Projet - Bibliothèques de Développement Multimédia

Paul Boutet, Ismaïl El Gote

Mai 2025

Contents

1 Spécifications et interface utilisateur

L'application repose sur un modèle de navigation par fenêtres, structuré autour de trois interfaces principales, chacune ayant un rôle bien défini dans l'expérience utilisateur.

1.1 Fenêtre Principale (MainWindow)

La fenêtre principale constitue le point d'entrée de l'application. Elle propose un menu d'accueil offrant à l'utilisateur trois choix : lancer une nouvelle partie, accéder aux paramètres ou ouvrir le rapport sous format PDF.

1.2 Fenêtre des Paramètres (SettingsWindow)

La fenêtre des paramètres permet à l'utilisateur de configurer les options liées à la caméra. Elle intègre un composant spécialisé, le *CameraWidget*, qui affiche un aperçu en direct du flux vidéo. L'utilisateur peut y activer ou désactiver le seuillage, et visualiser simultanément le flux en couleur ainsi qu'en niveaux de gris. L'objectif est ici de permettre à l'utilisateur d'adapter l'image à sa situation.

Le flux vidéo visible ici permet également de tester la détection du poing, que le seuillage soit activé ou non.

1.3 Fenêtre de Jeu (GameWindow)

La fenêtre de jeu constitue le cœur de l'application. Elle intègre un espace graphique 3D, rendu à l'aide du composant *GameWidget* basé sur OpenGL, qui occupe la majeure partie de l'écran. Un affichage en surimpression (HUD) présente en temps réel le score du joueur ainsi que le nombre de vies restantes, représentées par des croix (X).

Le flux vidéo de la caméra est également affiché en temps réel dans un coin de l'écran, avec la possibilité pour l'utilisateur de le masquer ou de l'afficher à l'aide de la touche **Espace**. Cette fenêtre assure également la gestion des événements de jeu en temps réel, garantissant une expérience interactive fluide et réactive.

1.4 Captures d'Écran

Voici trois captures illustrant les principales interfaces de l'application :

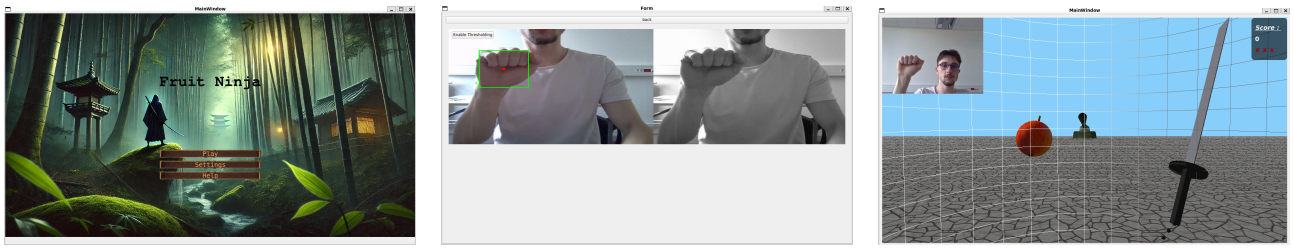


Figure 1: De gauche à droite : menu principal, fenêtre des paramètres, et fenêtre de jeu en cours d'exécution.

2 Conception

2.1 Architecture Générale

L'application repose sur le langage C++ et intègre trois technologies principales : Qt pour la gestion de l'interface utilisateur et des événements, OpenGL (avec pipeline fixe) pour le rendu 3D, et OpenCV pour la capture vidéo et la détection gestuelle.

2.1.1 Modèle-Vue-Contrôleur

Le schéma de développement suit une architecture proche du modèle MVC :

- **Modèle** : représenté par des classes comme Fruit et Cannon, qui encapsulent les données et la logique métier.
- **Vue** : assurée par le composant GameWidget qui gère le rendu 3D avec OpenGL, ainsi que par les éléments d'interface Qt (boutons, fenêtres, labels).
- **Contrôleur** : principalement la classe GameWidget, qui traite les entrées utilisateur, met à jour le modèle et déclenche le rendu. Les classes GameWindow et SettingsWindow agissent comme contrôleurs secondaires.

La communication entre les composants s'appuie sur le système de signaux et slots de Qt, assurant un couplage faible et une réactivité élevée.

2.2 Principales Classes

2.2.1 Fenêtres de l'Interface

MainWindow Cette classe hérite de QMainWindow et constitue la fenêtre principale de l'application. Elle affiche le menu initial, permettant de démarrer une partie, d'accéder aux paramètres via la SettingsWindow, ou de consulter le rapport.

GameWindow Également dérivée de QMainWindow, cette fenêtre héberge une session de jeu. Elle contient le GameWidget et affiche les informations contextuelles (score, vies). Ces données sont mises à jour dynamiquement grâce aux signaux émis par le widget de jeu.

SettingsWindow Cette classe, héritée de QWidget, permet de modifier les paramètres liés à la caméra. Elle contient un CameraWidget pour l'aperçu du flux vidéo, ainsi que des contrôles pour ajuster les options de traitement (ex. seuillage OTSU).

2.2.2 Composants de Jeu

GameWidget Composant central du jeu, cette classe dérive de QOpenGLWidget. Elle :

- Initialise le contexte graphique avec initializeGL();
- Gère le rendu de chaque trame via paintGL();
- Adapte la vue avec resizeGL().

Elle orchestre la création des fruits (createFruit()), la gestion de leur cycle de vie (m_fruit), le contrôle du Cannon, la détection des collisions (isFruitHit()), et l'interaction avec la détection gestuelle via CameraHandler. Des signaux Qt sont émis pour notifier les événements de jeu (scoreIncreased(), lifeDecrease()).

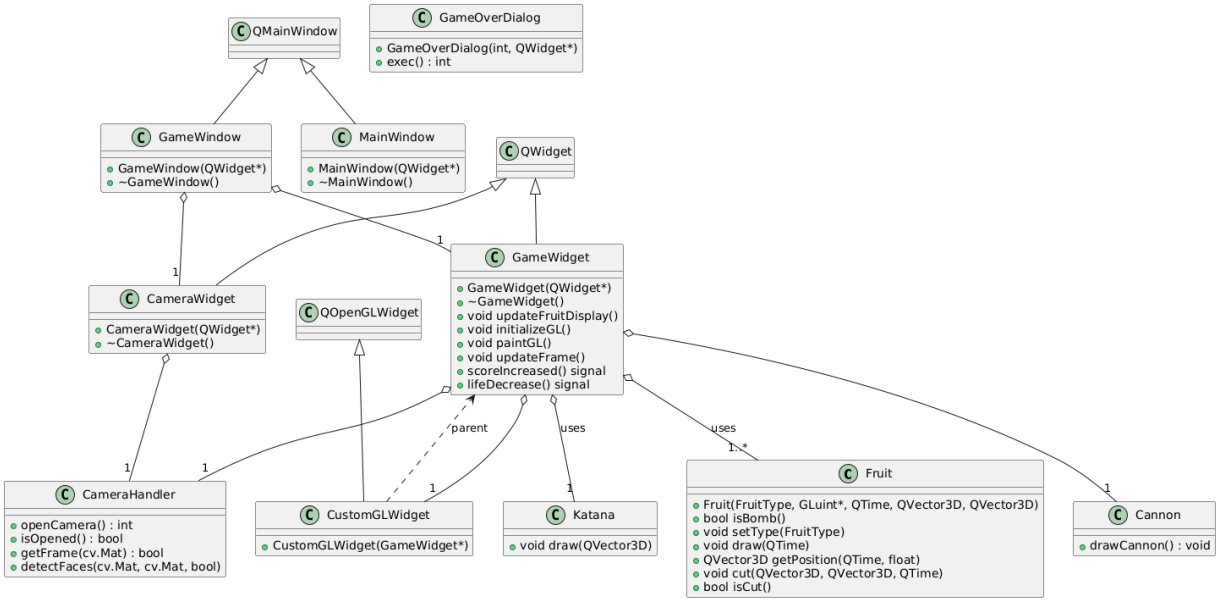
CameraHandler Cette classe est dédiée à la gestion du flux vidéo via OpenCV. Elle ouvre la webcam, capture les images, les convertit en niveaux de gris et applique un classificateur Haar (fist.xml) pour détecter la main du joueur (detectFaces()).

`Fruit` Cette classe représente un objet interactif du jeu (fruit ou bombe). Elle contient la position 3D, la vélocité, le type, la texture et l'état de l'objet. Elle sait se dessiner (`draw()`) et gérer sa découpe (`cut()`).

`Cannon` Le canon lance les objets dans la scène. Il gère sa position, son orientation (pouvant suivre la main détectée) et son affichage via `drawCannon()`.

`CameraWidget` Utilisé dans la `SettingsWindow`, ce widget Qt affiche le flux vidéo capturé, permettant de visualiser en temps réel les effets des paramètres modifiés.

2.3 Diagramme de classe



3 État de Finalisation de l'Application

3.1 Fonctions Validées et Opérationnelles

L'application est globalement fonctionnelle. L'interface permet une navigation fluide entre les écrans principaux, avec une gestion efficace des événements Qt. L'affichage en temps réel du score, des vies et le dialogue de fin de partie sont pleinement opérationnels. Le rendu 3D repose sur OpenGL fixe, avec une caméra perspective, un éclairage simple, des textures bien gérées et un système de secours en cas d'images manquantes.

La capture vidéo via webcam fonctionne, notamment sur macOS grâce à AVFoundation. La détection du poing est assurée par un classificateur Haar, avec possibilité d'utiliser un seuillage OTSU. Les permissions d'accès caméra sont gérées de manière robuste, accompagnées de messages explicites. La logique de jeu, avec trajectoires paraboliques, détection de collisions et mise à jour du score via les signaux Qt, est bien en place. Enfin, des effets sonores de base accompagnent les actions clés du jeu.

3.2 Fonctions Partiellement Finalisées ou à Améliorer

Certaines fonctions mériteraient d'être approfondies. La détection gestuelle est encore instable, notamment en conditions lumineuses variables, et pourrait être renforcée avec des solutions comme MediaPipe. Les performances sont acceptables (60 FPS) tant que peu d'objets sont présents, mais l'absence d'optimisations (comme le culling ou la mise en cache des textures caméra) limite l'efficacité. Côté audio, bien que les sons soient fonctionnels, leur volume est parfois irrégulier et l'absence d'audio 3D limite l'immersion.

3.3 Modules Non Implémentés

Plusieurs fonctionnalités prévues n'ont pu être intégrées. L'application ne propose pas de progression par niveaux, de bonus, de multijoueur ou de sauvegarde des scores. Sur le plan technique, elle n'exploite pas les shaders, le rendu instancié, les texture atlases ni le LOD. Aucune musique d'ambiance n'a été ajoutée.

3.4 Bogues Connus et Problèmes Subsistants

Quelques limitations persistent. La conversion BGR vers RGBA des images caméra est faite à chaque trame, ce qui affecte les performances. La projection 2D$\rightarrow$3D de la main détectée manque de précision en périphérie, et l'interface peut mal se redimensionner. La console reste trop verbeuse, avec des messages de débogage non filtrés. Des problèmes plus critiques, comme les permissions caméra ou les textures manquantes, ont toutefois été résolus.

3.5 Défis Spécifiques à l'Environnement macOS

Le développement sous macOS a soulevé des difficultés, notamment pour la gestion des permissions caméra, qui a nécessité des ajustements du fichier Info.plist et l'ajout de messages informatifs à l'utilisateur. L'usage d'AVFoundation a requis des vérifications de compatibilité, et les écrans Retina imposent une vigilance sur la mise à l'échelle, bien que cela n'ait pas été bloquant pour cette version

4 Fichiers d'entête

4.1 camerahandler.h

```
/**
 * @file camerahandler.h
 * @brief D claration de la classe CameraHandler.
 * @author Boutet Paul, El Gote Isma l
 */

#ifndef CAMERAHANDLER_H
#define CAMERAHANDLER_H

#include <opencv2/opencv.hpp>
#include <opencv2/objdetect.hpp>
#include <QString>
#include <vector>
#include <opencv2/core/types.hpp>

/**
 * @class CameraHandler
 * @brief G re l'acc s    la cam ra, la capture d'images et la d tection de
    visages.
 *
 * Cette classe encapsule les fonctionnalit s OpenCV pour ouvrir une cam ra,
 * r cup rer des images (frames) et effectuer une d tection de visages simple
 *    l'aide d'un classificateur en cascade.
 */
class CameraHandler
{
public:
    /**
     * @brief Constructeur de la classe CameraHandler.
     * Initialise les membres, notamment en tentant de charger le classificateur en
        cascade.
     */
    CameraHandler();

    /**
     * @brief Destructeur de la classe CameraHandler.
     * Lib re la capture vid o si elle est ouverte.
     */
    ~CameraHandler();

    /**
     * @brief Ouvre la cam ra par d faut.
     * @return int Code de statut : 1 en cas de succ s, 0 si aucune cam ra n'est
        trouv e, -1 en cas de probl me de permission ou autre erreur.
     */
    int openCamera();

    /**
     * @brief V rifie si la cam ra est ouverte et pr te    capturer.
     * @return true si la cam ra est ouverte, false sinon.
     */
    bool isOpened() const;

    /**
     * @brief R cup re une nouvelle image (frame) de la cam ra.
     * @param frame R f rence vers un objet cv::Mat qui recevra l'image captur e.
        (param tre de sortie)
     * @return true si une image a   t   r cup r e avec succ s, false sinon.
     */
    bool getFrame(cv::Mat& frame);
```

```
    /**
     * @brief D tecte une main dans une image donn e (on remarque que les variables
         s'appellent
     * face    chaque fois car nous avons reprise le fichier haarcascade que nous
        avions fait
     * au premiers TDs et pas renomm  les variables).
     * @param frame Image couleur d'entr e sur laquelle effectuer la d tection.
        Peut  tre  modifi e pour dessiner les d tections.
     * @param grayFrame Image en niveaux de gris (utilis e pour la d tection). Si
        vide, elle sera g  n  r  e    partir de 'frame'.
     * @param thresholdingEnabled Si true, un seuillage peut  tre  appliqu  (non
        impl ment  dans la signature actuelle, mais sugg r  par le nom).
     * @return std::vector<cv::Point> Liste des centres des visages d tect s.
     * @note L'algorithme utilise un classificateur en cascade Haar pour la
        d tection de visages.
     */
    std::vector<cv::Point> detectFaces(cv::Mat& frame, cv::Mat& grayFrame, bool
        thresholdingEnabled);



private:
    cv::VideoCapture cap; ///< Objet VideoCapture d'OpenCV pour g rer le flux de la
        cam ra.
    cv::CascadeClassifier faceCascade; ///< Classificateur en cascade OpenCV pour la
        d tection de visages.

    /**
     * @brief Charge le fichier XML du classificateur en cascade pour la d tection
        de main.
     * @return true si le chargement est r ussi, false sinon.
     * @note Le chemin vers le fichier cascade est g n ralement cod  en dur ou
        configurable.
     */
    bool loadFaceCascade();
};

#endif // CAMERAHANDLER_H
```

4.2 camerawidget.h

```
#ifndef CAMERAWIDGET_H
#define CAMERAWIDGET_H

#include <QWidget>
#include <QTimer>
#include <QImage>
#include <QPixmap>
#include "camerahandler.h"

namespace Ui {
class CameraWidget;
}

class CameraWidget : public QWidget
{
    Q_OBJECT

public:
    explicit CameraWidget(QWidget *parent = nullptr);
    ~CameraWidget();

private slots:
    void updateFrame();
```

```
    void on_thresholdingButton_clicked();

private:
    Ui::CameraWidget *ui;
    CameraHandler cameraHandler;
    QTimer *timer;

    void showPlaceholderMessage(const QString &title, const QString &message);
    bool thresholdingEnabled = false; // Flag to enable/disable thresholding
};

#endif // CAMERAWIDGET_H
```

4.3 cannon.h

```
/**
 * @file cannon.h
 * @brief D claration de la classe Cannon.
 * @author Boutet Paul, El Gote Isma l
 */

#include <qvectornd.h>
#include <QVector3D>
#include <qopengl.h>

#ifdef __APPLE__
#include <OpenGL/glu.h>
#else
#include <GL/glu.h>
#endif

/**
 * @class Cannon
 * @brief Repr sente le canon utilis  par le joueur pour lancer des fruits.
 *
 * Cette classe g re la position, l'orientation et le dessin du canon.
 * Elle peut  galement  r agir  la cr ation de fruits pour s'orienter.
 */
class Cannon {
private :
    QVector3D position;     ///< Position 3D du canon dans la sc ne.
    float angleX;           ///< Angle de rotation du canon autour de l'axe X.
    float angleY;           ///< Angle de rotation du canon autour de l'axe Y.
    float angleZ;           ///< Angle de rotation du canon autour de l'axe Z.
    GLUquadric* quadric;    ///< Objet quadrique GLU utilis  pour dessiner les
        parties du canon.
    GLuint cannonTexture;   ///< Identifiant de la texture OpenGL pour le canon.
    bool hasTexture;        ///< Indicateur bool en : true si une texture est
        assign e au canon, false sinon.


public:
    /**
     * @brief Constructeur de la classe Cannon.
     * Initialise les angles, la position par d faut et l'objet quadrique.
     */
    Cannon();

    /**
     * @brief Destructeur de la classe Cannon.
     * Lib re les ressources allou es (par exemple, l'objet quadrique GLU).
     */
    ~Cannon();
```

```
    /**
     * @brief D finit la position du canon.
     * @param position Nouvelle position 3D du canon.
     */
    void setPosition(QVector3D position) { this->position = position; }

    /**
     * @brief Oriente le canon pour qu'il pointe vers une direction donn e.
     * @param direction Vecteur 3D indiquant la direction cible.
     * @note Calcule les angles de rotation (angleX, angleY) n cessaires.
     */
    void setDirection(QVector3D direction);

    /**
     * @brief Assigne une texture au canon.
     * @param textureId Identifiant de la texture OpenGL    utiliser.
     */
    void setTexture(GLuint textureId) { cannonTexture = textureId; hasTexture = true;
        }

    /**
     * @brief Dessine le canon dans la sc ne OpenGL.
     * Utilise la position et les angles actuels pour transformer et dessiner le
        mod le du canon.
     */
    void drawCannon();

    /**
     * @brief M thode  appel e lors de la cr ation d'un fruit.
     * Permet au canon de s'orienter vers la direction de lancement du fruit.
     * @param direction Direction initiale du fruit lanc .
     */
    void onFruitCreated(QVector3D direction);

};
```

4.4 fruit.h

```
/**
 * @file fruit.h
 * @brief D claration de la classe Fruit.
 * @author Boutet Paul, El Gote Isma l
 */

#ifndef FRUIT_H
#define FRUIT_H

#include <qopengl.h>
#include <QColor>
#include <QVector3D>
#include <QTime>
#include <QVector4D> // Added for QVector4D
#ifdef __APPLE__
#include <OpenGL/glu.h>
#else
#include <GL/glu.h>
#endif

/**
 * @class Fruit
 * @brief Repr sente un fruit ou une bombe dans le jeu.
 *
 * G re la physique (position, mouvement), l'apparence (type, texture),
 * l' tat  (coup  ou non) et le dessin de l'objet fruit/bombe.
```

```
 */
class Fruit {
public:
    /**
     * @enum FruitType
     * @brief    numre    les diff rents types de fruits et la bombe.
     */
    enum FruitType {
        APPLE,
        STRAWBERRY,
        BANANA,
        PEAR,
        BOMB
    };

    /**
     * @brief Constructeur principal de la classe Fruit.
     * @param type Type de fruit (FruitType).
     * @param textures Pointeur vers un tableau d'identifiants de textures OpenGL.
     * @param currentTime Temps actuel, utilis  pour initialiser le temps de d part
         du fruit.
     * @param initSpeed Vitesse initiale du fruit.
     * @param initPosition Position initiale du fruit.
     */
    Fruit(FruitType type, GLuint* textures, QTime currentTime, QVector3D initSpeed,
        QVector3D initPosition);

    /**
     * @brief Constructeur de la classe Fruit avec position et vitesse al atoires.
     * @param type Type de fruit (FruitType).
     * @param textures Pointeur vers un tableau d'identifiants de textures OpenGL.
     * @param currentTime Temps actuel, utilis  pour initialiser le temps de d part
         du fruit.
     */
    Fruit(FruitType type, GLuint* textures, QTime currentTime);

    /**
     * @brief Constructeur de la classe Fruit avec type, position et vitesse
         al atoires.
     * @param textures Pointeur vers un tableau d'identifiants de textures OpenGL.
     * @param currentTime Temps actuel, utilis  pour initialiser le temps de d part
         du fruit.
     */
    Fruit(GLuint* textures, QTime currentTime);

    /**
     * @brief Destructeur de la classe Fruit.
     * Lib re les ressources allou es (par exemple, l'objet quadrique GLU).
     */
    ~Fruit();

    /**
     * @brief V rifie si le fruit est une bombe.
     * @return true si le type actuel est BOMB, false sinon.
     */
    bool isBomb();

    /**
     * @brief D finit le type du fruit.
     * @param type Nouveau type de fruit (FruitType).
     */
    void setType(FruitType type);

    /**
     * @brief Dessine le fruit    sa position actuelle.
```

```
     * @param currentTime Temps actuel, utilis  pour calculer la position et g rer
         les animations.
     * @note Appelle la m thode de dessin sp cifique au type de fruit.
     */
    void draw(QTime currentTime);

    /**
     * @brief Calcule et retourne la position du fruit    un temps donn .
     * @param currentTime Temps actuel pour lequel calculer la position.
     * @param firstPart Facteur optionnel pour le calcul de la position (utilis
         pour les fruits coup s).
     * @return QVector3D repr sentant la position du fruit.
     * @note La position est calcul e en utilisant une trajectoire parabolique
         bas e sur la vitesse initiale et la gravit .
     */
    QVector3D getPosition(QTime currentTime, float firstPart = 1.f);

    /**
     * @brief Retourne la direction (vitesse) initiale du fruit.
     * @return QVector3D repr sentant la vitesse initiale.
     */
    QVector3D getInitialDirection() { return initalSpeed; }

    /**
     * @brief Marque le fruit comme coup  et d finit le plan de coupe.
     * @param cutOriginPoint Point d'origine sur le plan de coupe.
     * @param cutNormalVector Vecteur normal au plan de coupe.
     * @param currentTime Temps actuel auquel le fruit est coup .
     */
    void cut(const QVector3D& cutOriginPoint, const QVector3D& cutNormalVector, QTime
        currentTime);

    /**
     * @brief V rifie si le fruit a   t   coup .
     * @return true si le fruit est coup , false sinon.
     */
    bool isCut() const;

private :
    FruitType currentFruit; ///< Type actuel du fruit (pomme, bombe, etc.).

    /**
     * @brief Dessine une pomme.
     * @param currentTime Temps actuel pour le calcul de la position.
     * @param firstPart Facteur pour dessiner une partie du fruit (utilis  si coup
         ).
     */
    void drawApple(QTime currentTime, float firstPart = 1.f);

    /**
     * @brief Dessine une fraise.
     * @param currentTime Temps actuel pour le calcul de la position.
     * @param firstPart Facteur pour dessiner une partie du fruit (utilis  si coup
         ).
     */
    void drawStrawberry(QTime currentTime, float firstPart = 1.f);

    /**
     * @brief Dessine une banane.
     * @param currentTime Temps actuel pour le calcul de la position.
     * @param firstPart Facteur pour dessiner une partie du fruit (utilis  si coup
         ).
     */
    void drawBanana(QTime currentTime, float firstPart = 1.f);

    /**
```

```
     * @brief Dessine une poire.
     * @param currentTime Temps actuel pour le calcul de la position.
     * @param firstPart Facteur pour dessiner une partie du fruit (utilis  si coup
         ).
     */
    void drawPear(QTime currentTime, float firstPart = 1.f);

    /**
     * @brief Dessine une bombe.
     * @param currentTime Temps actuel pour le calcul de la position.
     * @param firstPart Facteur pour dessiner une partie de la bombe (utilis  si
         coup e).
     */
    void drawBomb(QTime currentTime, float firstPart = 1.f);

    /**
     * @brief Applique une texture    l'objet.
     * @param textureID Identifiant de la texture OpenGL    appliquer.
     */
    void setTexture(GLuint textureID);

    /**
     * @brief D finit les propri t s mat rielles pour l' clairage  OpenGL.
     * @param ambient Composante ambiante du mat riau.
     * @param diffuse Composante diffuse du mat riau.
     * @param specular Composante sp culaire du mat riau.
     * @param shininess Exposant de brillance sp culaire.
     */
    void setMaterial(const GLfloat* ambient, const GLfloat* diffuse, const GLfloat*
        specular, const  GLfloat* shininess);

    /**
     * @brief S lectionne un type de fruit al atoire (excluant la bombe).
     * @return FruitType al atoire.
     */
    FruitType getRandomFruitType();

    /**
     * @brief G n re une vitesse initiale al atoire pour le fruit.
     * @return QVector3D repr sentant la vitesse initiale al atoire.
     */
    QVector3D getRandomInitSpeed();

    QVector3D initalSpeed;      ///< Vitesse initiale du fruit lors de son lancement.
    QVector3D initialPosition;  ///< Position initiale du fruit lors de son lancement
        .
    GLUquadric* quadric;        ///< Objet quadrique GLU utilis  pour dessiner des
        formes (sph res, cylindres).
    GLuint* textures;           ///< Pointeur vers le tableau global de textures
        OpenGL.
    QTime startTime;            ///< Temps auquel le fruit a  t  cr  ou lanc .

    bool m_isCut;               ///< Indicateur bool en : true si le fruit a  t
        coup , false sinon.
    QVector4D m_clipPlaneEquation; ///<  quation  du plan de coupe (Ax + By + Cz + D
        = 0) sous forme de QVector4D (A, B, C, D).
    QVector3D normal;           ///< Vecteur normal au plan de coupe (redondant avec
        m_clipPlaneEquation.toVector3D() ?).
    QTime cutTime;              ///< Temps auquel le fruit a  t  coup .

};



#endif // FRUIT_H
```

4.5 gameoverdialog.h

```
/**
 * @file gameoverdialog.h
 * @brief D claration de la classe GameOverDialog.
 * @author Boutet Paul, El Gote Isma l
 */

#ifndef GAMEOVERDIALOG_H
#define GAMEOVERDIALOG_H

#include <QDialog>

class QLabel;
class QPushButton;

/**
 * @class GameOverDialog
 * @brief Bo te de dialogue affich e lorsque la partie est termin e.
 *
 * Cette classe affiche le score final du joueur et propose des options pour
 * commencer une nouvelle partie ou quitter l'application.
 */
class GameOverDialog : public QDialog
{
    Q_OBJECT

public:
    /**
     * @brief Constructeur de GameOverDialog.
     * @param score Score final du joueur    afficher.
     * @param parent Widget parent, nullptr par d faut.
     * Initialise l'interface utilisateur de la bo te de dialogue avec le score.
     */
    explicit GameOverDialog(int score, QWidget *parent = nullptr);

    /**
     * @brief Destructeur de GameOverDialog.
     * Lib re les ressources allou es.
     */
    ~GameOverDialog();

private slots:
    /**
     * @brief Slot appel  lorsque le bouton "Nouvelle Partie" est cliqu .
     * G re la logique pour red marrer une partie en relan ant une fen tre de jeu
        .
     */
    void onNewGameClicked();

    /**
     * @brief Slot appel  lorsque le bouton "Quitter" est cliqu .
     * Ferme l'application ou la bo te de dialogue.
     * Rejette la bo te de dialogue.
     */
    void onExitClicked();

private:
    QLabel *gameOverLabel;  ///<  tiquette  affichant le message "Game Over".
    QLabel *scoreLabel;     ///<  tiquette  affichant le score final du joueur.
    QPushButton *newGameButton; ///< Bouton pour d marrer une nouvelle partie.
    QPushButton *exitButton;    ///< Bouton pour quitter le jeu/l'application.
};

#endif // GAMEOVERDIALOG_H
```

4.6 gamewidget.h

```
/**
 * @file gamewidget.h
 * @brief D claration de la classe GameWidget.
 * @author Boutet Paul, El Gote Isma l
 */

#ifndef GAMEWIDGET_H
#define GAMEWIDGET_H

#include <QWidget>
#include <QOpenGLWidget>
#include "fruit.h"
#include <qlabel.h>
#include <vector>
#include <QColor>
#include <QCoreApplication>
#include <QDir>
#include <QDebug>
#include <QTimer>
#include <QSoundEffect>
#include "camerahandler.h"
#include "cannon.h"
#include <QKeyEvent>
#include <QTime>
#include <QVector3D>
#include <opencv2/opencv.hpp>

typedef struct GLUquadric GLUquadric;

namespace Ui
{
    class GameWidget;
}

/**
 * @class GameWidget
 * @brief G re la logique principale du jeu, l'affichage 3D et l'interaction avec la
      cam ra.
 *
 * Cette classe est responsable de l'initialisation de l'environnement OpenGL,
 * du rendu des objets du jeu (fruits, canon), de la gestion des entr es utilisateur
 * (via la cam ra), et de la mise    jour de l' tat  du jeu.
 */
class GameWidget : public QWidget
{
    Q_OBJECT

public:
    /**
     * @brief Constructeur de GameWidget.
     * @param parent Widget parent, nullptr par d faut.
     */
    explicit GameWidget(QWidget *parent = nullptr);

    /**
     * @brief Destructeur de GameWidget.
     * Lib re les ressources allou es.
     */
    ~GameWidget();

    /**
     * @brief Met    jour l'affichage des fruits.
     * Force l'actualisation de l'affichage des fruits    l' cran .
     */
```

```
    void updateFruitDisplay();

signals:
    /**
     * @brief Signal  mis  lorsqu'un fruit est touch .
     * Connect  pour augmenter le score du joueur.
     */
    void scoreIncreased();

    /**
     * @brief Signal  mis  lorsqu'une bombe est touch e.
     * Connect  pour diminuer la vie du joueur.
     */
    void lifeDecrease();

public slots:
    /**
     * @brief Initialise l'environnement OpenGL.
     * Appel  une fois avant le premier appel    paintGL() ou resizeGL().
     * Configure l' tat  initial d'OpenGL (couleur de fond, profondeur, etc.).
     */
    void initializeGL();

    /**
     * @brief Initialise les textures utilis es dans le jeu.
     * Charge les images pour les fruits, bombes, etc., et les pr pare pour OpenGL.
     */
    void initializeTextures();

    /**
     * @brief D marre un compte    rebours avant le d but du jeu pour donner le
     * temps au joueur de se pr parer et    OpenGL de s'initialiser.
     * @param seconds Dur e du compte    rebours en secondes.
     */
    void startCountdown(int seconds);

    /**
     * @brief G re le redimensionnement de la fen tre OpenGL.
     * @param width Nouvelle largeur de la fen tre.
     * @param height Nouvelle hauteur de la fen tre.
     * Met    jour la projection et le viewport OpenGL.
     */
    void resizeGL(int width, int height);

    /**
     * @brief G re le rendu de la sc ne OpenGL.
     * Appel  chaque fois que le widget a besoin d' tre  redessin .
     * Dessine tous les    lments    du jeu (fruits, canon, fond, etc.).
     */
    void paintGL();

    /**
     * @brief Met    jour la frame de la cam ra et traite les interactions.
     * Ce slot est probablement connect     un QTimer pour r cup rer
        p riodiquement
     * une nouvelle image de la cam ra et la traiter.
     */
    void updateFrame();

private:
    Ui::GameWidget *ui;
    std::vector<Fruit *> m_fruit; ///< Conteneur pour tous les objets Fruit actifs
       dans le jeu.
    GLuint *textures; ///< Tableau d'identifiants de texture OpenGL.
    QFont m_font; ///< Police de caract res utilis e pour afficher du texte (ex:
       score, messages).
```

```
    QSoundEffect *m_sliceSound; ///< Effet sonore jou  lorsqu'un fruit est coup .
    QSoundEffect *m_shootSound; ///< Effet sonore jou  lors d'un tir.
    QLabel *label; ///< QLabel utilis  pour afficher le score et les vies.
    GLUquadric *cylinder; ///< Objet quadrique GLU.
    CameraHandler *cameraHandler; ///< Gestionnaire pour l'interaction avec la webcam
        .
    QTimer *cameraTimer; ///< Timer pour d clencher la mise  jour p riodique de
        la frame de la cam ra.
    cv::Mat currentFrame; ///< Image actuelle captur e par la cam ra (en couleur).
    cv::Mat grayFrame; ///< Image actuelle captur e par la cam ra (convertie en
        niveaux de gris).
    bool cameraInitialized; ///< Indicateur de l' tat  d'initialisation de la cam ra
        .
    QVector3D projectedPoint; ///< Coordonn es 3D d'un point projet  (
        potentiellement depuis l'espace cam ra vers l'espace jeu).
    bool hasProjectedPoint; ///< Indicateur de la disponibilit  d'un point projet .
    Cannon cannon; ///< Objet repr sentant le canon du joueur.
    bool displayCamera; ///< Indicateur pour afficher ou non le flux de la cam ra
        l' cran .
    GLuint m_cameraTextureId; ///< Identifiant de texture OpenGL pour le flux vid o
        de la cam ra.

    /**
     * @brief G re les  vnements  de pression de touche.
     * @param event  vnement  de clavier contenant les informations sur la touche
         press e.
     * Utilis  pour les contr les au clavier (d bogage, actions alternatives).
     */
    void keyPressEvent(QKeyEvent *event) override;

    /**
     * @brief Cr e une texture OpenGL  partir d'une couleur unie.
     * @param color Couleur de la texture  cr er.
     * @return QImage repr sentant la texture color e.
     * Utilis  comme fallback si le chargement d'une texture image  choue .
     */
    QImage createColorTexture(const QColor &color);

    /**
     * @brief Cr e et initialise un nouvel objet Fruit.
     * @return Pointeur vers le Fruit nouvellement cr .
     * Configure la position initiale, la vitesse, le type (fruit/bombe) du fruit.
     */
    Fruit *createFruit();

    /**
     * @brief Initialise la cam ra.
     * Configure la capture vid o et pr pare la cam ra pour son utilisation.
     */
    void initializeCamera();

    /**
     * @brief V rifie si un point donn  touche un fruit.
     * @param point Coordonn es du point d'interaction.
     * @param fruit Pointeur vers l'objet Fruit  tester.
     * @param currentTime Temps actuel, utilis  pour g rer les timings de collision
         ou d'animation.
     * @return true si le fruit est touch , false sinon.
     * @note L'algorithme peut impliquer de v rifier si 'point' est dans le rayon du
         'fruit'.
     */
    bool isFruitHit(const cv::Point &point, Fruit *fruit, QTime currentTime);

    /**
     * @brief Convertit un point de l'espace cam ra 2D en coordonn es de l'espace
        de jeu 3D
```

```
     * en le mappant sur le cylindre entourant le joueur.
     * @param cameraPoint Coordonn es 2D du point dans l'image de la cam ra.
     * @param gameX R f rence pour stocker la coordonn e X r sultante dans l'
         espace de jeu. (param tre de sortie)
     * @param gameZ R f rence pour stocker la coordonn e Z r sultante dans l'
         espace de jeu. (param tre de sortie)
     */
    void convertCameraPointToGameSpace(const cv::Point &cameraPoint, float &gameX,
        float &gameZ);
};

#endif // GAMEWIDGET_H
```

4.7 gamewindow.h

```
    /**
 * @file gamewindow.h
 * @brief D claration de la classe GameWindow.
 * @author Boutet Paul, El Gote Isma l
 */

#ifndef GAMEWINDOW_H
#define GAMEWINDOW_H

#include <QMainWindow>
#include "gamewidget.h"
namespace Ui {
class GameWindow;
}

/**
 * @class GameWindow
 * @brief Fen tre principale de l'application de jeu.
 *
 * Cette classe h rite de QMainWindow et sert de conteneur principal pour les
     widgets du jeu,
 * tels que GameWidget. Elle g re   galement   l'affichage du score et des vies.
 */
class GameWindow : public QMainWindow
{
    Q_OBJECT

public:
    /**
     * @brief Constructeur de GameWindow.
     * @param parent Widget parent, nullptr par d faut.
     * Initialise l'interface utilisateur et configure les connexions de signaux/
         slots.
     */
    explicit GameWindow(QWidget *parent = nullptr);

    /**
     * @brief Destructeur de GameWindow.
     * Lib re les ressources allou es, notamment l'objet ui.
     */
    ~GameWindow();


private:
    int lives = 3;          ///< Nombre de vies restantes pour le joueur. Initialis
            3.
    int score = 0;          ///< Score actuel du joueur. Initialis     0.
    GameWidget* gameWidget; ///< Pointeur vers le widget principal du jeu (o   se
        d roule l'action 3D).
```

```
    Ui::GameWindow *ui; ///< Pointeur vers l'objet d'interface utilisateur g n r
       par Qt Designer.

    /**
     * @brief Met    jour l'affichage des   tiquettes   (score et vies) dans l'
        interface utilisateur.
     * Cette m thode est appel e lorsque le score ou le nombre de vies change (par
        des signaux).
     */
    void updateLabelDisplay();
};

#endif // GAMEWINDOW_H
```

4.8 katana.h

```
    /**
 * @file katana.h
 * @brief D claration de la classe Katana.
 * @author Boutet Paul, El Gote Isma l
 */

#ifndef KATANA_H
#define KATANA_H

#ifdef __APPLE__
#include <OpenGL/glu.h>
#else
#include <GL/glu.h>
#endif

#include <QVector3D>
#include <vector>

/**
 * @class Katana
 * @brief Repr sente un katana dans le jeu, utilis  pour interagir avec les objets.
 *
 * Cette classe g re le dessin du katana en 3D.
 */
class Katana {
public:


    std::vector<GLuint> textures; ///< Tableau de textures pour le katana.

    /**
     * @brief Constructeur de Katana.
     * Initialise les ressources n cessaires pour le katana, comme l'objet quadrique
        .
     */
    Katana();

    /**
     * @brief Destructeur de Katana.
     * Lib re les ressources allou es, notamment l'objet quadrique.
     */
    ~Katana();

    /**
     * @brief Dessine le katana    la position sp cifi e.
     * @param position Position 3D o   le katana doit   tre   dessin .
     */
    void draw(const QVector3D& position);
```

```
    /**
     * @brief D finit les textures du katana.
     * @param textures Tableau de textures    appliquer au katana.
     */
    void setTextures(GLuint blade, GLuint handle, GLuint chain) { textures[0] = blade
        ; textures[1] = handle; textures[2] = chain; }

private:
    /**
     * @brief Dessine la lame du katana.
     */
    void drawBlade();

    /**
     * @brief Dessine le manche du katana.
     */
    void drawHandle();

    /**
     * @brief Dessine la cha ne ou un    lment    d coratif du katana.
     */
    void drawChain();
    GLUquadric* quadric; ///< Objet quadrique GLU utilis  pour dessiner les parties
        cylindriques du katana.
};

#endif // KATANA_H
```

4.9 mainwindow.h

```
    /**
 * @file mainwindow.h
 * @brief D claration de la classe MainWindow.
 * @author Boutet Paul, El Gote Isma l
 */
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>

QT_BEGIN_NAMESPACE
namespace Ui {
class MainWindow;
}
QT_END_NAMESPACE

/**
 * @class MainWindow
 * @brief Fen tre principale de l'application, servant de menu d'accueil.
 *
 * Cette classe est la premi re fen tre affich e    l'utilisateur. Elle permet
    typiquement
 * de lancer une nouvelle partie, d'acc der aux param tres, ou de quitter l'
    application.
 */
class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    /**
     * @brief Constructeur de MainWindow.
     * @param parent Widget parent, nullptr par d faut.
```

```
     * Initialise l'interface utilisateur de la fen tre principale.
     */
    MainWindow(QWidget *parent = nullptr);

    /**
     * @brief Destructeur de MainWindow.
     * Lib re les ressources allou es, notamment l'objet ui.
     */
    ~MainWindow();

private slots:
    /**
     * @brief Slot appel  lors du clic sur le deuxi me bouton poussoir (
        pushButton_2).
     * Permet d'ouvrir une fen tre de jeu.
     */
    void on_pushButton_2_clicked();

    /**
     * @brief Slot appel  lors du clic sur le premier bouton poussoir (pushButton).
     * Permet d'ouvrir une fen tre de param tres.
     */
    void on_pushButton_clicked();

private:
    Ui::MainWindow *ui; ///< Pointeur vers l'objet d'interface utilisateur g n r
       par Qt Designer.
};
#endif // MAINWINDOW_H
```

4.10 settingswindow.h

```
/**
 * @file settingswindow.h
 * @brief D claration de la classe SettingsWindow.
 * @author Boutet Paul, El Gote Isma l
 */

#ifndef SETTINGSWINDOW_H
#define SETTINGSWINDOW_H

#include <QWidget>
#include "camerawidget.h"

namespace Ui {
class SettingsWindow;
}

/**
 * @class SettingsWindow
 * @brief Fen tre de param tres pour configurer les options de l'application.
 *
 * Cette classe h rite de QWidget et permet    l'utilisateur de configurer
 * certains param tres, notamment la cam ra. Elle peut   galement   afficher
 * un aper u du flux vid o de la cam ra.
 */
class SettingsWindow : public QWidget
{
    Q_OBJECT

public:
    /**
     * @brief Constructeur de SettingsWindow.
     * @param parent Widget parent, nullptr par d faut.
```

```
     * Initialise l'interface utilisateur de la fen tre des param tres et
        potentiellement le cameraWidget.
     */
    explicit SettingsWindow(QWidget *parent = nullptr);

    /**
     * @brief Destructeur de SettingsWindow.
     * Lib re les ressources allou es, notamment l'objet ui et cameraWidget.
     */
    ~SettingsWindow();

private slots:
    /**
     * @brief Slot appel  lors du clic sur le bouton poussoir (pushButton).
     * Permet d'activer / d activer sur le seuillage.
     */
    void on_pushButton_clicked();

private:
    Ui::SettingsWindow *ui;     ///< Pointeur vers l'objet d'interface utilisateur
        g n r  par Qt Designer.
    CameraWidget *cameraWidget; ///< Pointeur vers un widget qui affiche le flux de
       la cam ra ou permet de configurer ses param tres.
};

#endif // SETTINGSWINDOW_H
```