PROJET : MINISPACE

LEANG Denis

2021-2022

FISE 2

Telecom Saint-Etienne

Table des matières

I – Spécifications utilisateur

- ## Fenêtre : Menu

Au lancement de l'application, l'utilisateur a la possibilité de choisir le niveau de difficulté souhaité en fonction du nombre d'astéroïdes. Il suffit de cliquer sur le numéro correspondant au nombre d'astéroïdes à générer.

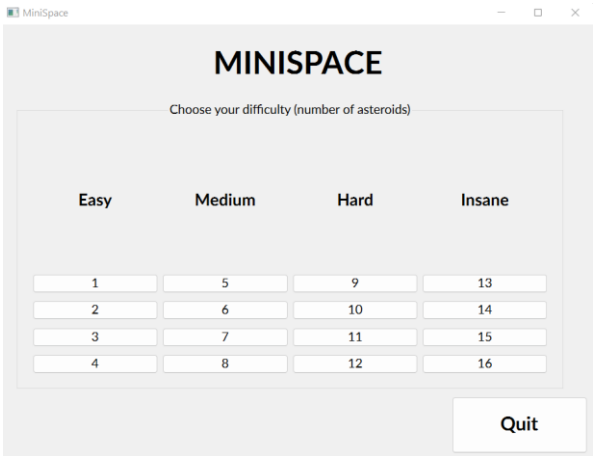
- ## Fenêtre : Game

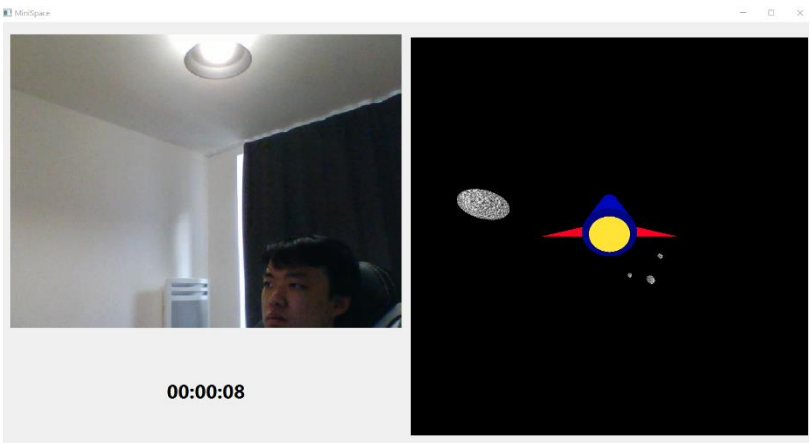
Après le choix de l'utilisateur, une nouvelle fenêtre s'ouvre avec la zone de webcam, la zone de jeu et un chronomètre. Le but du jeu étant d'atteindre la station spatiale sans toucher les astéroïdes, l'utilisateur a la possibilité de se déplacer de deux façons différentes :

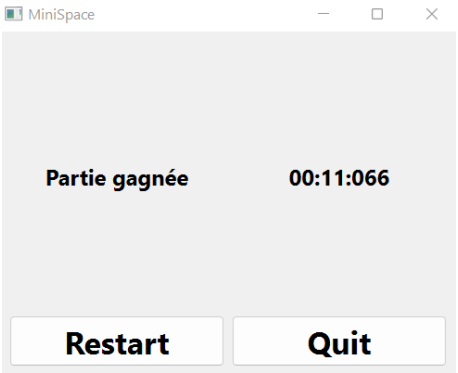
- **Avec le clavier** : les touches suivantes permettent de :
  - **Avancer** : Z
  - **Reculer** : S
  - **Effectuer une rotation** : Touches directionnelles
- **Avec la webcam** :
  - **Avancer** : Deux paumes face à la webcam
  - **Tourner à gauche/droite** : Un poing fermé plus bas que l'autre
  - **Tourner vers le haut/bas** : Deux poings fermés au même niveau

- ## Fenêtre : Result

La fin de partie est détectée lorsque l'utilisateur entre en contact avec un astéroïde (partie perdue) ou avec la station spatiale (partie gagnée). Quoiqu'il arrive, une nouvelle fenêtre s'ouvre à l'utilisateur lui indiquant son résultat et la durée totale de la partie. Il a la possibilité de recommencer une nouvelle partie (**Restart**) ou de quitter l'application (**Quit**).







II – Conception de l'application

- ## Détail des classes

L'application est constituée de 8 classes au total :

- 3 classes sont réservées au dessin des objets sur OpenGL :

Asteroid : La classe permet d'afficher un astéroïde sphérique muni d'une texture et des matériaux souhaités. Elle prend en paramètres les coordonnées (x,y,z) ainsi que le rayon de la sphère.

Station : La classe permet d'afficher une station spatiale munie d'une texture et des matériaux souhaités. Elle prend en paramètres les coordonnées (x,y,z) ainsi que le rayon de la sphère du corps central.

Starship : La classe permet d'afficher un vaisseau spatial muni des matériaux souhaités. Elle prend en paramètres les coordonnées (x,y,z) ainsi que des angles θ et φ pour la gestion des mouvements en coordonnées sphériques. Cette classe permet également d'effectuer des translations et rotations du vaisseau.

- 4 classes sont réservées à l'interface :

Menu : La classe permet d'afficher le menu principal du jeu. Elle entretient une relation de composition unique avec la classe Game et elle est composée de 1 objet de celle-ci.

Game : La classe permet d'afficher la fenêtre principale du jeu. Elle affiche le QWidget de la webcam, le QOpenGLWidget du jeu et le QLabel du chronomètre. La classe entretient une relation de composition avec la classe SpaceWidget et est composée de 1 objet de celle-ci. Elle promeut également un QWidget de la classe Webcam.

Webcam : La classe permet d'afficher un widget contenant l'affichage de la webcam ainsi que la détection des poings et paumes. Elle renvoie également un entier statique arbitraire vers la classe SpaceWidget afin de gérer les mouvements à partir de la webcam.

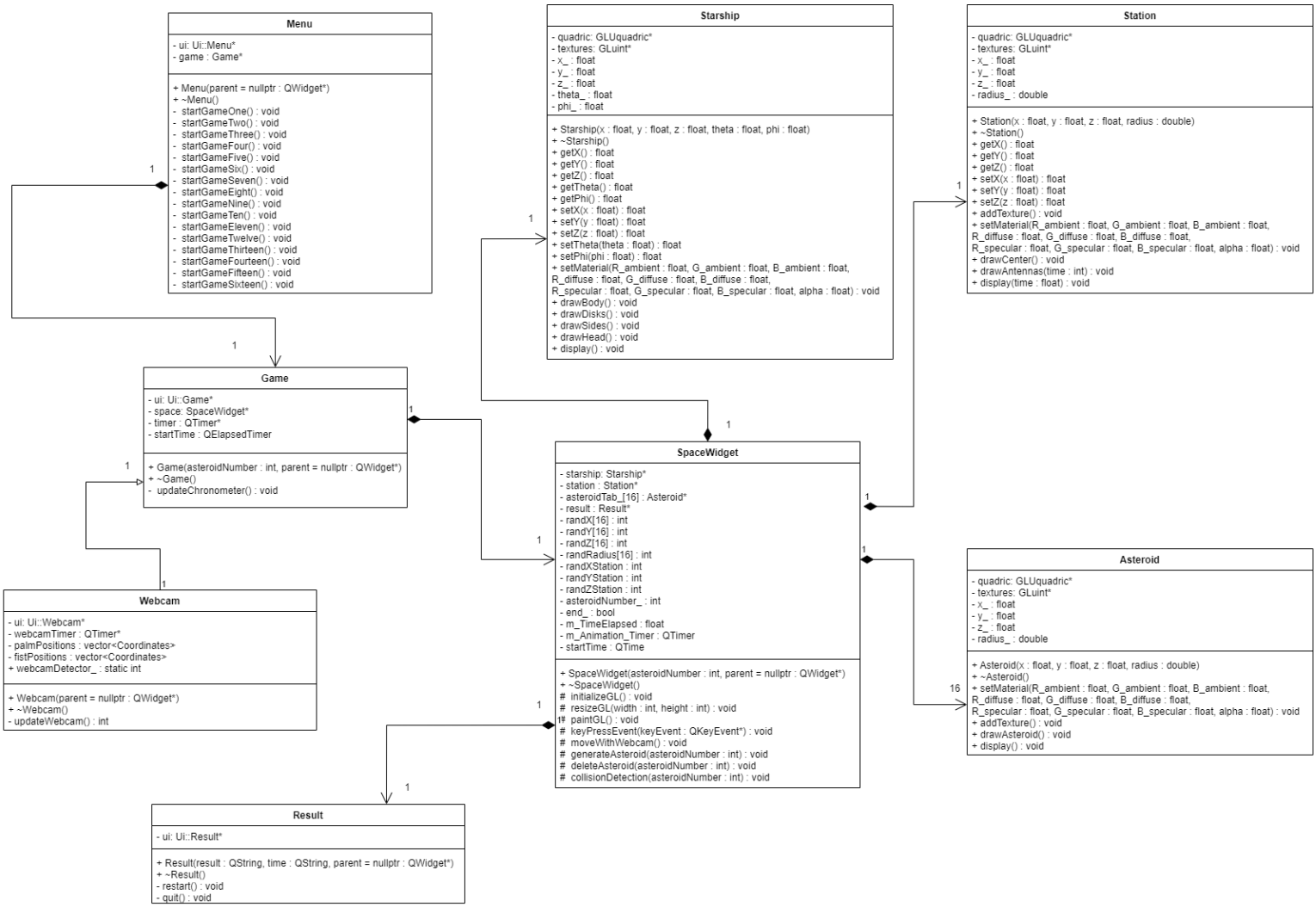
Result : La classe permet d'afficher la fenêtre finale du jeu. Elle contient le résultat de la partie, la durée totale de la partie ainsi que la possibilité de rejouer ou de quitter la partie.

- 1 classe sert de hub entre les classes interface et OpenGL :

SpaceWidget : Cette classe est le cœur de l'application. Elle est composée de nombreuses instances des classes Asteroid, Station et Starship. Elle gère l'affichage de ceux-ci dans un QOpenGLWidget avec les fonctions d'OpenGL. Ensuite, la classe offre également la gestion des déplacements du joueur dans l'espace avec soit les touches du clavier, soit la webcam grâce à un entier statique récupéré dans la classe Webcam. Enfin, la classe gère la détection des collisions et donc la fin de partie et est par conséquence composée d'instances de la classe Result.

- **Diagramme des classes**

Le diagramme de classes de l'application est le suivant :



III – Etat de finalisation de l'application

1. Cette fonctionnalité est réalisée dans son ensemble. Le choix du nombre d'astéroïdes fonctionne et est proposé à l'utilisateur. Leur position et taille sont définies de façon aléatoire à l'aide de QRandomGenerator. Enfin, une texture d'astéroïde est appliquée à l'ensemble des astéroïdes.

2. Le vaisseau spatial est constitué d'un corps cylindrique bleu muni d'une tête blanche en forme de cône. Sur les côtés inférieurs du vaisseau, des triangles rouges sont dessinés. Enfin, un réacteur en forme de disque est représenté à la base avec une couleur jaune. La gestion des matériaux et reflets a été effectué. En revanche, l'éclairage directionnel n'est pas très visible bien que la lampe ait été définie dans le SpaceWidget.

3. La possibilité de se déplacer avec la webcam a été implémenté. L'utilisateur peut avancer tout droit en maintenant ses deux paumes face à la webcam (la détection peut être approximative et il faut persévérer jusqu'à voir deux rectangles verts). Ensuite, la rotation du vaisseau est également possible en maintenant ses deux poings fermés face à la webcam. Pour tourner à gauche ou à droite, il suffit de monter le poing de gauche ou de droite vers le haut en maintenant l'autre immobile. La caméra suit bien le vaisseau.

4. Un widget promu de la classe Webcam est présent dans le widget Game et il représente la zone d'affichage de la webcam. La détection a été implémenté : les rectangles verts détectent les paumes alors que les rectangles bleus détectent les poings fermés.

5. La station spatiale est constituée d'un corps principal sphérique munie d'une texture. Des antennes bleue et rouge sont dessinées sur les côtés de la sphère. Une lampe clignotante de la forme d'un cylindre avec un disque a été mise en place et celle-ci clignote à intervalles réguliers. La station peut également tourner sur elle-même. Enfin, une texture du logo de TSE a été ajouté sur un disque adjacent à la sphère.

6. La détection des collisions est fonctionnelle en considérant chaque objet (vaisseau spatial, station, astéroïdes) comme étant des sphères et en calculant la distance séparant les centres. En cas de contact avec une astéroïde ou la station, la partie est respectivement perdue ou gagnée.

7. Lors de la détection de la collision, la fin de partie est déclarée et une fenêtre s'ouvre offrant le choix à l'utilisateur de débuter une nouvelle partie ou quitter le jeu. Le résultat final ainsi que la durée de jeu sont également affichés sur cette fenêtre. En revanche, recommencer une partie après en avoir fini une semble ralentir l'application et le nombre de FPS diminue drastiquement. Il est également nécessaire de fermer manuellement le processus de l'exécutable dans le gestionnaire des taches pour pouvoir relancer l'application.

IV – Fichiers d'entête

- ## Asteroid

```
#include <qopengl.h>
#include <GL/gl.h>
#include <GL/glu.h>
#include <QImage>

#pragma once
class Asteroid {

public:
    Asteroid(float x, float y, float z, double radius);
    ~Asteroid();

    void setMaterial(float R_ambient, float G_ambient, float B_ambient,
                     float R_diffuse, float G_diffuse, float B_diffuse,
                     float R_specular, float G_specular, float B_specular,
                     float alpha) const;
    void addTexture() const;
    void drawAsteroid() const;
    void display() const;

private:
    GLUquadric * quadric{nullptr};
    GLuint* textures = new GLuint[1];
    float x_;
    float y_;
    float z_;
    double radius_;
};

```

Champs :

- **x_** : Abscisse de l'astéroïde
- **y_** : Ordonnée de l'astéroïde
- **z_** : Côte de l'astéroïde
- **radius_** : Rayon de l'astéroïde
- **textures** : Tableau des textures
- **quadric** : Quadrique permettant les dessins OpenGL

Méthodes :

- **setMaterial** : Applique les matériaux sur les quadriques dessinés et prend en paramètres les valeurs RGB pour les composantes ambiante, diffuse et spéculaire
- **addTexture** : Applique la texture à l'astéroïde
- **drawAsteroid** : Dessine l'astéroïde avec le quadrique et les fonctions setMaterial et addTexture
- **display** : Affiche l'astéroïde dans la zone de dessin

- **Station**

```
#ifndef STATION_H
#define STATION_H

#include <qopengl.h>
#include <GL/gl.h>
#include <GL/glu.h>
#include <QImage>

#pragma once
class Station {

public:
    Station(float x, float y, float z, double radius);
    ~Station();

    float getX(){return x_;};
    float getY(){return y_;};
    float getZ(){return z_;};
    void setX(float x){x_ = x;};
    void setY(float y){y_ = y;};
    void setZ(float z){z_ = z;};
    void addTexture() const;
    void setMaterial(float R_ambient, float G_ambient, float B_ambient,
                     float R_diffuse, float G_diffuse, float B_diffuse,
                     float R_specular, float G_specular, float B_specular,
                     float alpha) const;
    void drawCenter() const;
    void drawAntennas(int time) const;
    void display(float time) const;

private:
    GLUquadric * quadric{nullptr};
    GLuint* textures = new GLuint[2];
    float x_;
    float y_;
    float z_;
    double radius_;
};

#endif // STATION_H

```

Champs :

- **x_** : Abscisse de la station
- **y_** : Ordonnée de la station
- **z_** : Côte de la station
- **radius_** : Rayon du corps central de la station
- **textures** : Tableau des textures
- **quadric** : Quadrique permettant les dessins OpenGL

Méthodes :

- **getX**, **getY**, **getZ** : Permet de retourner l'abscisse/l'ordonnée/la côte de la station
- **setX**, **setY**, **setZ** : Permet de modifier la valeur de l'abscisse/l'ordonnée/la côte de la station
- **addTexture** : Applique les textures à la station
- **setMaterial** : Applique les matériaux sur les quadriques dessinés et prends en paramètres les valeurs RGB pour les composantes ambiante, diffuse et spéculaire
- **drawCenter** : Dessine le corps principal sphérique avec le quadrique et les fonctions setMaterial et addTexture
- **drawAntennas** : Dessine les antennes de forme cylindrique avec des disques avec le quadrique et les fonctions setMaterial et addTexture
- **display** : Affiche la station dans la zone de dessin

- **Starship**

```
#ifndef STARSHIP_H
#define STARSHIP_H

#include <qopengl.h>
#include <GL/gl.h>
#include <GL/glu.h>
#include <QImage>

#pragma once
class Starship {

public:
    Starship(float x, float y, float z, float theta, float phi);
    ~Starship();

    float getX(){return x_;};
    float getY(){return y_;};
    float getZ(){return z_;};
    float getTheta(){return theta_;};
    float getPhi(){return phi_;};
    void setX(float x){x_ = x;};
    void setY(float y){y_ = y;};
    void setZ(float z){z_ = z;};
    void setTheta(float theta){theta_ = theta;};
    void setPhi(float phi){phi_ = phi;};
    void setMaterial(float R_ambient, float G_ambient, float B_ambient,
                     float R_diffuse, float G_diffuse, float B_diffuse,
                     float R_specular, float G_specular, float B_specular,
                     float alpha) const;
    void drawBody() const;
    void drawDisks() const;
    void drawSides() const;
    void drawHead() const;
    void display() const;

private:
    GLUquadric * quadric{nullptr};
    float x_;
    float y_;
    float z_;
    float theta_;
    float phi_;
};

#endif // STARSHIP_H

```

Champs :

- **x_** : Abscisse du vaisseau
- **y_** : Ordonnée du vaisseau
- **z_** : Côte du vaisseau
- **theta_** : Angle de colatitude
- **phi_** : Angle de longitude
- **quadric** : Quadrique permettant les dessins OpenGL

Méthodes :

- **getX**, **getY**, **getZ**, **getTheta**, **getPhi** : Permet de retourner l'abscisse/l'ordonnée/la côte/la colatitude/la longitude du vaisseau
- **setX**, **setY**, **setZ**, **setTheta**, **setPhi** : Permet de modifier la valeur de l'abscisse/l'ordonnée/la côte/la colatitude/la longitude du vaisseau
- **setMaterial** : Applique les matériaux sur les quadriques dessinés et prends en paramètres les valeurs RGB pour les composantes ambiante, diffuse et spéculaire
- **drawBody** : Dessine le corps principal cylindrique avec le quadrique et la fonction setMaterial
- **drawDisks** : Dessine les disques refermant le corps cylindrique principal avec le quadrique et la fonction setMaterial
- **drawSides** : Dessine les triangles à la base du cylindre à l'aide de GL_QUADS et la fonction setMaterial
- **drawHead** : Dessine la pointe conique reliée au corps cylindrique principal avec le quadrique et la fonction setMaterial
- **display** : Affiche le vaisseau dans la zone de dessin

- ## Menu

```h
#ifndef MENU_H
#define MENU_H

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

QT_BEGIN_NAMESPACE
namespace Ui { class Menu; }
QT_END_NAMESPACE

#pragma once
class Menu : public QWidget {
    Q_OBJECT

public:
    Menu(QWidget *parent = nullptr);
    ~Menu();

private:
    Ui::Menu *ui;
    Game* game;

private slots:
    void startGameOne();
    void startGameTwo();
    void startGameThree();
    void startGameFour();
    void startGameFive();
    void startGameSix();
    void startGameSeven();
    void startGameEight();
    void startGameNine();
    void startGameTen();
    void startGameEleven();
    void startGameTwelve();
    void startGameThirteen();
    void startGameFourteen();
    void startGameFifteen();
    void startGameSixteen();
};
#endif // MENU_H

```

Champs :

- **ui** : Variable de l'ui de la classe Menu
- **game** : Instance de classe Game

Méthodes :

- **startGame…** : Ouvre la fenêtre de jeu principal avec le nombre d'astéroïdes correspondant en fonction du numéro cliqué par l'utilisateur dans le menu

- **Game**

```
#ifndef GAME_H
#define GAME_H

#include "spacewidget.h"
#include <QWidget>
#include <QTimer>
#include <QTime>
#include <QElapsedTimer>

QT_BEGIN_NAMESPACE
namespace Ui { class Game; }
QT_END_NAMESPACE

#pragma once
class Game : public QWidget {
    Q_OBJECT

public:
    Game(int asteroidNumber, QWidget *parent = nullptr);
    ~Game();

private:
    Ui::Game *ui;
    SpaceWidget* space;
    QTimer* timer;
    QElapsedTimer startTime;

private slots:
    void updateChronometer();
};
#endif // GAME_H

```

Champs :

- **ui** : Variable de l'ui de la classe Game
- **space** : Instance de classe SpaceWidget
- **time** : Objet de classe QTimer
- **startTime** : Objet de classe QElapsedTimer

Méthodes :

- **updateChronometer** : Slot permettant d'afficher en temps réel un chronomètre dans l'interface mesurant la durée totale de la partie

- **Webcam**

```
#ifndef WEBCAM_H
#define WEBCAM_H

#include "opencv2/opencv.hpp"
#include <QWidget>
#include <QTimer>

QT_BEGIN_NAMESPACE
namespace Ui { class Webcam; }
QT_END_NAMESPACE

struct Coordinates {
    int x;
    int y;
};

#pragma once
class Webcam : public QWidget {
    Q_OBJECT

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

    static int webcamDetector_;

private:
    Ui::Webcam *ui;
    QTimer* webcamTimer;
    std::vector<Coordinates> palmPositions;
    std::vector<Coordinates> fistPositions;

private slots:
    int updateWebcam();
};

#endif // WEBCAM_H

```

Champs :

- **ui** : Variable de l'ui de la classe Webcam
- **webcamTimer** : Objet de classe QTimer
- **palmPositions** : Coordonnées x et y de la position des paumes
- **fistPositions** : Coordonnées x et y de la position des poings
- **webcamDetector_** : Entier servant à l'activation des conditions if dans SpaceWidget permettant les mouvements du vaisseau à partir de la webcam

Méthodes :

- **updateWebcam** : Slot permettant d'afficher en temps réel l'image renvoyée par la webcam. Permet également de détecter les poings et paumes en dessinant des rectangles et en mettant à jour la valeur de webcamDetector_

- **Result**

```
#ifndef RESULT_H
#define RESULT_H

#include <QWidget>
#include <QProcess>

QT_BEGIN_NAMESPACE
namespace Ui { class Result; }
QT_END_NAMESPACE

#pragma once
class Result : public QWidget {
    Q_OBJECT

public:
    Result(QString result, QString time, QWidget *parent = nullptr);
    ~Result();

private:
    Ui::Result *ui;

private slots:
    void restart();
    void quit();
};

#endif // RESULT_H

```

Champs :

- **ui** : Variable de l'ui de la classe Result

Méthodes :

- **restart** : Slot permettant de redémarrer l'application pour débuter une nouvelle partie
- **quit** : Slot permettant de quitter l'application

- **SpaceWidget**

```cpp
#include "asteroid.h"
#include "starship.h"
#include "station.h"
#include "result.h"
#include "webcam.h"
#include <QOpenGLWidget>
#include <QKeyEvent>
#include <QTimer>
#include <QTime>
#include <QApplication>
#include <QRandomGenerator>
#include <cmath>
#include <algorithm>

#pragma once
class SpaceWidget : public QOpenGLWidget {

public:
    SpaceWidget(int asteroidNumber, QWidget * parent = nullptr);
    ~SpaceWidget();

protected:
    void initializeGL();
    void resizeGL(int width, int height);
    void paintGL();
    void keyPressEvent(QKeyEvent * keyEvent);
    void moveWithWebcam();
    void generateAsteroid(int asteroidNumber);
    void deleteAsteroid(int asteroidNumber);
    void collisionDetection(int asteroidNumber);

private:
    Starship* starship = nullptr;
    Station* station = nullptr;
    Asteroid* asteroidTab_[16];

    int randX[16];
    int randY[16];
    int randZ[16];
    int randRadius[16];
    int randXStation;
    int randYStation;
    int randZStation;

    int asteroidNumber_;
    bool end_= true;

    float m_TimeElapsed { 0.0f };
    QTimer m_AnimationTimer;

    Result* result;
    QTime startTime = QTime::currentTime();
};

```

Champs :

- **starship** : Instance de classe Starship
- **station** : Instance de classe Station
- **asteroidTab_** : Tableau d'instances de classe Asteroid
- **randX** : Tableau des abscisses aléatoires des astéroïdes
- **randY** : Tableau des ordonnées aléatoires des astéroïdes
- **randZ** : Tableau des côtes aléatoires des astéroïdes
- **randRadius** : Tableau des rayons aléatoires des astéroïdes
- **randXStation** : Abscisse aléatoire de la station
- **randYStation** : Ordonnée aléatoire de la station
- **randZStation** : Côte aléatoire de la station
- **asteroidNumber_** : Nombre d'astéroïdes à générer
- **end_** : Condition de rentrée de boucle if pour la fin de partie
- **m_TimeElapsed** : Temps
- **m_AnimationTimer** : Objet de classe QTimer
- **result** : Instance de classe Result
- **startTime** : Objet de classe QTime

Méthodes :

- **initializeGL** : Permet d'initialiser les fonctions OpenGL
- **resizeGL** : Permet de gérer la taille de la fenêtre
- **paintGL** : Permet de dessiner la scène et gère la caméra qui suit le vaisseau.
- **keyPressEvent** : Permet de bouger le vaisseau avec les touches directionnelles et les lettres Z et S
- **moveWithWebcam** : Permet de bouger le vaisseau avec la webcam en récupérant l'entier statique de la classe Webcam nommé webcamDetector_
- **generateAsteroid** : Permet de générer le bon nombre d'astéroïdes à partir du paramètre en entrée
- **deleteAsteroid** : Permet de détruire les astéroïdes pour éviter les fuites mémoires
- **collisionDetection** : Permet de gérer les collisions entre vaisseau et station/astéroïdes. Déclenche également la fenêtre de fin de partie (Result) en cas de collision