Création et Affrontement de personnages articulés personnalisables (Partie II)

(Génération de HitBox et Algorithme de reconnaissance des collisions)

Projet Tuteuré réalisé par Kévin TRANCHO
encadré par Cyril NICAUD

Année 2016-2017

Table des matières

Chapitre 1

Introduction au problème et rappels

1.1 Combat de deux personnages personnalisés

Le contexte est le suivant :
L'utilisateur a créé son personnage entièrement personnalisé à l'aide des éditeurs proposés :

- Editeur d'**armature** : définition des articulations du personnage avec contraintes de parenté (exemple : les mouvements de l'avant-bras dépendent de ceux du bras)
- Editeur de **formes** : ajout d'une forme visible sur les articulations, fabriquées par des cercles reliés par des polygones
- Editeur de **mouvements** : définition des rotations des articulations à un temps donné et vecteur de déplacement d'un personnage

Une fois ce personnage créé, l'objectif était de le faire combattre contre un autre personnage peu importe leur structure, forme et mouvements.

1.2 Structure des personnages

1.2.1 Personnage

Un **personnage** est défini brièvement par :

- Son **armature** : un arbre binaire fils gauche - frère droit d'os
- Ses **actions** : l'ensemble de ses actions (chaque action est une table qui associe un os à la liste chaînée des rotations qu'il doit effectuer au cours du temps)
- L'**action en cours** à l'instant présent
- Un **vecteur de déplacement**
- Rectangle de **dimensions** du personnage

1.2.2 Armature

Chaque **os** de l'armature est défini brièvement par :
— Ses **extrémités** : coordonnées du début et de la fin de l'os (relatif à son parent)
— Son **angle** : la rotation actuelle
— Ses **successeurs** (arbre binaire fils gauche - frère droit)
— L'ensemble des **formes** qui lui sont associées (chaque forme de l'ensemble est un arbre binaire fils gauche - frère droit de cercles)
— L'**image calculée** de son ensemble de formes
— Son ensemble de **hitbox** (ensemble des éléments collisables représentatifs de l'os et sa forme)
— **Délai** avant nouvelle **collision**

1.3 Précisions sur les composants des personnages et os

1.3.1 Rectangle de dimensions d'un personnage

Le **rectangle de dimensions** d'un personnage est le **plus grand** rectangle contenant toutes les **extrémités des os** et toutes les **hitbox**.
En particulier la partie basse de ce rectangle permet de définir la distance au sol du personnage.

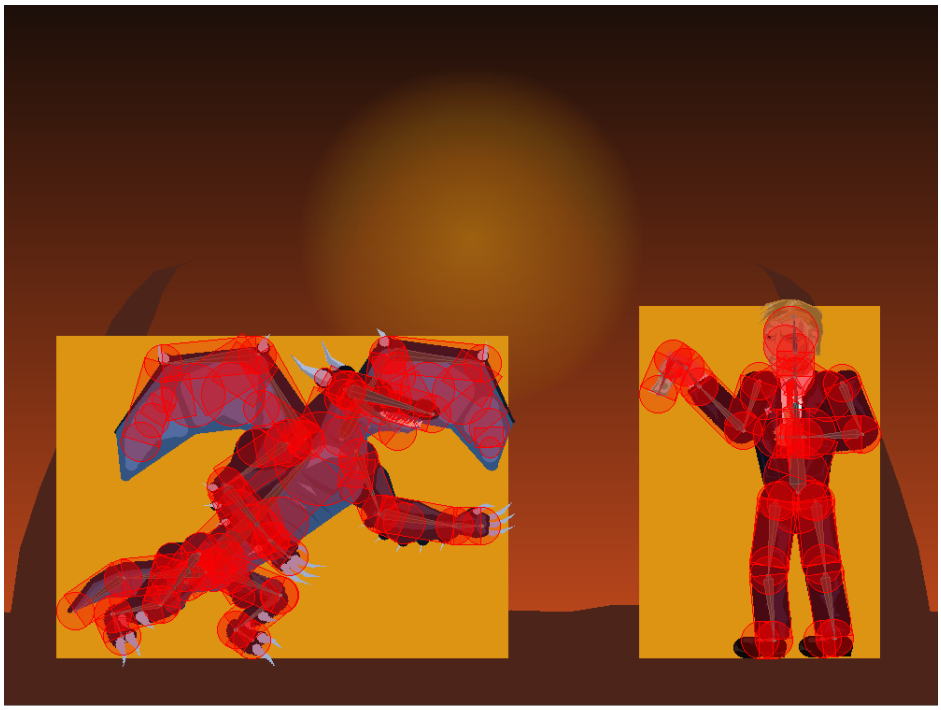


FIGURE 1.1 – *Rectangles de dimensions de deux personnages*

1.3.2 Types de mouvements

Mouvement de vecteur nul

Aucun vecteur à valeur de déplacement.
Il sera utilisé pour des **attaques/actions** où le personnage reste sur place. Exemple : un coup de poing

Mouvement de vecteur linéaire

Vecteur **déplaçant** le personnage dans une direction tant que l'action associée est active.
Ce type de mouvement est uniquement utile pour déplacer le personnage.

Mouvement de vecteur dynamique

Vecteur de déplacement variable avec cas d'arrêt (temps ou collision).
On distingue deux mouvements de ce type :

- Mouvement vers le haut : **saut** ou **bond**, ramené vers le sol par la gravité, cas d'arrêt : collision avec le sol
- Mouvement vers le bas : **glissade**, vecteur de grande amplitude au départ qui converge vers le vecteur nul sur une durée prédéfinie

On peut **combiner** ce type de mouvement avec un **déplacement** en lui donnant une direction, soit prédéfinie, soit par combinaison avec un mouvement de vecteur linéaire.

1.4 Génération et affichage des personnages

Pour chaque os du personnage on associe un ensemble de formes.
Si l'édition des formes n'est pas en cours, on choisit de calculer une image qui correspondra au même affichage que le parcours de l'ensemble des formes.
Le calcul de cette image est couteux (allocation mémoire de la zone de l'image et parcours de l'ensemble des formes), on l'effectue alors une fois jusqu'à nouvelle édition des formes.
Un problème d'optimisation de l'affichage était le suivant :

> à chaque affichage du personnage il faut afficher une forme dont l'angle de rotation est le même que l'angle absolu de l'articulation à laquelle elle est associée.

La solution choisie est celle de calculer la rotation de l'image de base pour un angle α, mais au lieu de l'allouer et la libérer pour un affichage unique, on la garde en mémoire pour possible réutilisation.
A ϵ fixé (ici est choisi $\epsilon = \frac{1}{360} \times \frac{\pi}{180}$ en radians)
On affiche à nouveau cette image sans en calculer de nouvelle si elle doit être affichée pour un angle β tel que

$$|\alpha - \beta| < \epsilon$$

Si $|\alpha - \beta| \geqslant \epsilon$, on calcule l'image pour une rotation β, on l'affiche et on la garde en mémoire pour réitérer le procédé.

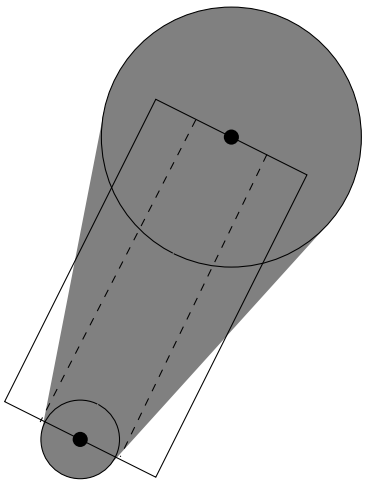
Ceci permet de ne calculer les rotations d'images que lorsque nécessaire (en mouvement et différence possiblement visible à l'oeil)

Chapitre 2

Génération des HitBox

2.1 Problème des HitBox

On aimerait ici une zone de collision que nous nommerons 'HitBox' telle que vérifier son intersection avec d'autres HitBox soit une opération rapide et que cette zone soit représentative du personnage.
Le problème ici est que la forme peut être quelconque, des figures simples telles que des rectangles et des cercles permettraient un calcul rapide.
Le choix a alors été de mélanger les deux figures et les utiliser pour approximer la forme.
Les calculs des intersections des différentes figures sont proposés en annexe.
Pour approximer une forme définie par deux cercles et le trapèze obtenu par le calcul de leurs tangentes externes, le choix s'est porté sur une approximation par un rectangle dont les centres des cercles sont les milieux de deux côtés opposés et dont la longueur est la moyenne des diamètres des deux cercles.



Sur la figure ci-dessus, on a représenté en gris la forme et en noir la hitbox approximant cette situation selon les propriétés données au dessus.

2.2 Algorithme de génération des HitBox

L'Algorithme choisi donne une hitbox pour chaque articulation d'un personnage.
Une hitbox est un arbre frère droit - fils gauche.
On définit une constante BONE_HITBOX (ici 20 pixels) le rayon de la hitbox représentative de l'articulation.
On définit une constante LIMITE_ABSORPTION (ici 8 pixels) la distance à laquelle deux cercle seront considérés comme presque équivalents.

Input : Articulation : os
Output : Hitbox : hitbox
hitbox = {};
hitbox.ajouter(Cercle(origine, BONE_HITBOX));
hitbox.ajouter(Cercle(os.extremité_fin - os.extremité_début, BONE_HITBOX));
for *forme* ∈ *os.formes* **do**
  hitbox.generer_forme(forme, forme.centre);
end
hitbox.simplifier();

Algorithme 1 : Génération de hitbox sur articulation

Function *generer_formes(hitbox, forme, coordonnée)*
  Soit plus_proche le cercle de hitbox le plus proche du point (forme.centre + coordonnée);
  actuel = Cercle(forme.centre + coordonnée, forme.rayon);
  if *inclusion_de_cercles(plus_proche, actuel)* **then**
    On conserve dans hitbox le cercle de plus grand rayon entre plus_proche et actuel;
  end
  else if ∃ *cercle* ∈ *hitbox tel que inclusion_de_cercles(cercle, actuel)* **then**
    On conserve dans hitbox le cercle de plus grand rayon entre cercle et actuel;
  end
  else
    hitbox.ajouter(actuel);
  end
  for *hérité* ∈ $\{forme.fils, forme.frere\}$ **do**
    hitbox.generer_formes(hérité, coordonnée);
  end

Function *simplifier(hitbox)*

```
for (x, y) ∈ hitbox × hitbox do
    if x.rayon < LIMITE_ABSORPTION then
        hitbox.supprimer(x);
        continuer;
    end
    x_cercle = Cercle(x.centre, x.rayon +
      −LIMITE_ABSORPTION   si   x.rayon < y.rayon
       LIMITE_ABSORPTION        sinon );
    y_cercle = Cercle(y.centre, y.rayon +
       LIMITE_ABSORPTION   si   x.rayon < y.rayon
      −LIMITE_ABSORPTION        sinon );
    if inclusion_de_cercles(x_cercle, y_cercle) then
        hitbox.supprimer(x);
        continuer;
    end
    for z fils de y do
        if inclusion_cercle_dans_rectangle(x, Rectangle(y, z, (y.rayon+z.rayon)/2 +
         LIMITE_ABSORPTION)) then
            hitbox.supprimer(x);
            continuer;
        end
    end
end
```

La fabrication des hitbox se fait alors en trois étapes :

— La création d'une hitbox de base englobant l'articulation définie par l'éditeur d'armature (assure l'existence d'une hitbox minimale basée sur l'articulation)

— La génération à partir de la forme en évitant la conservation des cercles dont il existe un cercle plus grand contenant celui-ci (évite les cas répétitions)

— La simplification de la hitbox en éliminant les cercles trop proches à LIMITE_ABSORPTION près (borne le nombre de cercles de la hitbox, évite de conserver des cercles trop proches)

Chapitre 3

Algorithme de reconnaissance des collisions

3.1 Problème des collisions

Notre personnage est maintenant muni de hitbox.
On veut maintenant déterminer s'il y a collision entre deux personnage et si oui, on souhaite obtenir l'articulation pouvant effectuer le plus fort impact sur le personnage adverse.

3.2 Solution

On a pour le personnage les propriétés suivantes :
- Le personnage connait son rectangle de dimensions (plus grande zone contenant toutes les hitbox)
- Pour chaque articulation on possède en mémoire une image de la dernière forme affichée avec le bon angle de rotation (autrement dit, on peut voir cette image comme une zone contenant la hitbox de l'articulation)
- La hitbox d'une articulation contient un nombre borné de cercles
- Une articulation possède un délai avant nouvelle collision, si ce temps n'est pas écoulé on ne vérifie par les collisions avec cette articulation

Ici, on va chercher par vérifier la collision des grands rectangles (dimensions puis image de l'articulation) pour aller évaluer les intersections à un niveau plus atomique (chaque cercle et rectangle des hitbox).
Dans le but de sélectionner l'articulation la plus 'puissante', il a été choisi de parcourir les articulation par ordre décroissant de vitesse (norme du vecteur de translation de la fin de l'extrémité du temps $t-1$ au temps t).
Lorsqu'on tombe sur une articulation de vitesse nulle, on arrête le parcours, car les suivantes seront aussi de vitesse nulle.

3.3 Algorithme

Le déroulement de la gestion de collision se passe dans une arène qui connaît :
— Deux personnages : player, adv
— Table des translations des articulations : player_table, adv_table

Les tables des translations des articulations sont des structures ordonnées par ordre décroissant de la vitesse de chaque articulation d'un personnage associé, mise à jour pour chaque temps t en fonction des changements de coordonnée absolue de l'extermité de fin d'un os du temps $t-1$ au temps t.
L'algorithme se lance si le personnage attaquant est dans le cas d'un mouvement de vecteur nul avec animation ou un mouvement dynamique.
Nous nommerons 'atk' le personnage attaquant et 'def' le personnage adverse, respectivement leurs tables : 'atk_table' et 'def_table'
HIT_DELAY le delai avant nouvelle collision (ici 20 instances de temps)

Input : Arene arene
Output : Articulation attaquante si collision
if *non collision_des_rectangles_de_dimension(atk, def)* **then**
  renvoyer 'rien';
end
for *atk_os* $\in$ *atk_table avec atk_os.vitesse() > 0* **do**
  if *def.collision_avec(atk_os)* **then**
    renvoyer atk_os;
  end
end
renvoyer 'rien';

```
Function collision_avec(def, atk_os)
    collision = False;
    for def_os ∈ def.armature avec def_os.delai_collision = 0 do
        if non collision_images_avec_rotation(def_os, atk_os) then
            continuer;
        end
        for atk_hitbox ∈ atk_os.hitbox do
            if non collision_cercle_et_rectangle_avec_image(atk_hitbox, def_os)
             then
                continuer;
            end
            if collision_hitbox(atk_hitbox, def_os.hitbox) then
                collision = True;
                def_os.delai_collision = HIT_DELAY;
            end
        end
    end
    renvoyer collision;
```

Remarque : un os en mouvement peut toucher plusieurs articulations adverse lors d'un impact.

Ici la vérification déterminant les intersections entre le cercle représentatif de l'élément actuel de la hitbox et les éventuels rectangles le reliant à un de ses élements fils.
On parcourt ici tous les éléments de def_hitbox.

Function *collision_hitbox(atk_hitbox, def_hitbox)*

if $\exists x \in def\_hitbox : collision\_cercles(x, atk\_hitbox)$ **then**
renvoyer True;
end
else if $\exists (x, y) \in def\_hitbox \times def\_hitbox :$
$collision\_cercle\_rectangle(atk\_hitbox, Rectangle(x.centre, y.centre, \frac{x.rayon+y.rayon}{2}))$
then
renvoyer True;
end
else if $\exists (x, y) \in def\_hitbox \times def\_hitbox \exists z \in atk\_hitbox.herites :$
$collision\_rectangles(\begin{matrix} Rectangle(atk\_hitbox.centre, z.centre, \frac{atk\_hitbox.rayon+z.rayon}{2}), \\ Rectangle(x.centre, y.centre, \frac{x.rayon+y.rayon}{2}) \end{matrix})$
then
renvoyer True;
end
else if $\exists x \in def\_hitbox \times def\_hitbox \exists z \in atk\_hitbox.herites :$
$collision\_cercle\_rectangle(x, Rectangle(atk\_hitbox.centre, z.centre, \frac{atk\_hitbox.rayon+z.rayon}{2}))$
then
renvoyer True;
end
renvoyer False;

Chapitre 4

Anexes

4.1 Collisions entre figures géométriques

Donnons des exemples de formules pour les intersections de figures dans le but de montrer que le calcul est rapide et se fait en complexité constante.

4.1.1 Distance d'un point à un segment

Soit D la droite passant par P_α et P_β avec $P_\alpha \neq P_\beta$.
Soit P un point.

$$d(P,D)^2 = \frac{((P_{\alpha_y} - P_{\beta_y})(P_x - P_{\alpha_x}) - (P_{\alpha_x} - P_{\beta_x})(P_y - P_{\alpha_y}))^2}{(P_{\alpha_x} - P_{\beta_x})^2 + (P_{\alpha_y} - P_{\beta_y})^2}$$

On peut poser $V_x = (P_{\alpha_x} - P_{\beta_x})$ et $V_y = (P_{\alpha_y} - P_{\beta_y})$, ce qui nous donne :

$$d(P,D)^2 = \frac{(V_y(P_x - P_{\alpha_x}) - V_x(P_y - P_{\alpha_y}))^2}{{V_x}^2 + {V_y}^2}$$

On définit la droite D_α la droite perpendiculaire à D en P_α.
De la même manière on définit la droite D_β perpendiculaire à D en P_β.
Si $d(P,D_\alpha) < d(P_\alpha,P_\beta)$ et $d(P,D_\beta) < d(P_\alpha,P_\beta)$

$$d(P,[P_\alpha P_\beta]) = d(P,(P_\alpha P_\beta))$$

Sinon si $d(P,D_\alpha) < d(P,D_\beta)$ (la projection othogonale ne se fait plus sur le segment, on prend la distance au point le plus proche)

$$d(P,[P_\alpha P_\beta]) = d(P,P_\alpha)$$

Sinon

$$d(P,[P_\alpha P_\beta]) = d(P,P_\beta)$$

4.1.2 Collision de zones

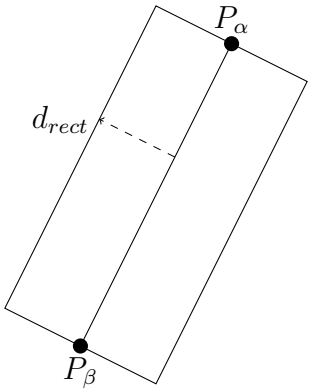
Soit une zone α définie par une origine P_{α_debut} et une extrémité P_{α_fin} telles qu'il existe $(x,y) \in {\mathbb{R}_+^*}^2$ vérifiant : $P_{\alpha\_fin} = (P_{\alpha\_debut_x} + x, P_{\alpha\_debut_y} + y)$.

Soit une zone β définie de la même manière que α avec par une origine P_{β_debut} et une extrémité P_{β_fin}.
Il y a collision entre les deux zones si :

$$((P_{\beta\_fin_x} \geqslant P_{\alpha\_debut_x}) et (P_{\beta\_debut_x} \leqslant P_{\alpha\_fin_x})) et ((P_{\beta\_fin_y} \geqslant P_{\alpha\_debut_y}) et (P_{\beta\_debut_y} \leqslant P_{\alpha\_fin_y}))$$

4.1.3 Distance à rectangle

On définit un rectangle par deux points : P_α, P_β et d_{rect} : une distance orthogonale au segment : $[P_\alpha P_\beta]$, ce qui donne la représentation du rectangle suivante :

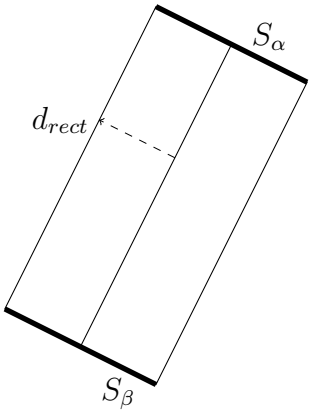


Avec cette défintion, la distance à un rectangle est une variante de la distance à un segment.
D'où D_α et D_β définies de la même manière que le cas du segment.
On définit pour un point P par la rotation de centre C et d'angle θ de la manière suivante : $Rot(P, C, \theta)$
On définit de même pour P l'homothétie de centre C et rapport k de la manière suivante : $Hom(P, C, k)$
On définit les segments S_α et S_β par

$$S_\alpha = [Hom(Rot(P_\beta, P_\alpha, \pi), P_\alpha, \frac{d_{rect}}{d(P_\alpha, P_\beta)}) Hom(Rot(P_\beta, P_\alpha, -\pi), P_\alpha, \frac{d_{rect}}{d(P_\alpha, P_\beta)})]$$

$$S_\beta = [Hom(Rot(P_\alpha, P_\beta, \pi), P_\beta, \frac{d_{rect}}{d(P_\alpha, P_\beta)}) Hom(Rot(P_\alpha, P_\beta, -\pi), P_\beta, \frac{d_{rect}}{d(P_\alpha, P_\beta)})]$$

Autrement dit ce sont les segments S_i, $i \in \alpha, \beta$ tels que leurs extrémités soient les points de la droite D_i à distance d_{rect} de P_i



Si $d(P, D_\alpha) < d(P_\alpha, P_\beta)$ et $d(P, D_\beta) < d(P_\alpha, P_\beta)$

$$d(P, Rectangle) = d(P, [P_\alpha P_\beta]) - d_{rect}$$

Sinon si $d(P, S_\alpha) < d(P, S_\beta)$ (si le point n'est plus encadré entre les segments S_α et S_β et du côté de S_α)

$$d(P, Rectangle) = d(P, S_\alpha)$$

Sinon

$$d(P, Rectangle) = d(P, S_\beta)$$

4.1.4 Intersection de cercle et de rectangle

Soit un cercle de centre C et rayon r.
Soit un rectangle définit comme précédemment.
On dit qu'il y a intersection entre ce cercle et ce rectangle si :

$$d(C, Rectangle) \leqslant r$$

4.1.5 Intersection entre Cercles

Soit un cercle de centre C_α et rayon R_α. Soit un second cercle de centre C_β et rayon R_β. Il existe une intersection de (C_α, R_α) et (C_β, R_β) si et seulement si :

$$d(C_\alpha, C_\beta) \leqslant R_\alpha + R_\beta$$

$$\sqrt{(C_{\beta_x} - C_{\alpha_x})^2 + (C_{\beta_y} - C_{\alpha_y})^2} \leqslant R_\alpha + R_\beta$$

Autrement dit, par croissance de la fonction $x \mapsto x^2$, on a l'inéquation suivante (plus rapide à calculer par machine) :

$$(C_{\beta_x} - C_{\alpha_x})^2 + (C_{\beta_y} - C_{\alpha_y})^2 \leqslant (R_\alpha + R_\beta)^2$$

4.2 Gestion des rafraichissements de l'écran

Dans le but d'éviter les affichages non nécessaires, qui seraient des opérations couteuses et permettre d'investir plus de ressources pour des algorithmes, gestion des événements et autres opérations.
En général on doit tout afficher à l'écran.
Soit **FPS** le nombre d'images par seconde.
Soit **REFRESH_TIME** = 1000 / FPS le temps minimal à attendre avant rafraichissement en millisecondes.
Le procédé qui suit permet un affichage qui ne choque pas l'oeil humain pour un FPS bien choisi (le projet tourne pour FPS = 120).

```
initialisation des données;
besoin_de_rafraichissement = True;
dernier_temps_de_rafraichissement = temps_actuel - REFRESH_TIME - 1;
while il y a des événements à gérer do
    if besoin_de_rafraichissement et dernier_temps_de_rafraichissement +
     REFRESH_TIME < temps_actuel then
        Faire l'affichage;
        besoin_de_rafraichissement = False;
        dernier_temps_de_rafraichissement = temps_actuel;
    end
    Lecture des événements;
    Gestion des événements et éventuelle mise à jour de
     besoin_de_rafraichissement;
end
Libération et arrêt des événements locaux à la fonction;
```

Algorithme 2 : Gestion des événements et affichage