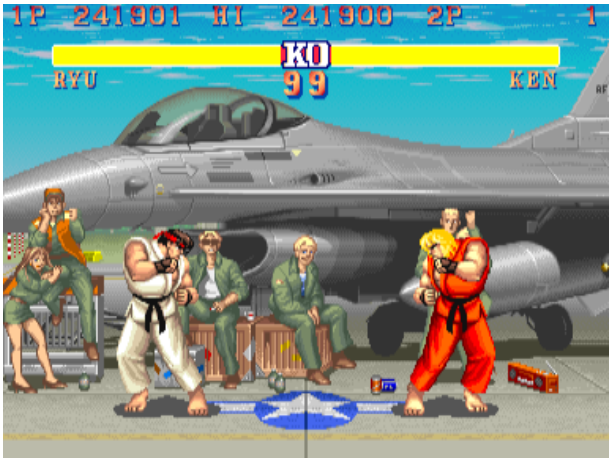
Examen Réparti 1 - PAF 2022: *Street Fighter*

PAF 2021 - Master STL S2

17 Mars 2021



Préliminaires

Aucun barême n'est donné, les copies seront évaluées sur les points suivants:

- qualité de la programmation fonctionnelle,
- maitrises des spécificités de Haskell vues en cours et en TD,
- pertinence de la modélisation : respect des principes vus au TD3, écriture systématique de propriétés (invariants, post-conditions), preuves et tests.

Le langage du partiel est le *Haskell* utilisé en cours et en TD. Les petites erreurs de syntaxe ne sont pas pénalisées.

Description du jeu

Le but de cet examen est de spécifier un jeu vidéo de bagarre (1 contre 1), dans la lignée de *Street Fighter*. Les joueurs contrôlent chacun un personnage, en 2D vu de côté, les personnages se font face et peuvent se déplacer vers la gauche ou la droite (avancer ou reculer vers leur adversaire), ainsi que sauter et se baisser. Les personnages peuvent se donner des coups, commandés par les actions des joueurs et se protéger des coups de l'adversaire. Un coup reçu par un personnage qui ne se protège pas fait baisser sa vie (une valeur numérique). Un personnage dont la vie arrive à 0 perd le duel.

Partie 1 : Coordonnées

Le jeu est représenté en deux dimensions et *vu de côté*. Les joueurs évoluent dans une *zone* `Z h l` dont les coordonnées vont de $(-l, 0)$ (en bas à gauche) à (l, h) (en haut à droite).

La position de différents éléments du jeu peut être décrite à l'aide de coordonnées `C x y` représentant le point (x, y).

Les mouvements `M d n` sont donnés par une direction `d` (haut, bas, gauche, droite - dans un premier temps) et une distance `n`.

```
data Zone = Zone Integer Integer
data Coord = Coord Integer Integer
data Direction = H | B | G | D
data Mouvement = Mouv Direction Integer
```

Question 1.1 Ecrire une fonction `bougeCoord :: Coord -> Mouvement -> Coord` qui bouge un point du plan selon un mouvement.

Question 1.2 Ecrire une fonction `bougeCoordSafe :: Coord -> Mouvement -> Zone -> Maybe Coord` qui bouge un point du plan selon un mouvement au sein d'une zone, en s'assurant que les coordonnées d'arrivée du point sont à l'intérieur de la zone.

Question 1.3 Ecrire la propriété `prop_gaucheDroite_bougeCoord :: Coord -> Integer -> Bool` qui stipule que deux utilisation successives de `bougeCoord` correspondant à un mouvement de distance d vers la gauche suivi d'un mouvement de distance d vers la droite correspond à l'identité.

Question 1.4 Prouver `prop_gaucheDroite_bougeCoord c d` pour tout `c` et `d`.

Question 1.5 Peut-on faire de même avec une propriété `prop_gaucheDroite_bougeCoordSafe c d` ? Justifier.

Question 1.6 Donner un jeu de trois tests *Hspec* pour les types et les fonctions décrits dans cette partie.

Partie 2 : Hitbox

Dans la plupart des jeux vidéo de bagarre, les combattants sont traités, de manière abstraite, sous forme de boîtes (*hitbox*), des ensembles de points du plan, souvent obtenus par compositions de rectangles de pixel (cf. Figure 1).

La forme de la hitbox d'un combattant peut varier au cours d'une partie (par exemple quand le combattant se baisse ou qu'il donne un coup). Quand un combattant donne un coup, il faut pouvoir tester si la hitbox du membre (pied ou poing) du personnage qui donne le coup rentre en contact avec la hitbox du corps du combattant adverse.

Dans un premier temps, on se restreint aux hitbox obtenues par composition de rectangles du plan.

```
data Hitbox =   Rect Coord Integer Integer
              | Composite (Seq Hitbox)
```

Question 2.1 On veut qu'une hitbox ne soit jamais vide. Donner un invariant (une propriété de type `Hitbox -> Bool`) pour le type `Hitbox` qui vérifie cette propriété.

Question 2.2 Ecrire un constructeur intelligent qui prend en entrée une série de couples de coordonnées $(c_i^1, c_i^2)_{i \leq n}$ et qui produit une hitbox correspondant à la réunion des rectangles situés entre les points (c_i^1, c_i^2).

Question 2.3 Ecrire une fonction `appartient :: Coord -> Hitbox -> Bool` qui décide si un point donné par ses coordonnées se situe dans une hitbox.

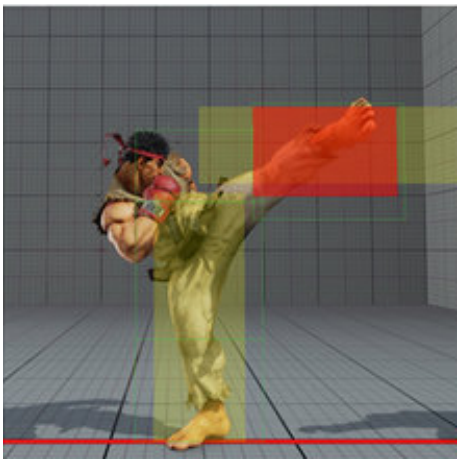


Figure 1: Exemple de hitbox : on distingue la hitbox du personnage (jaune) et celle de la technique utilisée (rouge).

Question 2.4 Ecrire une fonction `bougeHitbox :: Hitbox -> Mouvement -> Hitbox` qui déplace une Hitbox dans le plan.

Question 2.5 Proposer une propriété de post-condition pour la fonction `bougeHitbox` utilisant la fonction `appartient`.

Question 2.6 Décrire la fonction `bougeHitboxSafe :: Hitbox -> Mouvement -> Zone -> Maybe Hitbox` qui déplace une hitbox dans une zone en s'assurant qu'apres le déplacement, tous les points de la hitbox sont inclus dans la zone.

Question 2.7 Ecrire la fonction `collision :: Hitbox -> Hitbox -> Bool` qui décide si deux hitbox entrent en collision (c'est-à-dire, s'il existe au moins un point qui appartient aux deux hitbox).

Partie 3 : Jeu et Combattants

Une zone de jeu contient deux combattants. Le jeu se déroule par tour (appelé *frame*) pendant laquelle les combattants peuvent effectuer une (ou une partie d'une) action, comme se déplacer ou se donner des coups.

Un combattant est représenté par (entre autres) des coordonnées de base, une hitbox, une direction (gauche ou droite) vers laquelle il fait face, une santé (nombre de points de vie restants). Lorsqu'au moins un des deux combattants a un total de point de vie négatif ou nul, le jeu passe dans un état *terminé*.

Pendant le jeu, un combattant peut se déplacer sur l'axe x mais il ne peut pas "passer à travers" l'autre combattant, ni sortir de la zone de jeu. Les deux combattants doivent toujours se faire face (dans la suite, ils pourront changer de direction, en sautant l'un au dessus de l'autre par exemple).

Voici une première version des types pour le joueur et le jeu du module `Jeu`.

```
data EtatCombattant =   Ko
                      | Ok Integer      -- sa sante actuelle

data Combattant = Comb {
          positionc :: Coord,
          hitboxc :: Hitbox,
          facec :: Direction,
          etatx :: EtatCombattant
```

```
          }

data Jeu =    GameOver Integer -- le numero du joueur vainqueur
            | EnCours {
                  joueur1 :: Combattant,
                  joueur2 :: Combattant,
                  zoneJeu :: Zone
                  }
```

Question 3.1 Décrire des invariants pour les types ci-dessus (au besoin, modifier ces types) permettant de garantir des propriétés cohérentes, comme par exemple :

- les positions des deux combattants sont inclues dans la zone de jeu,
- les hitbox des deux combattants ne se chevauchent pas,
- les deux combattants se font face.

Question 3.2 On utilise une fonction `tourJeu :: Jeu -> Jeu` (son code ne nous intéresse pas pour le moment) qui fait avancer le jeu d'un tour (d'une *frame*) en appelant des fonctions `tourJoueur :: Joueur -> Joueur` pour chacun des joueurs (plus tard, on modifiera le type des fonctions pour prendre en compte les *entrées* clavier des utilisateurs du jeu).

Ecrire une post-condition pour cette fonction qui garantit que si un au moins un des joueurs a un total de points de vie négatif ou nul, le jeu est terminé au tour suivant.

Question 3.3 Donner une fonction `bougeJoueur` qui bouge un joueur au sein d'une zone de jeu selon un mouvement passé en paramètre, et qui garantit que les invariants de `Jeu` et `Joueur` sont respectés.

Question 3.4 Ecrire une propriété de post-condition pour la fonction `bougeJoueur`.

Partie 4 : Technique

Pour se battre les combattants peuvent utiliser des techniques et se protèger.

Un combattant qui utilise une technique ou se protège ne peux pas se déplacer.

Une technique représente une attaque (coup de poing, ou pied, ou autres) qui immobilise le combattant pendant un certain temps et tente de toucher l'adversaire.

Lors de l'utilisation d'une technique :

- le combattant est d'abord immobilisé pendant un certain nombre de frames (qui dépend de la technique utilisée), appelées *startup frames*.
- ensuite, une hitbox de coup (dont la forme est la position dépendent de la technique utilisée et de la position du combattant) apparaît et reste active pendant un certain nombre de frames, appelées *hit frames*.
- enfin, la hitbox de coup disparait mais le joueur reste immobilisé (et donc potentiellement vulnérable) pendant un certain nombre de frames, appelées *recovery frames*.
- si la hitbox de l'adversaire rentre en collision avec la hitbox de coup pendant les *hit frames*, deux choses peuvent se produire :
  - soit l'adversaire était en train de se protéger : il ne perd pas de vie, mais reste immobilisé, protégé pendant un certain nombre de frames, appelées *block stun*.
  - soit l'adversaire ne se protégait pas : il perd un certain nombre de points de vie, et reste immobilisé, vulnérable, pendant un certain nombre de frames, appelées *hit stun*.

Question 4 Donner code d'un module `Technique` permettant aux combattant d'utiliser des techniques et indiquer les modifications apportées au module `Jeu` avec types, invariants, opérations, propriétés et tests.