PROJET CASSE-BRIQUE

LYSA GRAMOLI – CLÉMENT ROUAULT

Table des matières

I. Spécifications 2
A. Fonctionnalité liée à l'interaction avec l'utilisateur 2
B. Interface graphique 2
II. Conception 3
A. Diagramme des classes 3
B. Classes utilisées 3
III. Finalisation 5
IV. Code des classe (header) 5
A. MyGLWidget 5
B. Model 6
C. Reflector 8
D. Brick 8
E. Paddle 9
F. Wall 10
G. Ball 11
H. Player 12

I. Spécifications

Dans cette partie, nous allons décrire les fonctionnalités de notre programme. Le programme doit pouvoir interagir avec l'utilisateur.

A. Fonctionnalité liée à l'interaction avec l'utilisateur

Le jeu devra :

- Indiquer au joueur lorsqu'il aura gagné ou perdu
- Créer une partie
- Quitter une partie
- Indiquer le score du joueur
- Utiliser la caméra afin de savoir si le joueur souhaite aller à gauche ou à droite avec sa main

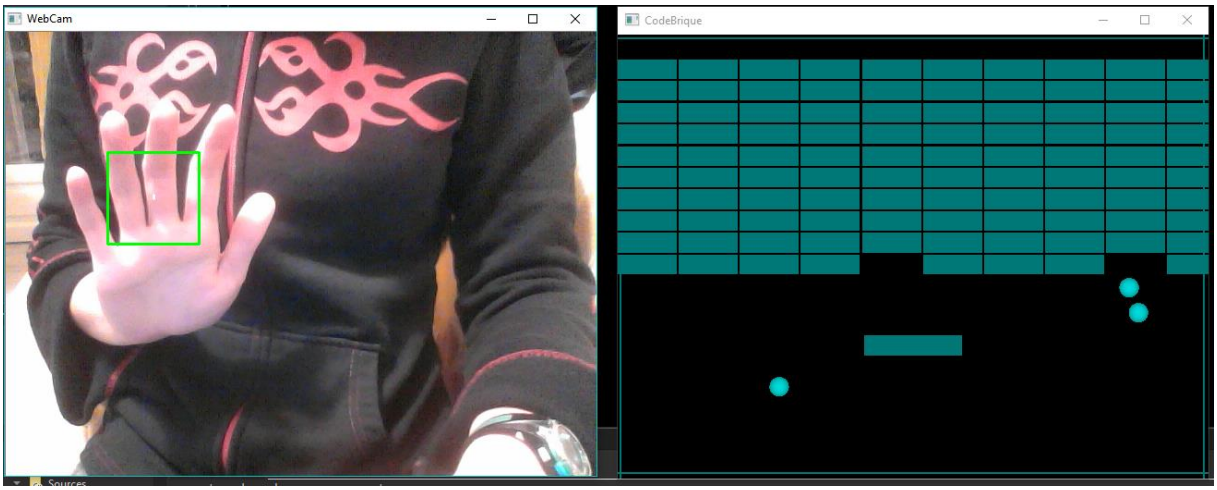
Le programme devra aussi respecter les règles du jeu :

- Lorsque la balle touche un mur ou le palet, elle rebondit
- Lorsque la balle touche une brique, elle rebondit et supprime la brique
- Lorsque la balle touche le mur du bas, le joueur perd un point de vie

B. Interface graphique

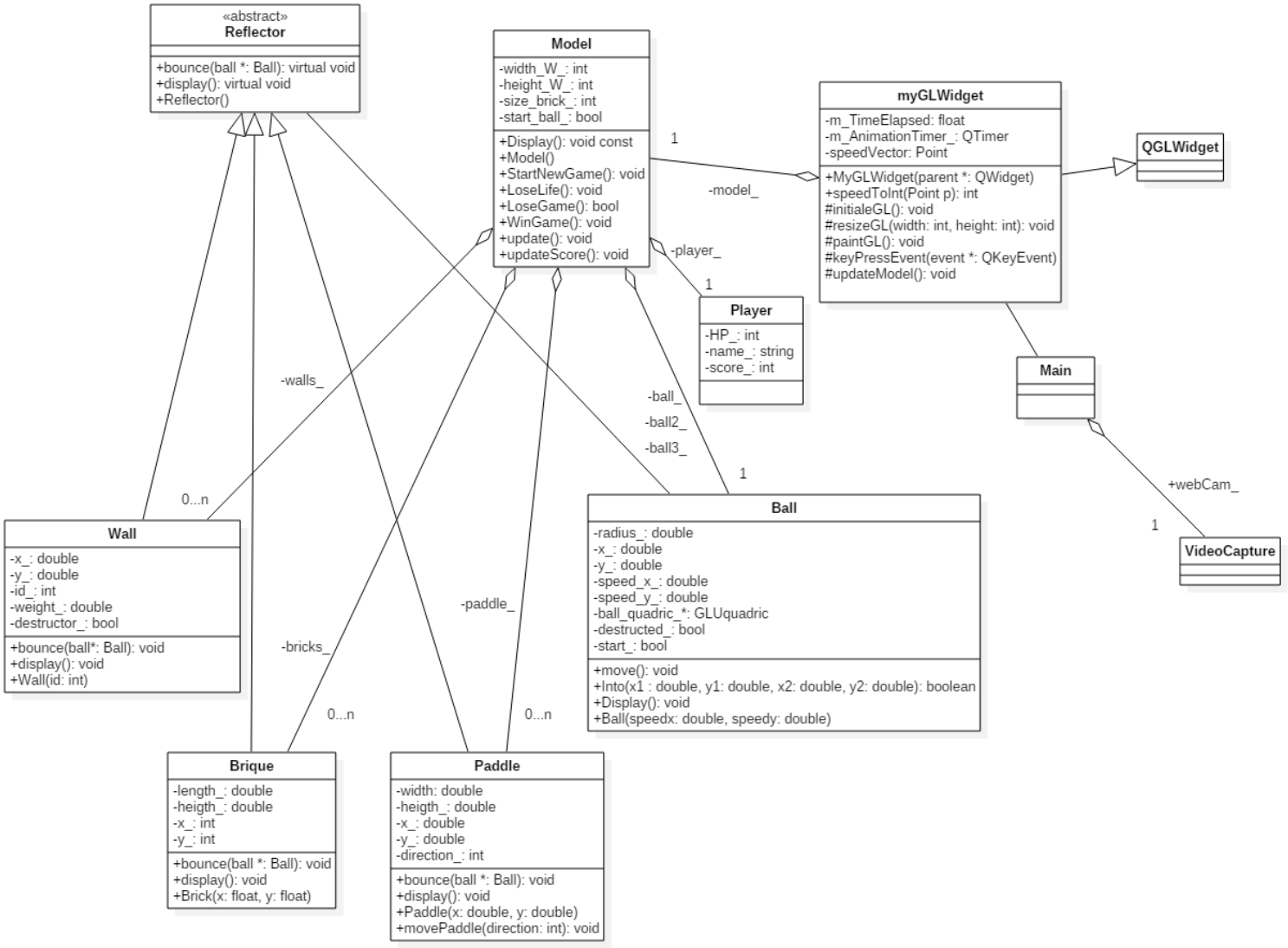
L'outil utilisé pour la représentation graphique est OpenGL. Le jeu deux fenêtres :

- Une fenêtre de jeu : Elle comportera une zone où les briques seront affichées et une zone pour le palet.
- Une fenêtre pour la webcam



II. Conception

A. Diagramme des classes



B. Classes utilisées

- **MyGLWidget :**

  Classe contenant une instance de Model, et permettant de gérer l'affichage des différents objets.

- **Model :**

  Classe contenant tous les objets du jeu (briques, balles, palet, joueur, murs) et des variables nécessaires à certaines fonctions.

- **Reflector :**

  Classe abstraite pour tous les objets sur lesquels les balles peuvent rebondir.
- **Brick :**

  Classe représentant une brique. On retient la coordonné en bas à droite de la brique pour la positionner.
- **Paddle :**

  Classe représentant le palet. On retient la coordonné en bas à droite de la brique pour le positionner.
- **Wall :**

  Classe représentant les murs. On n'a finalement pas besoin de 4 instances de cette classe, et même pas besoin de cette classe pour jouer, mais nous l'avons gardée car elle contient la fonction de représentation des 4 murs.
- **Ball :**

  Classe représentant une balle. On retient la coordonné du centre de la balle pour la positionner.
- **Player :**

  Classe représentant les données d'une partie en cours.

III. Finalisation

L'application finale permet de commencer une partie dès qu'elle est lancée. Les balles bougent et les collisions avec les briques et les murs fonctionnent bien. Le paddle renvoie la balle dans une direction dépendant de l'endroit où la balle touche le paddle, mais pas de manière satisfaisante. Il y a un bug lorsque le joueur bouge le palet et intercepte la balle avec le côté du palet à un instant particulier : la balle reste bloquée sur le paddle pendant quelques instants. Nous avons réduit la marge dans laquelle ce bug arrivait, et il est très difficile de le reproduire maintenant.

Le paddle peut être dirigé de deux façon. En appuyant sur les touches directionnelles droite et gauche le paddle se dirige dans une des directions, puis s'arrête lorsqu'il atteint le bord ou qu'on appuie sur haut. On peut également le diriger via la caméra dans le rectangle vert qui détecte les mouvements. Ce mode de contrôle ne marche pas bien et déplacer le paddle d'un bout à l'autre de l'écran est laborieux.

Lorsqu'une balle touche le mur du bas, on retire un point de vie au joueur. Il y a un problème lorsque plus d'une balle touchent le mur du bas en même temps : un seul point de vie est retiré.

Le score de la partie est affiché dans la console à chaque instant. Nous n'avons pas d'affichage direct dans la fenêtre de jeu.

IV. Code des classe (header)

A. MyGLWidget

```
/*
* Classe héritée de QGLWidget permettant de créer l'interface graphique du
jeu et d'interagir avec l'utilisateur.
*/

class MyGLWidget : public QGLWidget
{
    Q_OBJECT

public:

    //Constructeur
    MyGLWidget(QWidget * parent = nullptr);

    //Setters
    void setSpeedVector(Point p);

    // Méthode permettant de transformer un objet Point en un int. Le but
est de récupérer la direction du vecteur enregistré par la caméra
    int speedToInt(Point p);

    //Getters
    Point getSpeedVector();
protected:

    // Fonction d'initialisation
    void initializeGL();
```

```
    // Fonction de redimensionnement
    void resizeGL(int width, int height);

    // Fonction d'affichage
    void paintGL();

    // Fonction de gestion d'interactions clavier (Ne fonctionne pas avec
lorsque la caméra est activée)
    void keyPressEvent(QKeyEvent * event);

    //Méthode appelant la méthode update() de la classe Model afin de
mettre à jour les composants
    void updateModel();

private:
    // Timer d'animation
    float m_TimeElapsed { 0.0f };
    QTimer m_AnimationTimer_;

    //Attribut contenant la vitesse du vecteur enregistré par la caméra
    Point speedVector_;

    //Pointeur permettant de créer un model dans la fenêtre
    Model* model_;
};
```

B. Model

```
class Model
{
public:

    //Constructeur
    Model();

    //Méthode permettant de commencer une partie. Elle est appelée dans le
constructeur et après la perte d'une partie. Elle initialise tous les
composants.
    void StartNewGame();

    //Méthode gérant les points de vie du joueur. Le joueur perd une vie à
chaque balle touchant le mur du bas
    void LoseLife();

    /*Méthode gérant la perte d'une partie. Le joueur perd une partie quand
tous ses points de vies sont épuisés.
     *  Renvoie true si le joueur a perdu et false sinon
     */
    bool LoseGame();

    //Méthode gérant la victoire du joeur. Celui-ci gagne si toutes les
briques sont cassées et qu'il a encore des points de vie.
    void WinGame();
```

```
    //Getters
    int get_H_Wid();
    int get_W_Wid();
    bool get_start_ball_();

    //Méthode permettant d'afficher les composants du jeu
    void Display() const;

    //Méthode permettant de mettre à jour les composants
    void update();
    void updateScore();

    //Setters
    void setDirectionPaddle(int direction);
    void set_start_ball_(bool start);


private:

    //Vecteur contenant toutes les briques du jeu
    vector<Brick *> bricks_;

    //Vecteur contenant tous les murs du jeu
    vector<Wall *> walls_;

    //Pointeurs pointant sur 3 objets Ball : Permet de créer 3 balles
    Ball *ball_;
    Ball *ball2_;
    Ball *ball3_;

    //Création du Palet
    Paddle *paddle_;

    //Création du joueur
    Player *player_;

    //Pointeur pointant vers la fenêtre principale
    MainWindow *main_;

    //Attribut permettant de savoir la taille de la fenêtre
    int width_W_;
    int height_W_;

    //Attribut stockant l'entier permettant se savoir si le palet doit se
déplacer à gauche (=1), à droite (=-1) ou doit rester immobile (=0)
    int direction_paddle_;

    //Stockage de la taille du vecteur contenant les briques afin de le
comparer au nombre de briques brisées
    int size_brick_;

    //Booléen à true pour pouvoir déplacer les balle, false sinon
    bool start_ball_;
};
```

C. Reflector

```
class Reflector
{

public:
    //Constrcteur
    Reflector();

    //Méthode virtuelle qui gérera l'affichage dans les classes héritantes
    virtual void Display()const =0;

    //Méthode vituelle qui gérera le rebond dans les classes héritantes
    virtual void Bounce(Ball* b)=0;


};
```

D. Brick

```
class Brick : public Reflector
{

public:

    //Constructeur
    Brick(float x, float y);

    //Destructeur
    ~Brick();

    /* Méthode issue de la classe abstraite Reflector
     * permettant de faire rebondir la balle lorsque celle-ci touche la
brique et de détruire la brique touchée
     */
    void Bounce(Ball *b);

    //Getters
    int getX();
    int getY();
    int get_nb_broken_brick();
    bool getHit();

    //Setters
    void setX(int x);
    void setY(int y);
    void setHit(bool b);

    /* Méthode issue de la classe abstraite Reflector
     * permettant d'afficher la brique
     */
    void Display() const;
```

```
private:

    //Attribut permettant de savoir combien de briques ont été touchées
    int nb_broken_brick_;

    // Attribut qui permet de savoir si la brique a été touchée à cet
instant
    bool hit_;

    //Coordonnées en bas à droite de la brique
    GLfloat x_;
    GLfloat y_;

    //Hauteur et longueur de la brique
    double heigth_;
    double width_;

    //Permet de stocker le chemin pour la texture des briques
    QString imPath_;

    //Renvoie true si la brique a été touchée par la balle et false sinon
    bool destroy_;

};
```

E. Paddle

```
class Paddle : public Reflector{
public:

    //Constructeur
    Paddle(double x, double y);

    //Destructeur
    ~Paddle();

    /* Méthode issue de la classe abstraite Reflector
     * permettant de faire rebondir la balle lorsque celle-ci touche le
paddle
     */
    void Bounce(Ball *b);

    //Setters
    void setX(double x);

    /* Méthode issue de la classe abstraite Reflector
     * permettant d'afficher le paddle
     */
    void Display() const;

    //Getters
    double getX();

    //Méthode permettant de faire bouger le paddle selon une direction
donnée en paramètres (0 : immobile, 1: gauche ou -1: droite)
    void movePaddle(int direction);


private:
```

```
    // Enregistre la coordonnée du point en bas à droite du paddle
    double x_;
    double y_;

    //Hauteur et longueur du paddle
    double heigth_;
    double width_;

    //Attribut contenant la direction du paddle (0 : immobile, 1: gauche ou
-1: droite)
    int direction_;

};
```

F. Wall

```
class Wall : public Reflector{
public:

    //Constructeur
    Wall(int id);

    //Destructeur
    ~Wall();

    /* Méthode issue de la classe abstraite Reflector
     * permettant de faire rebondir la balle lorsque celle-ci touche un mur
et de détruire la balle si elle touche le mur du bas
     */
    void Bounce(Ball *b) ;

    /* Méthode issue de la classe abstraite Reflector
     * permettant d'afficher le mur
     */
    void Display() const;

    //Getters
    int getId();
    void setId(int id);
    bool getDestructor();

    //Setters
    void setDestructor(bool destructor);

private:

    //Numéro du mur, afin de déterminer si c'est le mur du bas (= 4), du
haut (= 2), de gauche (= 1) ou de droite (= 3)
    int id_;

    //Longueur du mur (= 30 si c'est le mur du haut et bas et = 12 sinon)
    double weight_;

    //Coordonnées en bas à droite du mur qui est un parallélépipède
    double x_;
    double y_;

    //Renvoie true si la balle a touchée le mur du bas et false sinon
```

```
    bool destructor_;
};
```

G. Ball

```
class Ball
{
public:
    //Constructeur
    Ball(double speedx, double speedy);
    //Destructeur
    ~Ball();
    //Méthode permettant à la balle de bouger
    void Move();
    //Méthode permettant de savoir si la balle touche un mur, une brique ou
un paddle. Retourne true si c'est le cas.
    boolean Into(double x1, double y1, double x2, double y2);

    //getters
    double getx_();
    double gety_();
    double getspeedx_() const;
    double getspeedy_() const;
    double getRadius_() const;
    bool get_start_();
    bool get_destructed_();

    //setters
    void setx_(double x);
    void sety_(double y);
    void setspeedx_(double speedx);
    void setspeedy_(double speedy);
    void set_start_(bool start);
    void set_destructed_(bool destructed);

    //Méthode permettant d'afficher la balle
    void Display() const;

private:

    //Rayon
    double radius_;
    //Coordonnées du centre de la balle
    double x_;
    double y_;
    //Vitesse de la balle en x et y
    double speedx_;
    double speedy_;
    //Création du modèle graphique de la balle
    GLUquadric *ball_quadric_;
    //Attribut à false si la balle ne touche pas le mur du bas et à true
sinon
    bool destructed_;
    //Attribut à false en début de partie pour éviter que la balle ne parte
sans l'accord du joueur, il doit être à true pour déplacer la balle
    bool start_;


};
```

H. Player

```
class Player
{
public:

    //Constructeur
    Player(string name);

    //Getters
    int getHP();
    int getScore();

    //Setters
    void setHP(int HP);
    void setScore(int score);

private:

    //Attribut contenant le score
    int score_;

    //Attribut contenant le nom du joueur
    string name_;

    //Attribut contenant ses points de vie
    int HP_;

};
```