Table des matières

0.0.1 Structure des fichiers

Les Principaux éléments qui composent les scènes graphiques de l'application sont les cameras et l'objet (ou plutôt terrain) visualisé. Dans cette optique, le code source a été distribué en 5 parties principales : tout d'abord, on trouve les 2 modules graphiques, suivis des 4 autres en rapport avec ces entités RenderWare. Les 2 premiers ont chacun une tâche bien définie : l'un devra s'occuper de créer et gérer l'interface, tandis que l'autre initialise l'application proprement dite. Si l'on entre plus dans les détails, on s'apercoit qu'il est nécessaire de posséder deux fichiers pour la gestion des cameras, et deux autres pour le terrain. La raison en est simple, à chaque fois, l'un propose les fonctions et procédures adéquates à l'entité, et l'autre les callbacks déclenchées par l'interface MOTIF.

0.0.2 Structure des données

La librairie RenderWare répondant à la norme C ANSI, la totalité de l'application à été codée à l'aide de ce langage. Vu le grand nombre de structures que cette librairie propose, il n'a nullement été nécessaire d'en créer de nouvelles. Un problème se pose cependant, il s'agit des repères différent selon que l'on se place par rapport à une caméra ou un objet. L'origine de ce problème réside dans le fait que le repère spacial de la caméra la suit dans ses déplacements, c'est à dire qu'elle en est toujours le centre et que de plus, l'axe des profondeurs (Oz en l'occurence) est toujours orienté dans le sens où regarde cette dernière; Ce qui le rend indirect par rapport au repère mathématique standard (figure 0.1).

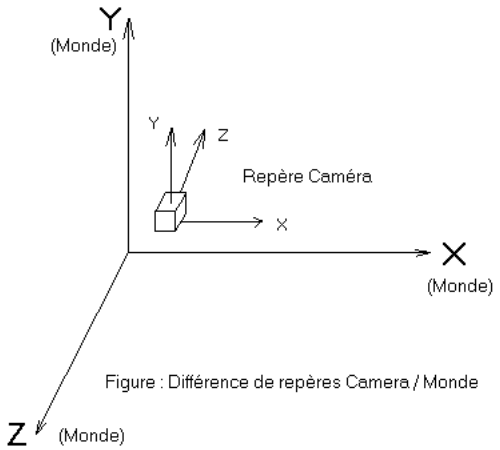

Figure 0.1: Différence de repères

```
Début
Si la camera recule
   alors
       la faire tourner de 180 degrés horizontalement
       pour qu'elle fasse face aux obstacle potentiels;
   finsi

Trouver le polygone le plus proche de la camera et face
à elle;
Calculer la distance qui le sépare de cette dernière;

Si la camera recule
   alors
       la faire tourner de - 180 degrés horizontalement
       pour qu'elle reprenne son sens initial;
finsi

Si la distance calculée est plus petite que celle dont
   on veut avancer
  alors
      Retourner Collision := VRAI;
  sinon
      Retourner Collision := FAUX
finsi

fin
```

Figure 0.2: algorithme de collision en rendu 3D

Les fonctions et procédures concernant les caméras (les déplacements et rotations notamment) ont donc été codées de facon à ce que ces changements de signe des coordonnées et des angles soit transparents pour l'utilisateur.

0.0.3 Algorithmes

Le principal algorithme de ce projet est bien sûr celui de gestion des collisions. Cependant, ce dernier est différent suivant que l'on utilise la camera de visualisation 3D (figure 0.2) ou celle d'édition du terrain "2D" (figure 0.4) .

En ce qui concerne les autres algorithmes, il est interessant de voir celui

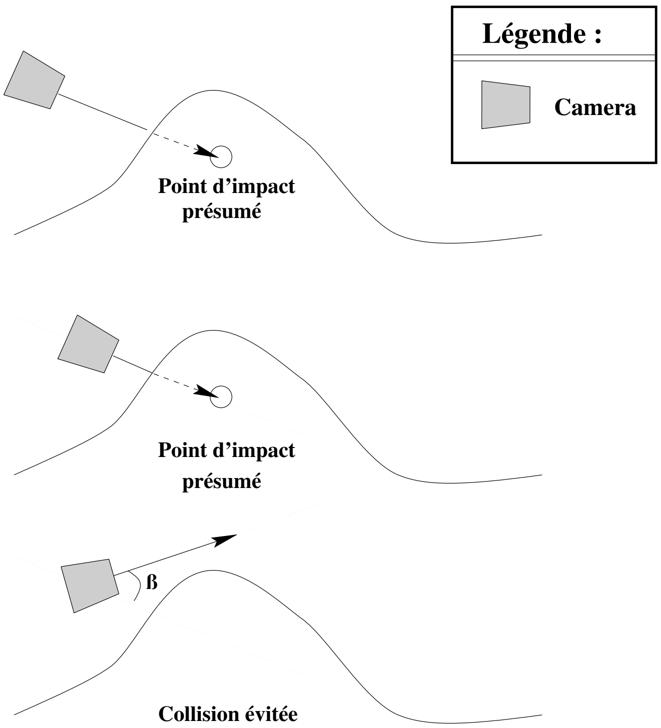

Figure 0.3: Principe d'évitement de collision en rendu 3D

qui permet d'eviter que les cameras ne sortent du terrain qu'elles visualisent (figure 0.5).

Il est toutefois évident qu'il est nécessaire pour mener a bien un déplacement de caméra que les deux algorithmes, collision et sortie de terrain soit combinés pour obtenir un résultat cohérent. L'organigramme des decisions de la figure 0.6 présente ce type de liaisons.

0.0.4 Problèmes rencontrés et Solutions choisies

Malheureusement, RenderWare impose pour implémenter ces types de tests des contraintes incontournables. La première est évidemment : comment se repérer dans le monde vis-à-vis des objets environnants. Cette librairie graphique offre deux fonctions permettant de le faire, mais de prime abord, une seule à été suffisante : RwPickScene. D'un clic sur un des polygones de la scène affichée par une caméra, on récupère un grand nombre de caractéristiques correspondantes (par exemple, le polygone en question, mais aussi le vertex le plus proche, la distance qui le sépare du clic...). Un paradoxe apparait alors. Avant de l'ennoncer, voyons une donnée imposée non pas par RenderWare, cette fois-ci, mais par toutes les librairies graphiques

```
Commentaire : Ne pas perdre de l'esprit que la camera est
              toujours perpendiculaire à son déplacement

Début

Commentaire : On fait en sorte que la caméra fasse face aux
              obstacles potentiels

Cas déplacement
   nord  : Tourner la camera de 90 degrés horizontalement;
   sud   : Tourner la camera de 90 degrés horizontalement;
   est   : Tourner la camera de 90 degrés horizontalement;
           Tourner la camera de 90 degrés latéralement;
   ouest : Tourner la camera de - 90 degrés horizontalement;
           Tourner la camera de - 90 degrés latéralement;
finCas

Trouver le polygone le plus proche de la camera et face à elle;
Calculer la distance qui le sépare de cette dernière;

Cas déplacement
   nord  : Tourner la camera de - 90 degrés horizontalement;
   sud   : Tourner la camera de - 90 degrés horizontalement;
   est   : Tourner la camera de - 90 degrés latéralement;
           Tourner la camera de - 90 degrés horizontalement;
   ouest : Tourner la camera de 90 degrés latéralement;
           Tourner la camera de 90 degrés horizontalement;
finCas

Si la distance calculée est plus petite que celle dont on veut
   avancer,
   alors
       Retourner Collision := VRAI;
   sinon
       Retourner Collision := FAUX
finsi


   fin
```

Figure 0.4: Algorithme d'évitement de collision en rendu 2D

```
Début

   On récupère les points qui forment les deux angles inferieur
    arrière et supèrieur avant de la boîte englobante du terrain
    pour connaitre les limites de celui-ci.

   Cas Caméra
      Camera 2D :

        Si la distance qui sépare l'extrémité qu'atteint la
           caméra
           et
           que cette même distance  est plus petite que celle
           dont on veut se déplacer
          alors
            On bloque la camera
          sinon
             On effectue le déplacement
        Finsi

      Caméra 3D :

        On duplique la Caméra, on effectue le déplacement désiré,
        Si plus rien n'est en dessous de ce clône, on est sorti
          alors
              On est sorti, la Camera 3D restera immobile
  sinon
              On effectue le déplacement
Finsi

fin
```

Figure 0.5: Test de débordement du terrain visualisé

qui disposent de caméras. Pour afficher, on définit des plans de clipping (un "far" ainsi qu'un "near") qui sont, en résumé, les distances entre lesquelles la caméra est capable de distinguer des objets. Cependant, et ceci est à imputer à la version 1.3 de RenderWare, il n'est possible de fixer que la distance à partir de laquelle la caméra n'est plus "aveugle". A ajouter à cela, il apparait que dans cette même version, nous soyons contraints d'être légèrement astygmate. En effet, il est impossible de préciser que l'on veut repérer tout ce qui est à proximité immédiate. Voici donc notre paradoxe : Comment savoir que l'on est trop près d'une paroi si l'on ne voit pas cette même paroi. Lors de déplacements rectilignes, ceci ne pose pas de problèmes (figure 0.3), puisqu'avant d'avancer, on cherche à savoir si l'on n'atterrira pas trop près ou au delà du décor. Cependant, si il vient à l'utilisateur l'idée de s'arrêter , cette fois-ci, rien ne garanti que s'il fait tourner la caméra, elle ne se retrouvera pas nez collé au sol, ou en se placant au niveau des plans de clipping avec une distance objectif-sol plus petite que la valeur minimale autorisée par la caméra.

Pour régler ceci, il aurait été possible de maintenir les mouvements à une distance minimale et suffisante du sol. Inconvénient alors, obtenir des écartements trop grands, éloignant de la réalité.

Une autre solution possible, et c'est celle que nous avons retenu, est de tester une collision de déplacement même lors des rotations statiques. Si un point qui n'est pas directement sous la caméra est déja trop près, celui qui y est l'est encore plus. La caméra prend alors un mouvement de balancier arrière plus proche, selon nous, de la réalité(figure 0.7).

Dans le cas de la caméra d'édition, le problème ne se pose pas, les déplacement sont uniquement transversaux et sans rotation.

Un dernier problème directement raccroché au mouvement est : quand faut-il s'arrêter ? Nous avons décidé d'empêcher que l'on puisse sortir du terrain que l'on visualise. A cet effet, les approches on été différentes. La caméra 2D, de par son orthogonalité permanente avec le sol, ne pose pas de problème, puisqu'il suffit de tester sa position par rapport aux limites du relief. La difficulté apparait avec la caméra de rendu 3D. Comment se repérer par rapport aux bords. Là encore, une solution envisagée, et c'est celle pour laquelle nous avons opté, est de se repérer vis-à-vis du sol immédiatement au dessous. On duplique cette caméra en une autre, "cobaye" , on lui fait effectuer le déplacement prévu (avec test de collision, bien-sûr) et on teste si à l'arrivée, elle ne se trouve pas dans le vide. Dans ce cas, on ne fait rien, sinon, rien n'empêche d'avancer.

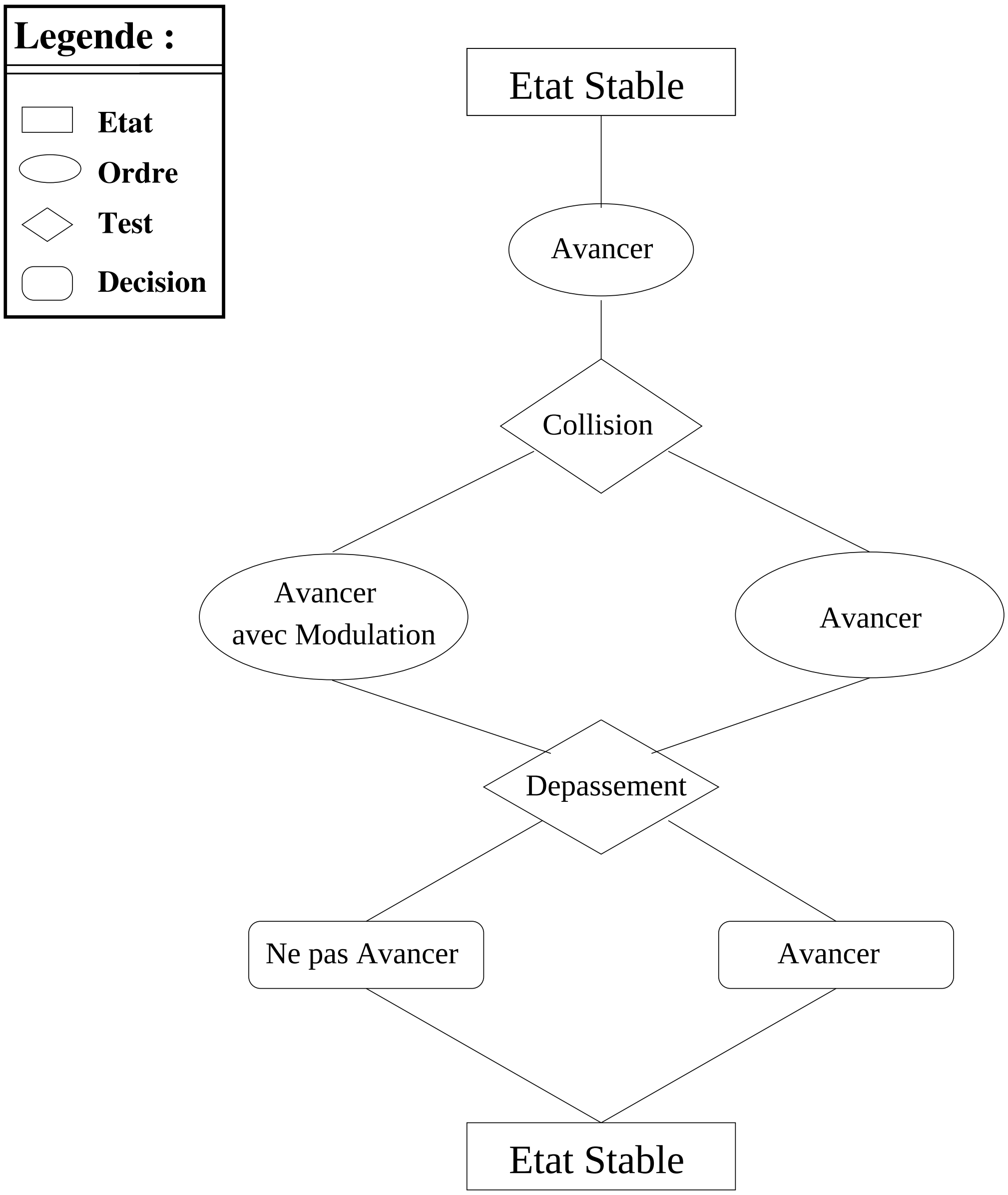

Figure 0.6: Organigramme des décisions prises lors d'un ordre de déplacement

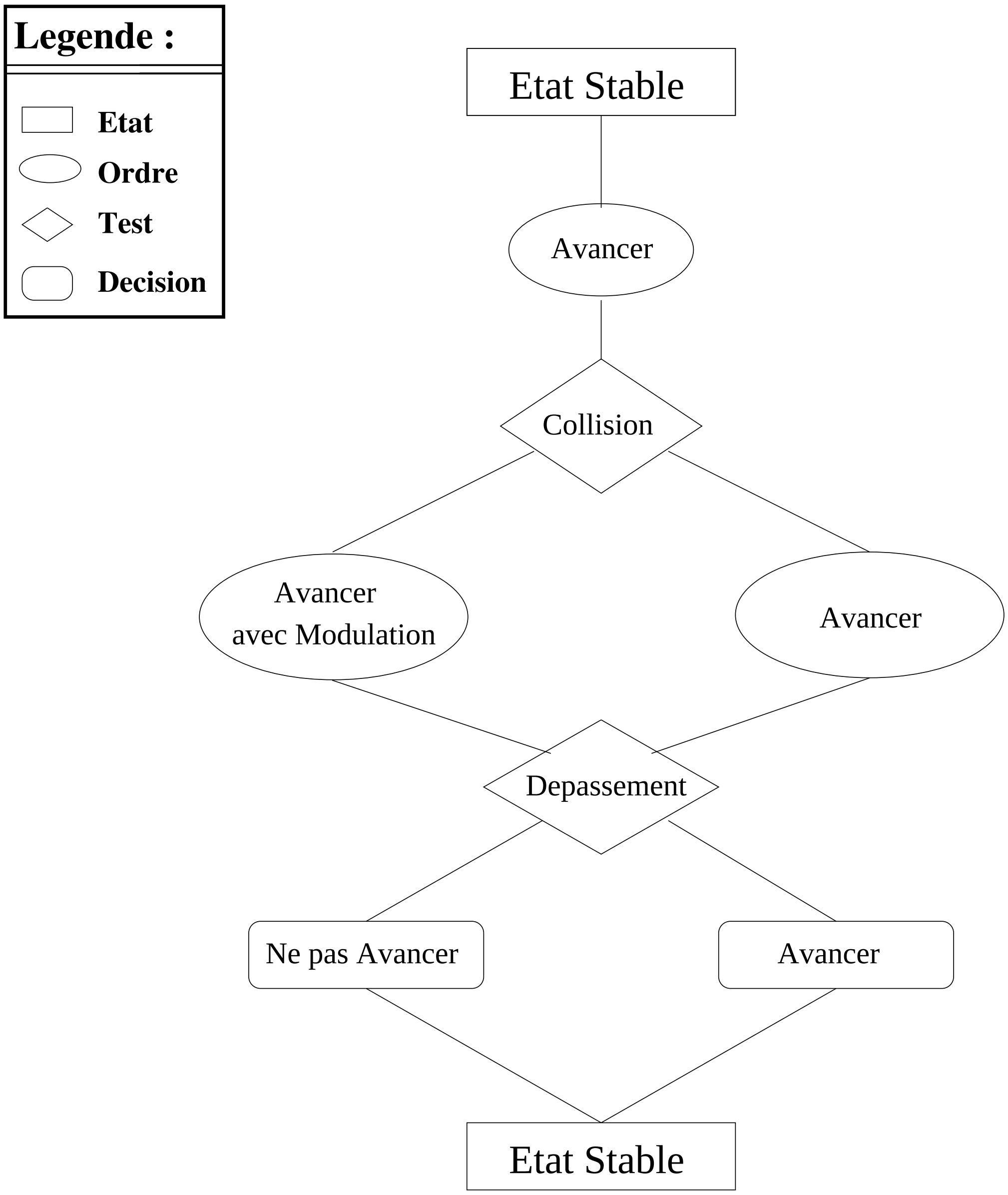

Figure 0.7: Mouvement de balancier lors de rotations statiques