

Rapport de Projet

Jeu de Défense - Détection de Mains et Interaction 3D

Yazid El Mahi

Younès Tadili

23 mai 2025



Ce rapport présente le travail réalisé dans le cadre du projet de défense concernant un jeu interactif de découpe de projectiles par détection de la main via webcam.

Introduction

Ce projet a pour objectif de développer un jeu de défense en 3D permettant à l'utilisateur d'interagir naturellement par le mouvement de sa main capté par une webcam. L'application propose une expérience immersive avec une interface graphique intégrée, combinant des techniques de vision par ordinateur, d'OpenGL 3D et d'interactions temps réel.

Ce rapport présente dans un premier temps les spécifications de l'interface utilisateur et décrit les interactions principales. La conception objet détaillera ensuite les classes majeures ainsi que leurs relations. Enfin, l'état actuel de l'application sera exposé, accompagné d'une analyse des fonctionnalités validées et des axes d'amélioration.

Spécifications de l'interface utilisateur

L'interface utilisateur du jeu de défense se compose principalement de deux fenêtres : un menu principal et la fenêtre de jeu interactive.

Style graphique et thème

Le choix du style graphique s'inspire directement de l'univers Minecraft, afin de créer une ambiance immersive et familière pour l'utilisateur. Ce choix se traduit par :

- L'utilisation d'une police Minecraft personnalisée pour le titre et les textes importants,
- Un fond graphique thématique reprenant les textures caractéristiques du jeu (blocs, ciel, portail du Nether),
- Des textures pixelisées sur les projectiles et éléments de décor pour renforcer l'esthétique rétro et cubique,
- Une palette de couleurs inspirée de l'univers Minecraft.

Cette cohérence visuelle vise à plonger l'utilisateur dans un univers ludique tout en facilitant l'identification des éléments interactifs.

Menu principal

Le menu principal accueille l'utilisateur avec un titre stylisé, une description succincte du jeu, et deux boutons d'action : **Démarrer le jeu** et **Quitter**.

- **Démarrer le jeu** : lance la scène 3D en plein écran et initialise le jeu.
- **Quitter** : ferme l'application.

Le menu est visuellement habillé avec une police Minecraft personnalisée et un fond graphique, offrant une ambiance cohérente avec le thème du jeu.

Fenêtre de jeu

La fenêtre de jeu affiche une scène 3D OpenGL contenant l'environnement, les projectiles, la main virtuelle, ainsi que les effets visuels (fragments, portail du Nether, etc.).

Affichage Les éléments sont rendus en 3D avec textures et animations. Une interface utilisateur superposée affiche en temps réel :

- Le score actuel,
- Le nombre de points de vie (PV),
- Le temps écoulé de la partie.

Interactions

- **Détection de la main** : la position 2D de la main captée par la webcam est transformée en coordonnées 3D, permettant de positionner une main virtuelle dans la scène.
- **Découpe des projectiles** : l'utilisateur interagit en déplaçant la main virtuelle pour couper les projectiles volants. Les collisions entre la main virtuelle et les projectiles sont détectées pour déclencher la fragmentation et l'attribution de points.
- **Contrôle du jeu** :
 - Démarrage et arrêt du jeu via boutons externes (menu ou fenêtre principale),
 - Mise en pause et reprise du jeu,
 - Affichage d'une boîte de dialogue *Game Over* avec score et temps, offrant les options **Réessayer** ou **Quitter**.

Feedback visuel La main détectée est mise en surbrillance via un cercle vert dans la fenêtre de capture vidéo. Les projectiles sont affichés avec leurs textures respectives et présentent une animation de rotation. Lors de la découpe, des fragments apparaissent pour renforcer l'effet visuel.

L'interface est conçue pour être intuitive et immersive, limitant la nécessité d'éléments de contrôle complexes et privilégiant une interaction naturelle basée sur les gestes de l'utilisateur.

Conception

Le projet s'appuie sur plusieurs classes principales, organisées pour gérer la logique du jeu, l'affichage graphique, ainsi que les interactions utilisateur.

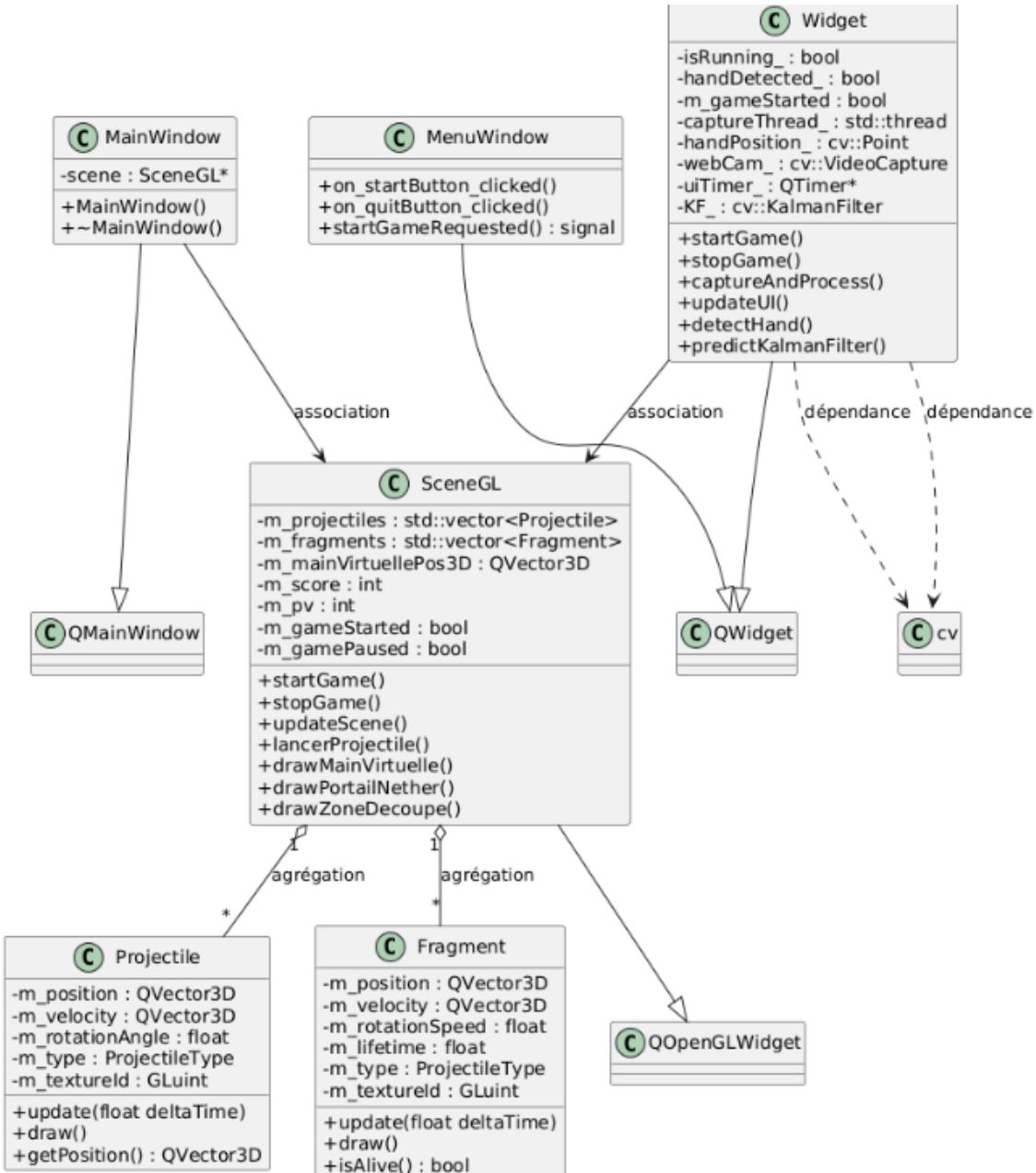
Principales classes

- **Widget** : classe principale de la fenêtre de jeu. Elle gère l'interface utilisateur, la capture vidéo via la webcam, la détection et le suivi de la main, ainsi que la synchronisation avec la scène 3D. Elle coordonne le démarrage, l'arrêt et la gestion du jeu.
- **SceneGL** : classe dérivée de `QOpenGLWidget` qui gère le rendu 3D. Elle contient la logique d'affichage des éléments graphiques (environnement, projectiles, fragments, main virtuelle, portail), la gestion des textures, et la mise à jour de la scène à chaque frame.
- **Projectile** : représente un projectile volant dans la scène (boule de feu, flèche, cube TNT, etc.). Cette classe gère la position, la vitesse, la rotation, la mise à jour physique (gravité, déplacement) et le dessin de chaque projectile avec sa texture spécifique.
- **Fragment** : représente un fragment issu de la découpe d'un projectile. Chaque fragment possède sa propre position, vitesse, rotation et durée de vie, et est rendu en 3D avec une taille réduite.
- **MenuWindow** : fenêtre principale d'accueil, proposant au joueur de démarrer le jeu ou de quitter. Cette classe applique un style graphique personnalisé (police Minecraft, fonds d'écran) et émet un signal pour démarrer la partie.
- **MainWindow** : classe conteneur simplifiée qui intègre la scène OpenGL dans une fenêtre principale. Elle est utilisée pour afficher la scène 3D dans une application Qt standard.

Relations entre les classes

- **Widget** contrôle la logique de capture et de détection de la main, et communique les positions à **SceneGL** pour mise à jour graphique.
- **SceneGL** contient et gère les objets **Projectile** et **Fragment**, en orchestrant leur mise à jour et rendu.
- **MenuWindow** est la porte d'entrée du programme et, par un signal, déclenche le démarrage du jeu en affichant la **Widget**.
- **MainWindow** intègre **SceneGL** comme widget central pour le rendu 3D dans un contexte Qt standard.

Diagramme des classes



État de finalisation de l'application

Fonctions validées

- **Détection et suivi de la main** via webcam, avec filtrage Kalman pour une position stable.
- **Affichage 3D dynamique** de la scène, incluant l'environnement (sol, murs, skybox, portail Nether).
- **Gestion et rendu des projectiles** avec plusieurs types (sphère, cube, cône, cylindre) et textures spécifiques.
- **Mécanique de jeu** : lancement automatique des projectiles à intervalles réguliers.
- **Gestion des collisions** entre projectiles et position de la main virtuelle, avec génération

- de fragments animés.
- **Interface utilisateur** : menu d'accueil stylisé, affichage des scores, points de vie, temps de jeu.
- >Gestion des états du jeu : démarrage, pause, reprise, fin de partie.

Fonctions en cours de finalisation

- >Amélioration de la précision et rapidité de la détection de la main dans différents environnements lumineux.
- >Ajout de sons et effets sonores synchronisés avec les actions du jeu (à venir).

Bugs connus et limitations

- >Fragilité de la détection de la main sur fonds complexes ou en faible luminosité.
- >Limitation du contrôle par la main virtuelle : certaines collisions ne sont pas détectées avec une précision parfaite.
- >Pas encore de gestion des niveaux ou difficulté progressive.

Projectile.h

```
// Auteur : Yazid El Mahi & Younès Tadili

#ifndef PROJECTILE_H
#define PROJECTILE_H

#include <QVector3D>
#include <GL/gl.h>
#include <algorithm> // pour std::clamp

// Enumération des types de projectiles possibles
enum class ProjectileType {
    Sphere,
    Cone,
    Cylindre,
    Cube
};

class Projectile
{
public:
    // Constructeur : initialise position, vitesse, type, texture
    Projectile(const QVector3D& position, const QVector3D& velocity, ProjectileType type, GLu

    // Destructeur
    ~Projectile();

    // Met à jour la position et la rotation en fonction du temps delta (en secondes)
    void update(float deltaTime);

    // Dessine le projectile selon son type (sphere, cube, etc)
    void draw() const;
```

```

// Renvoie la position actuelle du projectile
QVector3D getPosition() const;

// Renvoie le type du projectile
ProjectileType getType() const;

private:
    // Dessine un cube simple pour les projectiles de type cube
    void drawCube() const;

    QVector3D m_position;      // Position actuelle 3D
    QVector3D m_velocity;     // Vitesse 3D actuelle
    ProjectileType m_type;     // Type de projectile
    GLuint m_textureId;       // Texture OpenGL associée

    float m_rotationAngle = 0.0f; // Angle de rotation actuel (degrés)
    QVector3D m_rotationAxis;   // Axe de rotation normalisé
};

#endif // PROJECTILE_H

```

Fragment.h

```

// Auteur : Yazid El Mahi & Younès Tadili

#ifndef FRAGMENT_H
#define FRAGMENT_H

#include <QVector3D>
#include <GL/gl.h>
#include <GL/glu.h>

#include "projectile.h" // Pour ProjectileType

class Fragment
{
public:
    // Constructeur : position initiale, vitesse, vitesse de rotation, type, texture
    Fragment(const QVector3D& position, const QVector3D& velocity, float rotationSpeed, Proj

    // Met à jour la position, vitesse, rotation et durée de vie
    void update(float deltaTime);

    // Renvoie vrai si le fragment est encore visible (durée de vie > 0 et pas trop bas)
    bool isAlive() const;

    // Dessine le fragment (cube, sphere, cone, cylindre)
    void draw() const;

private:

```

```

void drawCubeFragment() const;
void drawSphereFragment() const;
void drawConeFragment() const;
void drawCylinderFragment() const;

QVector3D m_position;           // Position 3D actuelle
QVector3D m_velocity;          // Vitesse 3D actuelle
float m_rotation;              // Angle de rotation actuel (degrés)
float m_rotationSpeed;         // Vitesse de rotation (degrés/s)
float m_lifetime;              // Durée de vie restante (secondes)
ProjectileType m_type;         // Type du fragment
GLuint m_textureId;           // Texture OpenGL associée
};

#endif // FRAGMENT_H

```

SceneGL.h

```

// Auteur : Yazid El Mahi & Younès Tadili

#ifndef SCENEGL_H
#define SCENEGL_H

#include <QOpenGLWidget>
#include <QTimer>
#include <QEapsedTimer>
#include <vector>

#include "projectile.h"
#include "fragment.h"

class SceneGL : public QOpenGLWidget
{
    Q_OBJECT

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

    // Gestion du jeu
    void startGame();
    void stopGame();
    void resetGame();
    void resumeGame();
    void setGamePaused(bool paused);

    // Met à jour la position virtuelle de la main (position 2D convertie en 3D)
    void setMainVirtuellePos2D(const QPoint& pos);

signals:
    void gameOver();

```

```
protected:
    // Fonctions OpenGL
    void initializeGL() override;
    void resizeGL(int w, int h) override;
    void paintGL() override;

    // Gestion des événements souris (optionnel)
    void mouseMoveEvent(QMouseEvent* event) override;

private:
    // Chargement des textures
    void loadTexture();
    GLuint chargerTexture(const QString& path);

    // Dessins spécifiques
    void drawCube(float size);
    void drawPortailNether();
    void drawSkybox();
    void drawSol();
    void drawMurs();
    void drawZoneDecoupe();
    void drawMainVirtuelle();

    // Mécanique du jeu
    void lancerProjectile();
    void updateScene();

    // Attributs
    std::vector<Projectile> m_projectiles;      // Liste des projectiles actifs
    std::vector<Fragment> m_fragments;           // Liste des fragments générés

    GLuint m_textureFireball = 0;
    GLuint m_textureFleche = 0;
    GLuint m_textureTNT = 0;
    GLuint m_textureSol = 0;
    GLuint m_textureMur = 0;
    GLuint m_textureCiel = 0;
    GLuint m_textureObsidienne = 0;
    GLuint m_texturePortail = 0;
    GLuint m_sabreTexture = 0;

    QVector3D m_mainVirtuellePos3D;                // Position 3D virtuelle de la main

    QTimer* m_timer;
    QElapsedTimer m_elapsed;
    QElapsedTimer m_gameTimer;

    float m_gameTimeSeconds = 0.0f;
    float m_tempsDepuisDernierTir = 0.0f;
```

```
int m_score = 0;
int m_pv = 3;

bool m_gameStarted = false;
bool m_gamePaused = false;
bool m_useMouseInput;

qint64 m_elapsedPausedTime = 0;
};

#endif // SCENEGL_H
```

Widget.h

```
// Auteur : Yazid El Mahi & Younès Tadili

#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include <QThread>
#include <QTimer>
#include <QMutex>
#include <opencv2/opencv.hpp>

#include "scenegl.h"

class Widget : public QWidget
{
    Q_OBJECT

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

    void startGame();
    void stopGame();

protected:
    void updateUI();
    void captureAndProcess();

private slots:
    void on_startGameButton_clicked();
    void on_stopButton_clicked();
    void on_quitButton_clicked();
    void on_captureButton_clicked();

    void handleGameOver();

private:
```

```

bool detectHand(const cv::Mat& image, cv::Point& handPosition);
bool detectHandByColor(const cv::Mat& image, cv::Point& handPosition);
bool detectHandByContour(const cv::Mat& image, cv::Point& handPosition);
cv::Point predictKalmanFilter(const cv::Point& measurement);
void initKalmanFilter();

Ui::Widget *ui;
SceneGL* scene_;

cv::VideoCapture* webCam_;
cv::Mat latestFrame_;
cv::Mat prevFrame_;
bool hasPrevFrame_;

bool handDetected_;
cv::Point handPosition_;
QMutex handPositionMutex_;

QTimer* uiTimer_;
std::thread captureThread_;

cv::KalmanFilter KF_;
cv::Mat<float> measurement_;

bool isRunning_;
bool m_gameStarted;
bool m_gamePaused = false;
};

#endif // WIDGET_H

```

MenuWindow.h

```

// Auteur : Yazid El Mahi & Younès Tadili

#ifndef MENUWINDOW_H
#define MENUWINDOW_H

#include <QWidget>
#include <QLabel>

namespace Ui {
class MenuWindow;
}

class MenuWindow : public QWidget
{
    Q_OBJECT

public:
    explicit MenuWindow(QWidget *parent = nullptr);

```

```
~MenuWindow();  
  
signals:  
    void startGameRequested();  
  
private slots:  
    void on_startButton_clicked();  
    void on_quitButton_clicked();  
  
protected:  
    void resizeEvent(QResizeEvent *event) override;  
  
private:  
    Ui::MenuWindow *ui;  
    QLabel* backgroundLabel;  
};  
  
#endif // MENUWINDOW_H
```

MainWindow.h

```
// Auteur : Yazid El Mahi & Younès Tadili  
  
#ifndef MAINWINDOW_H  
#define MAINWINDOW_H  
  
#include < QMainWindow>  
  
class SceneGL;  
  
class MainWindow : public QMainWindow  
{  
    Q_OBJECT  
  
public:  
    explicit MainWindow(QWidget *parent = nullptr);  
    ~MainWindow();  
  
private:  
    SceneGL* scene;  
};  
  
#endif // MAINWINDOW_H
```