Projet Jeu 3D TSI

Bob l'éponge vs Minions :

Pierre feuille ciseau

SOMMAIRE

PRESENTATION

Dans le cadre du projet de jeu 3D en OpenGL, j'ai choisi de faire un jeu de pierre feuille ciseau sous la forme d'une bataille entre des Minions et Bob l'éponge qui doit défendre son restaurant contre ces envahisseurs. Chacun des minions qui arrive a dans sa main une pierre, une feuille ou un ciseau et Bob doit tirer le bon signe sur chacun des minions pour les neutraliser. Il peut tirer dans un rayon de 30.0f.

J'ai repris le programme_10 du TP en réorganisant les fonctions ainsi que les variables pour rendre le code plus clair à comprendre et à modifier. Dans mon jeu, j'ai choisi de laisser le personnage principal qui est Bob l'éponge toujours en (0,0,0) et de faire bouger tout le reste, c'est plus facile pour le codage des interactions avec les minions et les décors.

Objets 3D

Pour Bob l'éponge, le Minion, ainsi que la pierre, le feuille et le ciseau, j'ai utilisé des modèles au format .obj que j'ai trouvé sur le net après les avoir corrigé sur Blender. Je n'avais pas de textures adaptées à ces objets donc j'ai essayé de mettre des textures simples téléchargées sur le net.

DECORS

Pour les décors, j'ai fait un cube de 100 x 100 centré en (0.0.0) et traversé par un rectangle de largeur 32 symbolisant la route. Pour les textures décor, je les ai faites moi-même sur Photoshop à partir d'images existantes afin d'avoir les bonnes proportions et de bien coller les bords pour donner une impression d'homogénéité.

CAMERA

Le fait de déplacer tous les décors au lieu de déplacer mon joueur me permet de garder la caméra fixe à une distance pratique pour bien voir tous les objets. Je n'ai donc pas à suivre le joueur avec la caméra, j'effectue uniquement des rotations de la caméra autour de l'axe y afin d'admirer les décors sous-marins.

DECLARATIONS JEU

Le fichier *declarations_jeu* contient tous les éléments utilisés dans le jeu :

- Les variables GLU de Bob l'éponge et du texte
- La structure *transformation*
- *L*es coordonnées géométriques du décor
- Les angles de rotation
- Des enums, des booléens qui facilitent le codage.
- Les positions et les signes aléatoires des Minions
- Les textures des décors

CLASSES

CLASSE DECOR

Le fichier *Decor* contient la classe Decor qui permet de créer un décor sous la forme d'un mur sur lequel vient se superposer une texture. Son constructeur prend en paramètre les 4 points définissant le décor et 1 normale :

```
class Decor
{
  public:
    GLuint shader_program_id;
    GLuint vbo_object;
    GLuint vboi_object;
    GLuint texture_id_object;
    int nbr_triangle_object;

    struct transformation
    {
      mat4 rotation;
      vec3 rotation_center;
      vec3 translation;

      transformation():rotation(),rotation_center(),translation(){}
    };

    transformation transformation_model;
    mat4 rotation;
    vec3 rotation_center;
    vec3 translation;

    vec3 p0, p1, p2, p3, n0;

  Decor(vec3 a, vec3 b, vec3 c, vec3 d, vec3 n);
  void init_decor();                                  // initialise le décor
  void afficher_decor();                              // affiche le décor
  void deplacement(float x, float z);                 // déplace le décor

};
```

CLASSE MINION

Le fichier Minion contient la classe Minion qui permet de créer un mignon aux coordonnées x, y et z passées en paramètre du constructeur. Elle permet aussi d'initialiser et d'afficher le Minion créé :

```
class Minion

{
  public:

    GLuint shader_program_id;
    GLuint vbo_object;
    GLuint vboi_object;
    GLuint texture_id_object;
    int nbr_triangle_object;

    struct transformation
    {
      mat4 rotation;
      vec3 rotation_center;
      vec3 translation;

      transformation():rotation(),rotation_center(),translation(){}
    };

    transformation transformation_model;
    mat4 rotation;
    vec3 rotation_center;
    vec3 translation;

    float s,x,y,z, vitesse, vitesseTir;
    mesh m;

  Minion(float a, float b, float c);
  void init_minion();
  void afficher_minion();

};
```

CLASSE SIGNE

Le fichier Signe contient la classe Signe dans laquelle sont contenues 3 sous-classes (Pierre, Feuille, et Ciseau). Il permet donc de créer les signes pour Bob L'éponge ainsi que pour chacun des Minions et de tirer. Son constructeur permet de définir ses coordonnées ainsi que sa vitesse de tir :

```
class Signe

{
  public:

    GLuint shader_program_id;
    GLuint vbo_object;
    GLuint vboi_object;
    GLuint texture_id_object;
    int nbr_triangle_object;

    struct transformation
    {
      mat4 rotation;
      vec3 rotation_center;
```

```
      vec3 translation;

      transformation():rotation(),rotation_center(),translation(){}
    };

    transformation transformation_model;
    mat4 rotation;
    vec3 rotation_center;
    vec3 translation;

    float s, x,y,z, vitesseTir;
    mesh m;

  Signe(float a, float b, float c, float v);
  void init_model();
  void afficher_model();
  bool tirer(bool active);

};
```

MAIN

Dans mon Main, je commence par déclarer 7 décors, 3 minions ainsi que 12 signes (1 pierre, 1 feuille et 1 ciseau pour chacun des minions ainsi que pour Bob l'éponge) :

```
// Declaration Decor
Decor dSol(pt0,pt1,pt2,pt3,n0);
Decor dPlafond(pt4,pt5,pt6,pt7,n1);
Decor dRoute(pt8,pt9,pt10,pt11,n0);
Decor dFond(pt4,pt5,pt1,pt0,n2);
Decor dArriere(pt7,pt6,pt2,pt3,n4);
Decor dGauche(pt7,pt4,pt0,pt3,n5);
Decor dDroite(pt5,pt6,pt2,pt1,n6);
Decor d[]={dRoute,dPlafond,dFond,dArriere,dGauche,dDroite,dSol};

//Declaration Minions
int NbMinions=3;
Minion m0(xM[0],0.0f,zM[0]);
Minion m1(xM[1],0.0f,zM[1]);
Minion m2(xM[2],0.0f,zM[2]);
Minion m[]={m0,m1,m2};

//Declaration pierres/feuilles/ciseaux
Pierre pB(0.4f, 0.5f, 0.0f,3.0f);
Feuille fB(0.5f, 0.5f, 0.0f,3.0f);
Ciseau cB(0.6f, 0.5f, 0.0f,3.0f);

Pierre p0(xM[0]+0.5f,0.5f,zM[0],1.0f);
Feuille f0(xM[0]+0.5f,0.5f,zM[0],1.0f);
Ciseau c0(xM[0]+0.5f,0.5f,zM[0],1.0f);

Pierre p1(xM[1]+0.5f,0.5f,zM[1],1.0f);
Feuille f1(xM[1]+0.5f,0.5f,zM[1],1.0f);
Ciseau c1(xM[1]+0.5f,0.5f,zM[1],1.0f);

Pierre p2(xM[2]+0.5f,0.5f,zM[2],1.0f);
Feuille f2(xM[2]+0.5f,0.5f,zM[2],1.0f);
Ciseau c2(xM[2]+0.5f,0.5f,zM[2],1.0f);

Pierre p[]={pB,p0,p1,p2};
Feuille f[]={fB,f0,f1,f2};
Ciseau c[]={cB,c0,c1,c2};
```

Ensuite je déclare toutes les fonctions que j'ai écrit plus tard (autres que les fonctions GLUT de base :

```
// Fonctions Bob
void init_bob();
void afficher_bob();
// Fonctions Texte
void init_texte();
void afficher_score();

//Fonctions Jeu
bool sauter(bool saut);
void deplacement(float x, float z);
void rotation(float angle);
void defilement();
void test();
```

Ainsi, j'ai gardé les initialisations de Bob l'éponge et du score de la partie ainsi que leurs affichages dans le fichier main car j'ai jugé qu'on n'avait pas besoin de créer une classe pour tous les objets quand on n'a pas besoin d'en créer plus d'un comme ici.

FONCTIONS GLUT

Les fonctions `main()`, `init()`, `display_callback()`, et `load_texture()` sont écrites comme dans le programme_10.

Dans ma fonction `keyboard_callback()`, j'ai définis mes touches comme ceci :

!	Quitter le jeu
q et d	Effectuer des rotations de Bob l'éponge
w et x	Effectuer des rotations de camera
s	Sauter
a z ou e	Tirer une pierre, une feuille, ou un ciseau

Le saut ainsi que les tirs activent des variables booléennes vers les fonctions de saut et de tirs codées plus loin. Le saut prend en compte la possibilité de collisions en limitant les x et les z à partir desquels on peut sauter. Les rotations quant font appel à la fonction rotation() écrite plus loin aussi.

Dans la fonction `special_callback()`, les flèches provoquent des déplacements sous forme de translations selon la rotation de bob l'éponge. Par exemple, l'appuie sur la flèche du haut déplace bob l'éponge dans la direction où il regarde : `x=dL*sin(-angle_y_model_1)` et `z=dL*cos(-angle_y_model_1)`

Elle fait appelle à la fonction `déplacement()` écrite plus loin et gère aussi les collisions en limitant les x et les z à partir desquels on peut se déplacer.

La fonction `timer_callback()` me permet produire mes sauts en faisant appel à la fonction `sauter()`, mes tirs avec Bob l'éponge avec la fonction `Signe.tirer()`, et le défilement avec la fonction `defilement()`.

Le code des fonctions d'affichage du score n'est pas de moi mais de Sidoine Berger qui les a partagés dans le groupe de la promo. Je l'ai repris car je n'avais pas le temps de faire ces fonctions pour afficher le score de mon jeu.

FONCTIONS JEU

La fonction `rotation()` consiste à tourner tout sauf Bob l'éponge dans le sens inverse de la rotation de ce dernier pour donner l'illusion que c'est lui qui tourne dans l'autre sens.

De même que pour la fonction `rotation()`, la fonction `deplacement()` consiste à déplacer tout sauf Bob l'éponge dans le sens inverse du déplacement de ce dernier pour donner l'illusion que c'est lui qui se déplace dans l'autre sens.

Dans la fonction `sauter()`, j'effectue un saut parabolique d'un angle de π/4 de hauteur maximale 4. Je translate donc tout le reste dans le sens inverse de ce saut.

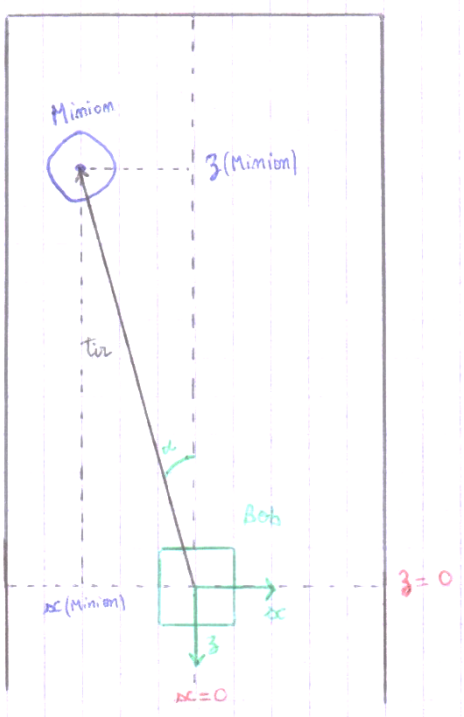
Dans la fonction `defilement()`, Je fais arriver les 3 minions du fond du décor de plus en plus vite et à des positions aléatoires suffisamment écartées pour pas qu'ils se chevauchent entre eux. Chacun se voit attribuer un signe aléatoire entre les 3 enums PIERRE FEUILLE ou CISEAU.

Je commence par tester les conditions de collisions (hors signes) qui font un « game over » qui sont :

- Le fait d'atteindre le fond arrière qui est le décor du restaurant.
- Le fait qu'il y ait collision avec Bob l'éponge.

Lorsqu'il y a « game over », j'ai choisi la facilité en remplaçant le texte « score » par un texte « game over » et en perturbant la boucle d'affichage pour rendre les interactions impossibles.

Ensuite, en cas de tirs, j'envoie la position du signe tiré et l'indice du Minion concerné à la fonction `test()`. Cette fonction commence par tester les collisions entre le signe envoyé par Bob l'éponge et celui que porte le Minion. J'utilise la distance et l'angle du tir pour comparer la position du projectile à la position du Minion portant le signe visé comme ceci.



On connait déjà la position du Minion. Bob tir dans la direction où il regarde, donc on peut déterminer x et z du signe tiré pour qu'il y ait collision :

$$tir_{collision} = \sqrt{x_{Minion}{}^2 + z_{Minion}{}^2} \quad \text{(Théorème de Pythagore)}$$

On doit comparer les angles aussi, on connaît l'angle de tir donc on détermine l'angle auquel doit se trouver le Minion :

$$\tan \alpha = x_{Minion}/z_{Minion}$$

$$\alpha = \tan^{-1}(x_{Minion}/z_{Minion})$$

S'il y a un écart inférieur à 0.5 entre eux je considère qu'il y a collision.

Dans le cas où il y a collision entre les signes c'est là que je qui compare les signes :

- Si je gagne mon pierre feuille ciseau, mon score augmente de 1 et le Minion touché est réinitialisé.
- Si je perds mon pierre feuille ciseau, c'est « game over » je quitte le jeu
- S'il y a égalité rien ne se passe.

CONCLUSION

Ce projet m'a beaucoup passionné dans la mesure où j'avais du plaisir à faire le jeu.

Il marche relativement bien mais il y a encore quelques petits problèmes de collisions entre les projectiles et les Minions. J'ai aussi un petit problème de collision des projectiles avec le fond du décor lorsque je m'en rapproche trop.

Si j'avais eu un peu plus de temps, j'aurai fait plus de tests pour perfectionner le jeu. Je pourrai aussi améliorer la jouabilité en rajoutant la possibilité d'appuyer sur 2 touches à la fois, ou de jouer au clavier et à la souris en même temps tel un véritable FPS.