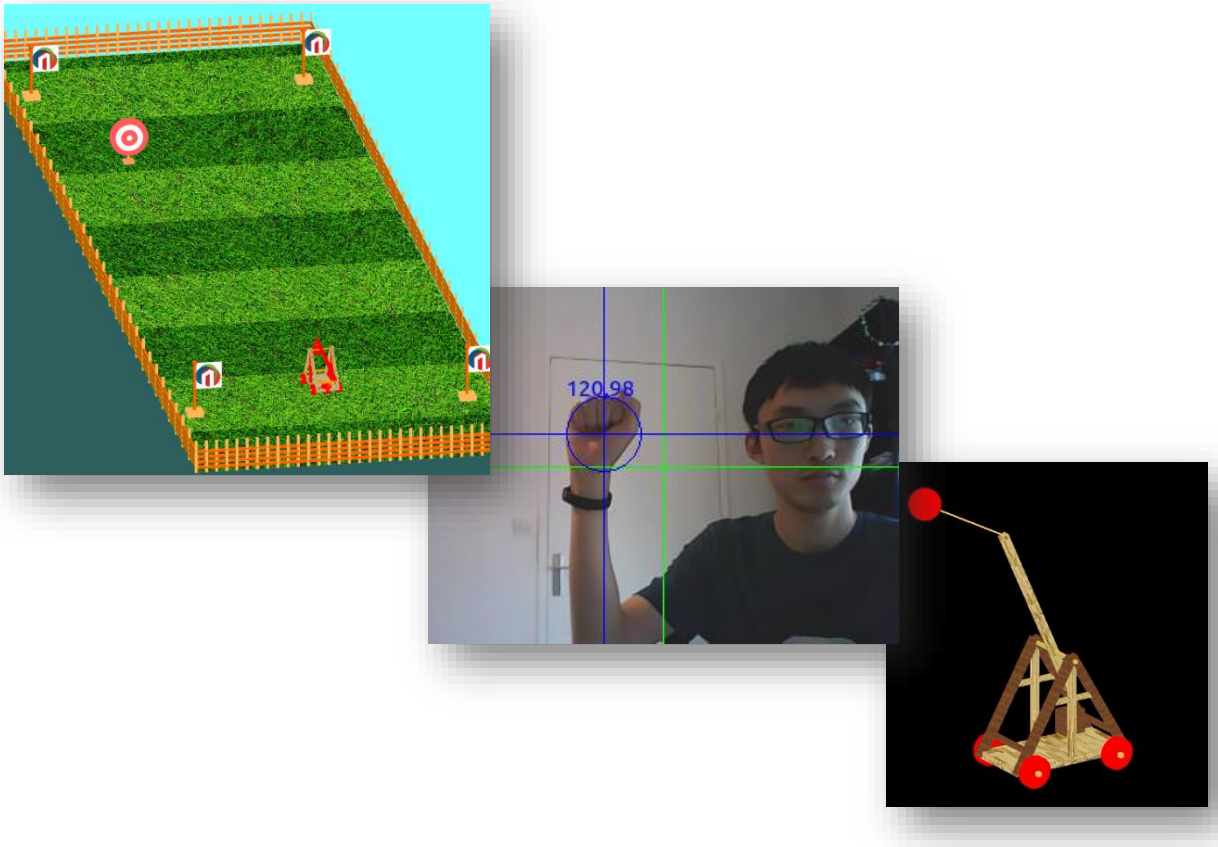
RAPPORT PROJET CATAPULTE

TELECOM SAINT ETIENNE

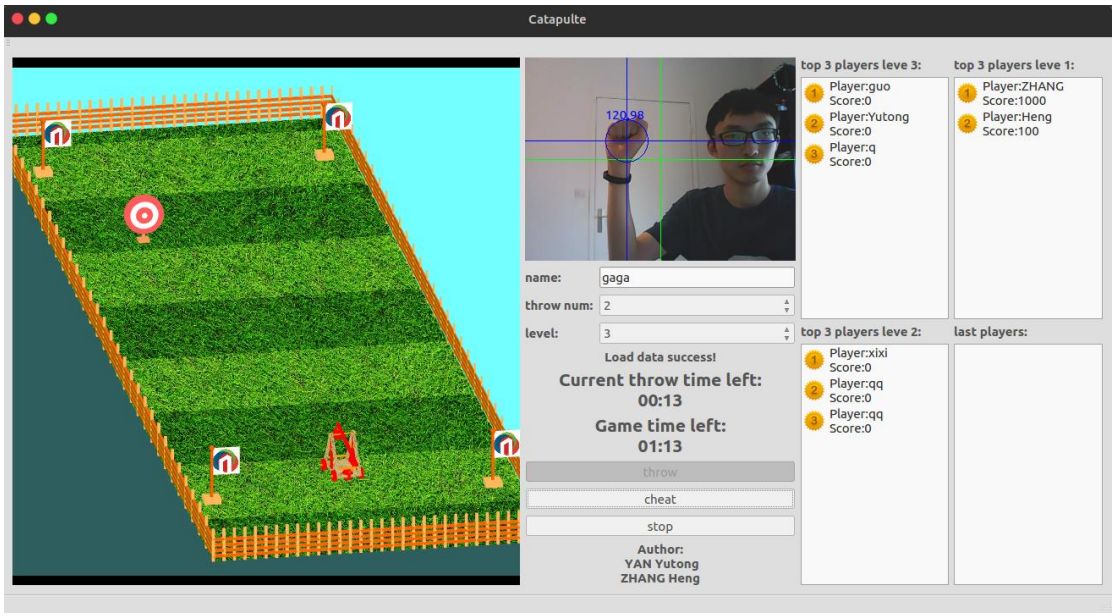


YAN Yutong
ZHANG Heng

Sommaire

Spécification de l'interface graphique

L'interface graphique de notre projet est présente au-dessous :



1. Pour commencer, vous devez remplir votre nom, le nombre de coup que vous voulez jouer, et le niveau de difficulté. Ensuite, vous cliquez sur le Button **<start>** pour lancer le jeu.
2. Si les informations que vous avez remplies sont tous valides, vous allez voir que la fenêtre en temps réel est activée et ça va détecter votre poing. Si le poing est détecté, il y aura un cercle bleu qui est superposé sur votre poing. Sinon, le cercle sera rouge pour vous indiquer que le poing n'est pas détecté.
3. Vous pouvez ajuster l'angle et la force de trébuchet. Le Button **<cheat>** peut vous aider en afficher la direction du jet devant le trébuchet. Quand vous avez fini l'ajustage, vous pouvez lancer votre trébuchet.

Pour le lancer, vous devez justement ouvrir votre paume. Le projectile sera lancé automatiquement quand le paume est détecté à la même position du poing.

4. Quand le projectile est tombé à la cible ou le sol, il y aura une fenêtre pour vous préciser la distance entre le projectile et le centre de la cible. Le score sera noté par le maximum entre dix moins la distance et zéro.

$$Score = max(10 - diantance, 0).$$

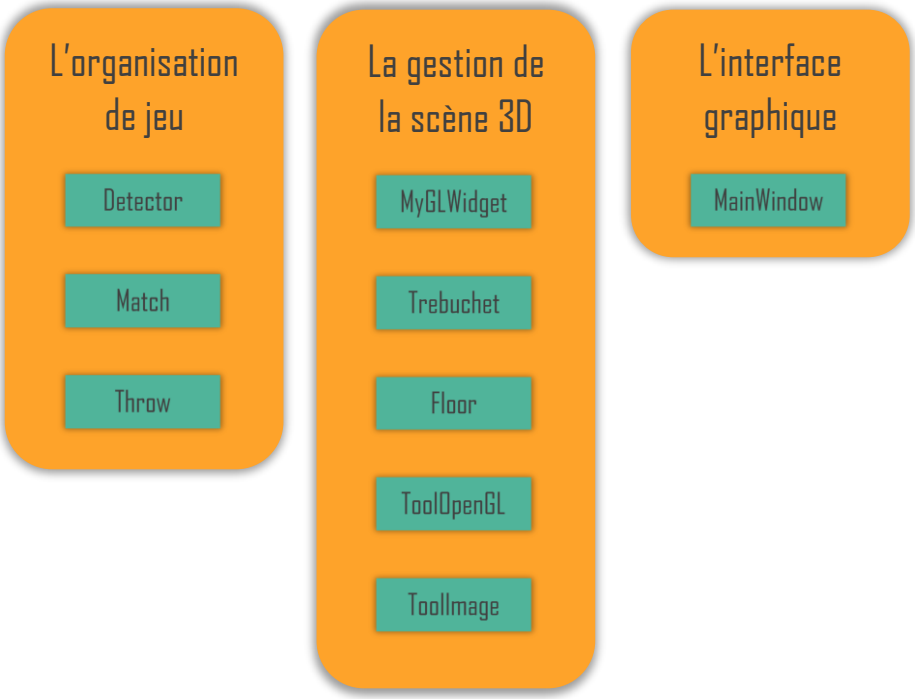
5. Après un certain nombre de coup de lancement du projectile, vous allez voir votre note finale. Elle sera affiche dans la liste **<last players>** en bas à droite de l'interface. Si votre score est supérieur que le score du top3, la liste de **<top 3 players>** sera mis à jour.

Conception

Diagramme des classes

Il y a 3 parties principales dans notre programme :

1. Une partie pour l'organisation de jeu,
2. Une partie pour la gestion de la scène 3D,
3. Une parte pour l'affichage de l'interface.



1. Dans la partie de **l'organisation de jeu**,
    a. La classe **Detector** permet de détecter le poing ou la paume en utilisant l'algorithme Viola-Jones.
    b. La classe **Match** permet de gérer un match avec les paramètres nécessaires (nom de l'utilisateur, le nombre de coup que vous voulez jouer, et le niveau de difficulté).
    c. La classe **Throw** permet de gérer un coup de lancement avec les coordonnées et la hauteur aléatoire selon le niveau de difficulté choisi.
4. Dans la partie de **la gestion de la scène 3D**,
    d. La classe **MyGLWidget** permet d'initialiser la scène 3D et afficher les modèles 3D dans la scène 3D.
    e. La classe **Trebuchet** permet de tracer le trébuchet dans la scène 3D avec plusieurs paramètres (angle de rotation de trébuchet, angle du bras oscillant, etc.).
    f. La classe **Floor** permet de tracer la pelouse, les grilles, le ciel, etc.
    g. La classe **ToolOpenGL** est le <driver> de la bibliothèque OpenGL. Elle contient les méthodes comme : **drawBox**, **drawSphere**, etc. Elle est l'encapsulation de la bibliothèque OpenGL. Nous l'avons réalisé pour

faciliter notre programme et faire la partie de modèle (le trébuchet et la pelouse) indépendante avec la bibliothèque 3D utilisée (OpenGL dans ce cas).

h. La casse **ToolImage** permet de réaliser une série de traitement simple d'image, comme **resizeImage**, **flipImage**, etc.

2. Dans la partie de **l'interface graphique**,
   a. La classe **MainWindow** permet d'afficher correctement les éléments visuels, comme la scène 3D, la fenêtre en temps réel, la liste du match, etc.

Détection du poing et de la paume

Plus précisément, nous avons utilisé l'algorithme Viola-Jones pour la détection du poing et de la paume. Concrètement, cet algorithme est déjà intégré dans la bibliothèque **opencv**. Donc nous pouvons déclarer les d étecteurs avec des classifieurs différents (pour le poing et pour la paume). Nous utilisons le classifieur de visage fourni par **opencv**, et ça va nous renvoyer les coordonnées du poing et de la paume détectée.

Principes physiques

Lorsque nous avons lancé le projectile, il y a 2 états possibles :

1. Avant la séparation, le projectile va suivre la corde,
2. Après la séparation, dans l'orientation verticale, c'est un mouvement de chute libre ; dans la direction horizontale, c'est un mouvement uniforme.

Pour le mouvement de chute libre dans l'orientation verticale :

Le déplacement vertical lv est en relation secondaire avec le temps t :

$$lv = a*t*t;$$

Donc la coordonnée de la hauteur de la projectile s est la hauteur du lancement h moins le déplacement vertical lv :

$$z = h - lv = h - a*t*t,\ a \text{ est un constant;}$$

Pour le mouvement uniforme dans la direction horizontale :

La vitesse v est proportionnel à la position verticale du lancement, On met pos la coordonnée verticale de la position du poing, alors la vitesse v est :

$$v = b*pos + c,\ b \text{ et } c \text{ sont des constants;}$$

Du coup le déplacement horizontal lh est :

$$lh = v*t,\ t \text{ est le temps;}$$

Donc le coordonnée du longueur de la projectile x est le longueur de lancement s plus le déplacement horizontal lh:

$$x = s + lh = s + v*t = s + (b*pos + c);$$

L'état de finalisation de l'application

Les fonctions validées :

1. Visualisation globale de la scène (avec les éléments suivants : trébuchet, filets de protection, sol en herbe, logo de TSE, éclairage),
2. Animation du trébuchet pour fixer la rotation et l'inclinaison du bras,
3. Animation du lancer du projectile avec mouvement dans les airs jusqu'à la cible,
4. Gestion de l'apparition des cibles en fonction du niveau de difficulté, affichage du temps, du nombre de cibles restant et du score,
5. Visualisation de l'image de la caméra et de la zone de captation du geste,
6. Initialisation de l'interaction par un mouvement vertical vers le bas,
7. Réglage du trébuchet en suivant le déplacement de la main : orientation du trébuchet par déplacement droite-gauche et inclinaison par déplacement haut-bas,
8. Déclenchement du tir par déplacement rapide vers le bas.

Les fonctions pas finalisées :

Non.

Les bogues :

Inconnu.

Les fichiers entêtes

mainwindow.h

```
class MainWindow : public QMainWindow{

public:

    explicit MainWindow(QWidget *parent = 0);

    ~MainWindow();

private:

    Ui::MainWindow *ui;       // classe d'interface

    QTimer * showRealTimeVideoTimer;          // Timer pour l'affichage de la fenêtre en temps réel

    QTimer * updateScene3DTimer;              // Timer pour l'affichage de la scène 3D

    QTimer * checkHandPositionTimer;          // Timer pour la détection de main

    VideoCapture videoCapture;                // Classe pour le capture de camera

    ToolImage toolImage;                      // classe pour les traitements d'image

    const String fistDetectorPath = "xmlFiles/hand.xml";          // le fichier de classifier de poing

    Detector fistDetector = Detector(fistDetectorPath);           // Classe pour la détection de poing

    const String palmDetectorPath = "xmlFiles/palm.xml";          // le fichier de classifier de paume

    Detector palmDetector = Detector(palmDetectorPath);           // Classe pour la détection de paume

    Mat srcMat;                               // la photo de camera sous forme 'Mat'

    QImage srcQImage;                         // la photo de camera sous forme 'QImage'

    QPixmap srcQPixmap;                       // la photo de camera sous forme 'QPixmap'

    QSize imageSize;                          // la taille de photo

    QPixmap welcomImage = QPixmap("./img/tse.png");          // l'image d'accueil

    void showMessage(QString message);                       // l'affichage d'une message

    void showRealTimeLabel();                                // l'affichage du temps restant

    QPoint currentPos;                                       // La position de main courante

    QPoint lastPos;                                          // La position de main avant

    bool detected = false;                                   // Si le main est détecté

    bool thrown = false;                                     // Si le projectile est lancée

    bool started = false;                                    // Si le jeu est commencé

    Match * currentMatch;                                    // Classe de jeu en cours

    Throw * currentThrow;                                    // Classe de lancement en cours

    void throwBall();                                        // Lancement de projectile

    void updateMatchList();                                  // mettre à jour les scores record
```

```
public slots:
    void updateScene3D();                       // Mettre à jour la scène 3D
    void updateRealTimeCam();                   // Mettre à jour la fenêtre de photo
    void checkHandPosition();                   // La détection de main
private slots:
    void on_startButton_clicked();              // Ce que nous allons faire quand la Button 'start' est cliquée
};
```

match.h

```
class Match{
public:
    Match();                                    // Le constructeur sans paramètres
    Match(string playerName, int nbThrow, int level);  // Le constructeur avec paramètres
    void addThrow(Throw * t){throwList.push_back(t);}  // Ajouter un lancement de projectile
    int getLevel(){return level;}               // Renvoyer le niveau de difficulté
    int getNbThrow(){return nbThrow;}           // Renvoyer le nombre de la séquence
    int getThrowListSize(){return throwList.size();}  // Renvoyer le nombre de lancement
    void addScore(float score){totalScore += score;}  // Ajouter un score
    float getTotalScore(){return totalScore;}   // Renvoyer le score
    QString toQString();                        // Affichage de jeu
    string getPlayerName(){return playerName;}  // Renvoyer le nom de joueur
private:
    string playerName = "NULL";                 // nom de joueur
    int nbThrow = 0;                            // Le nombre de la séquence
    float totalScore = 0;                       // Le score
    int level = 0;                              // Le niveau de difficulté
    list<Throw*> throwList;                     // La liste de lancement
};
```

throw.h

```
class Throw{
public:
    Throw(int level);                           // Le constructeur
    float getTargetPoxX(){return targetPoxX;}   // Renvoyer la position x de cible
    float getTargetPoxY(){return targetPoxY;}   // Renvoyer la position y de cible
```

```
    float getTargetPoxZ(){return targetPoxZ;}        // Renvoyer la position z de cible
private:
    float targetPoxX, targetPoxY, targetPoxZ;        // La position de cible
    int level;                                       // Le niveau de difficulté
};
```

trebuchet.h

```
class Trebuchet{
public:
    Trebuchet();                                     // Le constructeur
    void drawTrebuchet();                            // Tracer le trébuchet
    void setSwingArmAngle(float angle);              // Configuration de l'angle de bras
    void setStandAngle(float angle);                 // Configuration de l'angle de trébuchet
    float getBallX(){return -1*ballX;}               // Renvoyer la position x de projectile
    float getBallY(){return ballY;}                  // Renvoyer la position y de projectile
    float getBallZ(){return ballZ;}                  // Renvoyer la position z de projectile
private:
    float swingArmAngle = 0;                         // L'angle de bras
    float standAngle = 0;                            // L'angle de trébuchet
    float standX, standY, standZ ;                   // La position de trébuchet
    float ballX, ballY, ballZ;                       // La position de projectile
};
```