame=12
super_punch_debut_y=45
super_punch_height=34
super_punch_width=150
super_punch_debut_x=180

[Attack - LIGHT-KICK]
light_kick_damage=20
light_kick_start_up_frame=2
light_kick_hit_frame=4
light_kick_recovery_frame=11
light_kick_hit_stun=30
light_kick_block_stun=15
light_kick_debut_y=50
light_kick_height=44
light_kick_width=115
light_kick_debut_x=180
```

Exemple du fichier de configuration du personnage Chun Li

Ensuite, nous avons juste à rajouter une nouvelle valeur d'enum Personnage pour notre nouveau personnage. La partie plus longue, est la partie où il faut rajouter les spirites du nouveau personnage dans notre dossier de configuration.

Notre gestion de création d'un nouveau personnage est assez simple à réaliser et peu lourde en modification de code existant. L'utilisateur peut modifier les caractéristiques de son personnage facilement, sans toucher au code. Par exemple, il peut rajouter des dommages à ses attaques, devenir invincible en se rajoutant des milliers de points de vie, avoir une petite hitbox pour éviter de se faire toucher…

Enfin, si le joueur veut se rajouter une technique, il suffit d'indiquer dans le fichier texte, puis de rajouter l'initialisation ainsi qu'un bind du joueur vers la technique dans notre fabrique.

Seule limite, nous avons estimé que l'utilisateur utiliserait les fichiers avec sérieux, c'est-à-dire qu'ils ne sont pas régénérés automatiquement en cas de suppressions, et si l'utilisateur entre une valeur supérieure à un Integer en Java, il n'y a aucun mécanisme de sécurité pour garantir le bon fonctionnement du programme.

Manuel de jeu

L'utilisateur a le choix entre deux personnages, Chun Li et Ryu. Chaque personnage peut sauter, se déplacer de gauche et à droite, s'accroupir puis allez à gauche et à droite, trois coups ainsi qu'une garde.

	Joueur 1	*Joueur 2*
Sauter	Z	↑
S'accroupir	S	↓
Allez à droite	D	→
Allez à gauche	Q	←
Donner un coup de poing	X	1
Donner un coup de pied	W	2
Attaque à distance	R	3
Garde	E	0

De plus, si un des joueurs appuie sur F2, les hitboxs des joueurs seront affichés. Nous avons décidé d'afficher les hitboxs des coups par défaut pour voir plus facilement les coups.

Enfin le bouton F3 permet d'avancer le combat frame par frame pour observer les techniques ou faire un débogage frame par frame pour mieux comprendre ce qui se passe.

Dans l'écran de sélection des personnages, le touche A sert à basculer d'un contrôle du joueur à l'intelligence artificiel pour le joueur 2 et vice versa.

Figure 1 Capture d'écran du jeu

Sur l'écran, on peut apercevoir la barre de vie pour les deux joueurs ainsi que le compteur de combo en haut à gauche pour le joueur 1 et à droite pour le joueur 2.

Implémentations

Un joueur possède cinq états différents. Un personnage ne peut pas être dans deux états en même temps. Seule exception : on peut être stun en jump, mais le stun ne s'effectueras que lorsque le jump sera totalement effectué, c'est-à-dire lorsque l'étape de réception du jump sera terminée.

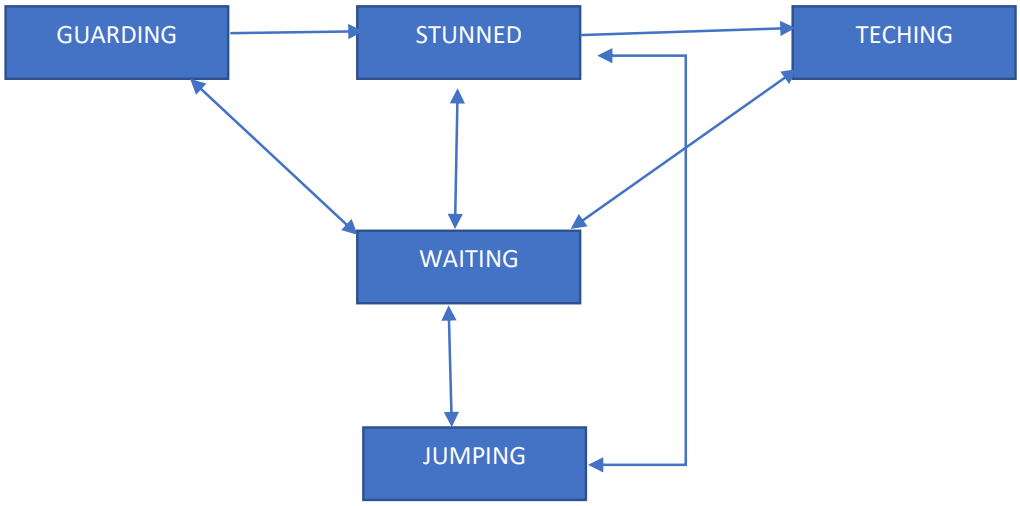

Figure 2 Transition d'un état à un autre

L'état waiting indique que le character attend une nouvelle commande pour effectuer soit un mouvement, soit une demande d'attaque, de garde ou de jump.

L'état Stunned indique le character est Stun (block ou hit Stun).

L'état Teching et Jumping indique que le joueur doit continuer soit son jump, soit sa technique en cours.

L'état guarding indique que le personne se protège.

Enfin, un *FightChar* possède une *hashmap* de technique, on peut ainsi lui bind ainsi une infinité de technique. Lorsqu'on lance une technique, on associe la technique lancée à la *techniqueCourante* accessible via un observator.

Un *FightChar* possède une méthode pour se prendre des dégâts. La méthode *takeDamage*. Cette méthode permet de mettre à jour la vie d'un personnage ainsi que son stun. Elle vérifie que si le joueur est en train de faire une attaque, les dégâts ainsi que la valeur de stun sont doublés.

Dans notre implémentation, une technique est un objet comportant plusieurs valeurs.

```
[ATTACK - LIGHT-KICK]
LIGHT_KICK_DAMAGE=20
LIGHT_KICK_START_UP_FRAME=2
LIGHT_KICK_HIT_FRAME=4
LIGHT_KICK_RECOVERY_FRAME=11
LIGHT_KICK_HIT_STUN=30
LIGHT_KICK_BLOCK_STUN=15
LIGHT_KICK_DEBUT_Y=50
LIGHT_KICK_HEIGHT=44
LIGHT_KICK_WIDTH=115
LIGHT_KICK_DEBUT_X=180
```

Les damages sont le nombre de dégâts que fait une technique. Debut_y, debut_x, height et widht sont des données pour créer l'hitbox d'une technique. De plus, une technique possède une puissance de stun représenté par les hit_stun et hit_block_stun. Enfin, une technique peut être dans quatre états, en start up, en hit en recovery et terminé.

Une technique possède son propre compteur de frame. On connait l'état d'une technique selon la valeur de sa frame.

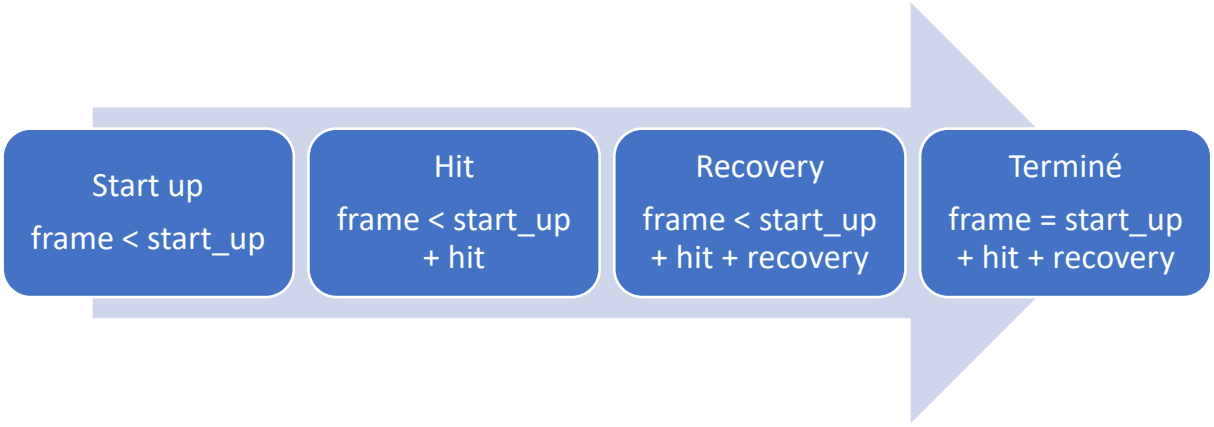

Figure 3 Ordre chronologique d'une technique

Un joueur peut lancer une technique, lorsqu'elle est lancée, on met son nombre de hit à 0, ainsi que son nombre de frame à 0. Le nombre de hit permet de s'assurer qu'une technique touche au maximum une fois l'autre joueur.

Une technique possède une fonction *step* qui permet d'avancer frame par frame l'état de la technique. Lorsqu'on est en état startUp et recovery,ce sont juste des frames d'attentes, on n'effectue aucun calcul, c'est-à-dire que ni l'attaque, ni les personnages sont modifiés.

Lorsque l'on passe en état hit, on met dans un premier temps à jour la position de la hitbox avec la position du personnage. Ensuite, on vérifie si la hitbox de l'attaque entre en collision avec la hitbox de l'autre joueur.

Si elle touche l'hitbox de l'autre joueur, on considère que l'attaque a réussi. On appelle alors la méthode *takeAttack* de other pour modifier l'état de other ainsi que décrémenter sa vie si besoin et gérer si on est dans un cas de contre.

De plus, on appelle la méthode *stepCombo(true)* pour signaler qu'on a effectué un coup réussi et faire évoluer le compteur de combo. Enfin, on incrémente le nombre de coups donnés pour vérifier qu'un coup ne touche qu'une fois dans le contrat.

Lorsque l'état recovery est accompli, on arrive dans l'état terminé, c'est-à-dire que la technique a fini de s'exécuter, et que le joueur ne doit plus rester en état Teching mais revenir en Waiting.

Fonctionnalités

Fonctions non implémentées

Le jump vers la gauche ainsi que vers la droite n'ont pas été implémenté. Par extension, nous ne pouvons pas tester le changement de côté.

Fonctionnalités supplémentaires

Choix de personnage

Nous pouvons créer un nombre infini de personnage avec notre méthode. Notre seule limite et difficulté sont les sprites. Nous avons décidé de ne pas en rajouter plus, car cela demande de nombreuses heures pour modifier des sprites pour qu'il s'adapte à notre interface graphique mais de laisser la liberté au joueur de modifier ses propres fichiers de configuration pour pouvoir créer son propre personnage.

Implémentation d'une IA

Nous avons implémenté une IA basique qui prends au hasard une de nos commandes. Son comportement est semi aléatoire, nous favorisons ses déplacements ainsi que de donner des coups.

Couter

Nous avons implémenté et spécifié la capacité de contrer une attaque, c'est-à-dire de donner un bonus au joueur qui réussit à toucher l'autre joueur lorsqu'il est lui-même en train d'attaquer.

Conclusion

Le principal objectif d'implémenter, de spécifier ainsi que tester notre projet a été atteint.

Cette manière de programmer a été pour nous complétement nouvelle puisque d'habitude, on programmait très souvent avant de spécifier.

L'implémentation des tests ainsi que des contrats nous a permis de trouver l'origine des bugs plus rapidement et efficacement. Cela nous a permis d'observer l'utilité et le besoin d'une spécification ainsi qu'une série de test efficace lors d'un réel projet.

Spécification

Nous avons décidé de supprimer le service Hitbox et de créer directement un service HitboxRectangle,

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
    && Player::Character(0).isFaceRight() != Player::Character(1).isFaceRight()
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))
    Character(1).isFaceRight() != character(2).isFaceRight()
    Character::PositionX(1) < Character::PositionX(2) ⇒ Character(1).isFaceRight() = true
    Character::PositionX(1) > Character::PositionX(2) ⇒ Character(1).isFaceRight() = false

[**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)) = h - Character ::largeur(1)
    Character ::positionY(char(init(h, w, s, p1, p2), 2)) = h - Character ::largeur(2)
    Character ::faceRight(char(init(h, w, s, p1, p2), 1)) = true
    Character :::faceRight(char(init(h, w, s, p1, p2), 2)) = false
[**step**]:
    char(step(E, C1, C2), 1) = step(char(E, 1), C1)
    char(step(E, C1, C2), 2) = step(char(E, 2), C2)

PlayerService

Service: PlayerService
Types: Character
Constructors:
init: Character → [Player]
pre init(c) requires c != null
Observators:
Character: [PlayerService] → Character

IAService

Service: IAService **refines** PlayerService
Types: Character
Operators:
cmd: [PlayerService] → Commande

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**]:
PositionX > 0 && PositionY > 0 && largeur > 0 && hauteur > 0
[**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]:
      If hit = true ->   combo() + 1 == combo(step(hit))
      Else (combo() ==  combo(step(hit)) || combo(step(hit)) = 0)
[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
      jumping: [Character] → boolean
      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]
moveUp: [Character] → [Character]
pre moveUp() **requires** !isJumping
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(moveDown (C)) = HitboxState::CROUCHING

```
[moveDownRight]
      HitboxState(moveDownRight (C)) = HitboxState::CROUCHING
      #Même postCondition que [moveLeft] sauf pour HitboxState
      #On change juste la valeur de speed(c) que l'on divise par 2


[moveDownRight]
      HitboxState(moveDownRight (C)) = HitboxState::CROUCHING
      #Même postCondition que [moveRight] sauf pour HitboxState
      #On change juste la valeur de speed(c) que l'on divise par 2

[moveUp]
[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)
      hitStun: [FightChar] → int
      blockStun: [FightChar] → int
      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
      moveX():[FightChar] → void
            pre moveX requires (isTeaching() || isBlocking() || isHitStunned()
|| isBlockstunned())     = false
```

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)
[step]
      If state_actuel = WAITING
            step(C, GUARDING) = guarding(C)
            step(C, PUNCH) = launchAttack(PUNCH)
            step(C, LIGHTKICK) = launchAttack (LIGHTKICK)
            step(C, SUPERPUCNH) = launchAttack (SUPERPUCNH)
      else
            isHitFrame() || isBlockStun() ⇒ character() = character(step(C))
      #sauf pour les combos qui sont modifiés (décrémentation de la frame)
            isTeaching() ⇒ techniqueCourante().step()
            isJumping ⇒ jump().step()

      Combo::frameRestante(step(C)) = Combo::frameRestante() -1 ||
      Combo::frameRestante(step(C)) = 95   #Restart temps combo


[takeAttack] :
      isBlocking(takeAttack(d, hs, bs)) = false
      isBlocking(C)  ⇒  isBlockstunned(takeAttack(d, hs, bs)))
      !isBlocking(C) ⇒  isHitstunned(takeAttack(d, hs, bs)))
      isBlocking(C) ⇒ (life(C) = life(takeAttack(d, hs, bs)) &&
      bs == getBlockStun (takeAttack(d, hs, bs))
```

```
      isTeching() && !isBloking(C) ⇒  (life(C) = life(takeAttack(d, hs, bs))
      + damage*2) && hs * 2 == getHitStun (takeAttack(d, hs, bs))

      !isTeching()!isBloking(C) ⇒  (life(C) = life(takeAttack(d, hs, bs)) +
damage) &&
      hs == getHitStun (takeAttack(d, hs, bs))



      Combo(takeAttack (d, hs, bs)) = 0
[endTechnique]:
      isTeching() = false
      stateActuel = Waiting
```

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
      alreadyTouch : [Technique] → int
      nbInt: [Technique] → int
```

Operators:

```
      init : int x int x int x int x int x int
      step : [Technique] x FightChar x FightChar → [Technique]
            pre step(T) requires frame <= startuptime + hittime + recoverytime
      launchTechnique: [Technique] → [Technique]
      endTechnique: [Technique] → [Technique]
            pre endTechnique requires isTeching()
```

Observation:

```
      [invariant] :
            Frame <= startUpTime + HitTime + RecoveryTime
            nbInt <= 1
      [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(me, other)) = frame(T)+1
```

```
            (isRecovery(step(me,other))   ||   isStartUp(step(me,other)))   &&
other::getLife(step(me, other)) = other::getLife()
            IsHitTime() && box.collidesWith(other.charBox()) ⇒
      alreadyTouch (step(me, other)) =  true
            IsHitTime() && box.collidesWith(other.charBox()) && !alreadyTouch
⇒ other.takeDamage(damage, hs, bs)

      [launchTechnique] :
            frame = 0
            isTeching() = true
```

JumpService

Service: Jump

Type : Hitbox

Observators:

```
      startuptime: [Jump] → int
      moveUp: [Jump] → int
      onAir: [Jump] → int
      moveDown: [Jump] → int
      landing: [Jump] → int
      frame: [Jump] → int
      vitesseX: [Jump] → int
      vitesseY: [Jump] → int
      Me: [Jump] →  [FightChar]
```

Operators:

```
      init : int x int x int x int x int x int x int x FightChar
            pre init requires sut > 0 && hs > 0 && moveDown >0 … && me !=null
&& moveDown == moveUp
      step : [Jump] x FightChar x FightChar → [Jump]
            pre step(T) requires frame <= startuptime + hittime + recoverytime
      launchJump: [Jump] → [Jump]
      endJump: [Jump] → [Jump]
            pre endJump requires isTeching()
Observation:
      [invariant] :
            Frame <= startUpTime + moveUp + onAir + moveDown + landing
      [init] :
            startuptime(init(sut, moveUp, onAir, moveDown,landing,vitesseX, vitesseY)) = sut
            moveUp(init(sut, moveUp, onAir, moveDown,landing,vitesseX, vitesseY))= hs
            onAir(init(sut, moveUp, onAir, moveDown,landing,vitesseX, vitesseY))= onAir
            moveDown (init(sut, moveUp, onAir, moveDown,landing,vitesseX, vitesseY)) = moveDown
            landing (init(sut, moveUp, onAir, moveDown,landing,vitesseX, vitesseY))= landing
            vitesseY (init(sut, moveUp, onAir, moveDown,landing,vitesseX, vitesseY))= vitesseY
            vitesseX  (init(sut,  moveUp,  onAir,  moveDown,landing,vitesseX,  vitesseY))=  vitesseX

      [step]:
            frame(step(other)) = frame(T)+1
      [launch]:
            frame(launch) = 0
```

Test MBT

Objectif 1 : Précondition de init Engine

- **Cas de test 1.1** : cas positif Engine ::testInit

Conditions initiales :
Joueur J1 et J2 initialisés
Opérations :
E1.1 = init(720,1080,500,J1,J2)
Oracle :
Pas d'exception levée

- **Cas de test 1.2** : cas négatif Engine ::testInit_Negatif

Conditions initiales :
Joueur J1 et J2 initialisés
Opérations :
E1.2 = init(0,1080,500,J1,J2)
Oracle :
Exception levée

Objectif 2 : GameOver

- **Cas de test 2.1** : gameOverJ1: Engine ::testGameOverJ1

Conditions initiales :
Character::life(char(E1.1,0))>0
Character::life(char(E1.1,1))>0
Opérations :
takeAttack(char(E1.1,0),maxLife(char(E1.1,0),100,100)
Oracle :
gameOver(E1.1)=true

- **Cas de test 2.2** : gameOverJ2 Engine ::testGameOverJ2

Conditions initiales :
Character::life(char(E1.1,0))>0
Character::life(char(E1.1,1))>0
Opérations :
takeAttack(char(E1.1,1),maxLife(char(E1.1,1),100,100)
Oracle :
gameOver(E1.1)=true

Objectif 3 : Incrémentation Combo

- **Cas de test 3.1** : si touché : Combo:: testComboSiTouche

Conditions initiales :

Si on attaque, on touche l'adversaire :

```
H1 = Character::charBox(Engine::char(E1.1,0))
H2_X = Hitbox::positionX(Character::charBox(Engine::char(E1.1,1)))
H2_Y = Hitbox::positionY(Character::charBox(Engine::char(E1.1,1)))
HitboxTech = Technique::box(FighterCharacter::techniqueCourante(Engine::char(E1.1,1)))
HT_X = Hitbox::positionX(HitboxTech)
HT_Y = Hitbox::positionY(HitboxTech)

Hitbox::belongsTo(H1,H2_X+HT_X,H2_Y+HT_Y)
```

Opérations :

```
Faire X fois
{
    E3.1 = Engine::step(E1.1,Commande::PUNCH,Commande::NEUTRAL)
    Tant que FighterCharacter::teching(Engine::char(0))
        E3.1 = Engine::step(E3.1,Commande::PUNCH,Commande::NEUTRAL)
}
```

Oracle :

```
FightChar::combo(Engine::char(0))==X
```

- **Cas de test 3.2** : si non touché : Combo:: testComboSiNonTouche

Conditions initiales :

Aucune

Opérations :

```
Faire X fois
{
    E3.2 = Engine::step(E1.1,Commande::PUNCH,Commande::NEUTRAL)
    Tant que FighterCharacter::teching(Engine::char(0))
        E3.2 = Engine::step(E3.2,Commande::PUNCH,Commande::NEUTRAL)
}
```

Oracle :

```
FighterCharacter::combo(Engine::char(0))==0
```

Objectif 4 : Reset Frame Combo

- **Cas de test 4.1** : Reset Frame Combo : Combo::testResetFrame

Conditions initiales :

Combo::frameRestante(FighterCharacter::getComboService(Engine::char(E3.1,0))>0

Opérations :

K = Combo::frameRestante(FighterCharacter::getComboService(Engine::char(E3.1,0))

E4.1 = E3.1

Faire K fois
{
E4.1 = Engine::step(E4.1,Commande::NEUTRAL,Commande::NEUTRAL)
}

Oracle :

Combo::frameRestante(FighterCharacter::getComboService(Engine::char(E4.1,0)=0

Objectif 5 : Reset valeur Combo

- **Cas de test 5.1** : Reset valeur Combo : Combo::testResetValCombo

Conditions initiales :

Combo::frameRestante(FighterCharacter::getComboService(Engine::char(E3.1,0))>0

Opérations :

K = Combo::frameRestante(FighterCharacter::getComboService(Engine::char(E3.1,0))

5.1 = E3.1

Faire K fois
{
E5.1 = Engine::step(E5.1,Commande::NEUTRAL,Commande::NEUTRAL)
}

Oracle :

FighterCharacter::getCombo(Engine::char(E5.1,0)=0

Objectif 6 : Move Left limite du terrain

- **Cas de test 6.1** : moveLeft Border Left : Character::moveLeftBorder

Conditions initiales :

Character::PositionX(Engine::char(E1.1,0))=1

Opérations :

C6.1 = Character::MoveLeft(Engine::char(E1.1,0))

Oracle :
Character::PositionX(C6.1)=1

Objectif 7 : Move Right limite du terrain

- **Cas de test 7.1** : moveLeft Border Right : Character::moveRightBorder

Conditions initiales :

Character::PositionX(Engine::char(E1.1,0))=Engine::width(E1.1)-1

Opérations :
C7.1 = Character::MoveRight(Engine::char(E1.1,0))

Oracle :
Character::PositionX(C7.1)=Engine::width(E1.1)-1

Objectif 8 : Diminution de la vie si takeAttack

- **Cas de test 8.1** : diminution vie: FightCharacter::testTakeAttackVie

Conditions initiales :
Aucune

Opérations :
C8.0 = Engine::char(E1.1,0)
C8.1 = FightCharacter::takeAttack(Engine::char(E1.1,0),10,10,10)

Oracle :
Character::life(C8.1)= Character::life(C8.0)-10

Objectif 9 : Mise en Stunned si takeAttack

- **Cas de test 9.1** : stunned : FightCharacter::testTakeAttackStunned

Conditions initiales :
Aucune

Opérations :
C9.0 = Engine::char(E1.1,0)
C9.1 = FightCharacter::takeAttack(Engine::char(E1.1,0),10,10,10)

Oracle :
FightCharacter::hitStunned(C9.0)=true