J’ai surligné ou mis en rouge les chose à changer il manque des classes à commenter

Rapport de projet « Casse-Briques »

# Présentation de l’interface

Dans notre interface, nous avons sur la fenêtre principale en haut le nombre balles restante, le score et le niveau dans lequel nous sommes.

Nous avons un bouton qui nous offre trois propositions : entamer une nouvelle partie, aller dans les paramètres ou fermer le jeu. Appuyer sur paramètre nous renvoi sur une boite de dialogue où l’on peut choisir la largeur du palet, l’activation de la caméra ou non.

# Présentation des classes

Notre application se décompose en plusieurs classes.

Le premier type de classes est les objets que l’on fait apparaitre dans OpenGl  et traiter le jeu: balle, brique, palet, mur et cassebriques. Notre classe bloc est une classe virtuelle que l’on utilise pour la gestion des collisions.

Le deuxième type de classe est la gestion des données des joueurs qui permettent de sauvegarder les scores et les afficher : joueur, listesjoueurs.

Le troisième type de classe regroupe les classes permettant d’afficher / traiter les informations des fenêtres : parametresdialog, mainwindow, joueursdialog.

Et nous avons la classe traitant la caméra : camera.

# Etat de finalisation de l’application

Toutes les fonctionnalités suivantes sont implémentées dans notre application :

* Déplacement du palet commandé par le déplacement de la main du joueur à partir de la WebCam.
* Rebond de la boule sur les murs, sur le palet et sur les briques.
* Destruction des briques (disparition) lorsqu’elles sont touchées par la boule.
* Rebondissement sur le palet et contrôle de la direction de la boule en fonction du point d’impact sur le palet.
* Décompte des boules utilisées et contrôle de la fin de partie.
* Génération aléatoire d’un nouveau niveau avec une vitesse supérieure de la boule par rapport au niveau précédent.
* Calcul des points.
* Choix de la taille du palet par le joueur.
* Sauvegarde du score/nom de joueur et affichage du classement.
* Quoi d’autre ?

# Fichiers d’entête

class Balle

{

public:

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Constructeurs & Destructeur \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Constructeur pouvant servir au debug (inutile actuellement)

Balle(float x, float y, float dirX, float dirY);

// Constructeur utilisé dans le programme

Balle(Palet \*palet, int niveau);

// Destructeur

~Balle();

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Setters & Getters \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Direction de la balle

void **setDirection**(const float x, const float y) {m\_direction[0]=x;m\_direction[1]=y;} change les valeurs de la direction de la balle.

float **getDirectionX**() const {return m\_direction[0];} renvoi la direction en X de la balle

float **getDirectionY**() const {return m\_direction[1];} renvoi la deriction en Y de la balle

// Centre de la balle

Les 4 fonctions suivantes donne ou change les valeurs du centre de la balle.

float **getCentreX**() const {return m\_positionCentre[0];}

float **getCentreY**() const {return m\_positionCentre[1];}

void **setCentreX**(float x) {m\_positionCentre[0] = x;}

void **setCentreY**(float y) {m\_positionCentre[1] = y;}

// Rayon de la balle

float **getRayon**() const {return m\_rayon;}renvoi le rayon de la balle

// Etat de la balle : est-elle sur le palet ?

bool **getEstSurPalet**() const {return m\_estSurPalet;}renvoi un booléen qui nous informe si la balle est en position initiale sur le palet

void **setEstSurPalet**(bool etat) {m\_estSurPalet = etat;}nous pouvons changer le booléen qui nous informe si la balle est sur le palet.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Déplacement \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void **deplacer**();Deplace la balle si elle n’est pas en position initiale

void **envoyerBalle**(); Enlève la balle de la position initiale et utilise la méthode deplacer de la meme classe.

// Affichage de la balle en fonction du vecteur m\_positionCentre

void **Display**();

private:

GLUquadric \*m\_sphere; // Quadrique permettant d'afficher la sphère

float m\_rayon;

float m\_couleur[3];

float m\_vitesse;

float m\_direction[2]; // un vecteur de coord. x et y dont les valeurs sont entre 0 et 1

float m\_positionCentre[2];

bool m\_estSurPalet; est à true quand la balle est en position initiale de début de partie.

};

// Classe virtuelle permettant d'implémenter le contexte du jeu (palet, murs, sol, briques)

class Bloc

{

public:

Bloc();

virtual ~***Bloc***();

// Les objets héritant de cette classe doivent s'afficher, et gérer des collisions avec des balles

virtual void ***Display***() = 0; affiche l’objet

virtual bool ***collision***(Balle\* &balle) = 0;vérifie si une collision entre le bloc et la balle.

virtual void ***traiterCollision***(Balle\* &balle) = 0;si il y a collision on change la direction de la balle.

protected:

float m\_points[4][2]; // Contient tous les points du bloc, sous la forme x1,y1,x2,y2,x3,y3,x4,y4.

float m\_couleurs[3];vecteur qui contient les composante RGB du bloc

float m\_largeur;

float m\_hauteur;

};

class Brique : public Bloc

{

public:

Brique(float x, float y, float largeur);contructeur de la classe brique

virtual ~***Brique***();

virtual void ***Display***();

virtual bool ***collision***(Balle\* &balle); vérifie si une collision entre le bloc et la balle.

virtual void ***traiterCollision***(Balle\* &balle); si il y a collision on change la direction de la balle.

private:

float m\_position[2]; // Position du coin supérieur gauche de la brique

};

// Classe dediee a l'affichage d'une scene OpenGL

class CasseBriques : public QGLWidget

{

Q\_OBJECT

public:

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Constructeur & Destructeur \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

CasseBriques(Camera\* camera, QWidget \* parent = nullptr);

~***CasseBriques***();

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Etat du jeu \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Initialisation du jeu

void **initialiserJeu**();

// Stoppe la partie

void **stopJeu**();

// Recommencer une partie depuis le début

void **nouvellePartie**();

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Actions sur le palet \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void **setLargeurPalet**(float largeur); définit largeur du palet

float **getLargeurPalet**() const {return m\_largeurPalet;}renvoi la valeur de la largeur du palet

void **deplacerPalet**(float x);déplacement du palet suivant x

protected:

// Fonction d'initialisation de l'affichage

void ***initializeGL***();

// Fonction de redimensionnement

void ***resizeGL***(int width, int height);

// Fonction d'affichage

void ***paintGL***();

// Fonctions de gestion des interactions

void ***keyPressEvent***(QKeyEvent \* event);

void ***mouseMoveEvent***(QMouseEvent \*event);

private slots:

// Mise à jour des variables du jeu (lié au timer de mise à jour du jeu : m\_timerGame)

void **updateGame**();

private:

/\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Méthodes \*/

/\*\*\*\*\*\*\*\*\*\*\*\*/

// Charger une configuration de briques à partir d'un fichier

void **chargerNiveau**();

// Traitement des collisions pendant une partie

void **traitementCollisions**();

// Teste si la partie doit continuer ou non

void **testJeuEnCours**();

/\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Attributs \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Caméra

Camera\* m\_camera;

// Timers pour la mise à jour

QTimer m\_timerGL;

QTimer m\_timerGame;

// Eléments du jeu

Palet \*m\_palet;

std::vector<Mur \*> m\_murs;

std::vector<Balle \*> m\_balles;

std::vector<Brique \*> m\_briques;

// Attributs de configuration d'une partie

unsigned int m\_nombreBallesInitial;

unsigned int m\_nombreBallesRestantes;

unsigned int m\_nombreBallesEnCours;

double m\_score;

int m\_niveau;

// Attributs de configuration de l'espace de jeu

int m\_briquesParLigne;

int m\_briquesParColonne;

float m\_largeurBrique;

float m\_espaceEntreBriquesLigne;

float m\_espaceEntreBriquesColonne;

float m\_largeurPalet;

// Attributs pour la condition de certaines réalisations

bool m\_collision; // Indique s'il y a déjà eu une collision sur une brique dans une exécution paintGL. Permet d'éviter un double inversement de direction de la balle.

bool m\_balleSurPalet; // Indique si une des balles présentes en jeu est sur le palet (ie n'a pas encore été lancée)

bool m\_perdu; // Indique si le joueur a perdu

bool m\_gagne; //Indique si le joueur a gagné

};

class Mur : public Bloc

{

public:

Mur(float p[][2], int type, float coord);constructeur qui prend les coordonnées des vertex en entrée, si c’est un mur droite gauche ou en haut et les valeurs que la balle ne doit pas depasser.

virtual ~***Mur***();destructeur

virtual void ***Display***();

virtual bool ***collision***(Balle\* &balle);

virtual void ***traiterCollision***(Balle\* &balle);

private:

int m\_type; // gauche = 1 / droite = 2 / haut = 3

float m\_coord; // valeur de x ou y que la balle ne doit pas dépasser

bool m\_visible;

};

class Palet : public Bloc

{

public:

Palet(float x, float y, float largeur, float hauteur, float xMin, float xMax);constructeur qui prend en entrées la position la largeur, la hauteur.

virtual ~***Palet***();

virtual void ***Display***();

virtual bool ***collision***(Balle\* &balle);

virtual void ***traiterCollision***(Balle\* &balle);

void **decaler**(const float x, const float y); // Applique une translation au palet

float **getCentreX**() const {return m\_position[0];}renvoi le centre du palet en X

float **getCentreY**() const {return m\_position[1];} renvoi le centre du palet en Y

float **getHauteur**() const {return m\_hauteur;} renvoi la hauteur du palet

float **getVitesse**() const {return m\_vitesse;} renvoi la vitesse du palet

float **getAngleMin**() const {return m\_angleMin;}renvoi l’angle minimum que pourra avoir la balle (par rapport à l’horizontale)

void **setLargeur**(float largeur);modifie la largeur du palet

void **setCentreX**(float positionCentreX);modifie la position du palet

float **getLargeur**() const {return m\_largeur;}renvoi la largeur du palet

private:

float m\_vitesse;vitesse du palet

float m\_position[2]; // Position du centre

float m\_xMin;valeur de x minimal où se situe le palet

float m\_xMax; valeur de x maximal où se situe le palet

float m\_angleMin; // L'angle maximal dont peut dévier la balle en touchant le palet

int m\_codeDerniereCollision; // 1 : gauche / 2 : haut / 3 : droite / 0 : pas de collision

};