IData.hpp : INTERFACE

Methode :

* Enum class DataType :
  + Contains the set of shapes and other objects available.
* Virtual ~IData () :
  + Default destructor
* Virtual DataType const &getShape() const :
  + Retrieve the DataType of the object..
* virtual void setShape(DataType const &shape) :
  + Allows to modify the DataType of the object.
* virtual Position const &getPosition() const
  + Retrieves the position of the object.
* virtual void setPosition(Position const &position)
  + Allows you to change the position of the object.
* virtual void reset(void)
  + Allows reset.

AData.hpp : IData

Methode :

* Virtual ~AData () :
  + Default destructor.
* DataType const &getShape() const :
  + Retrieve the DataType of the object according to the DataType
* virtual void setShape(DataType const &shape) :
  + Allows to modify the DataType of the object.
* virtual Position const &getPosition() const
  + Retrieves the position of the object.
* virtual void setPosition(Position const &position)
  + Allows you to change the position of the object.
* virtual void setPositionX(double const x);
  + Allows you to change the position X.
* virtual void setPositionY(double const y);
  + Allows you to change the position Y.
* virtual void setPositionZ(double const z);
  + Allows you to change the position Z (3D).
* virtual void reset(void)
  + Allows reset.

Protected:

* DataType \_shape
  + Contains the type of the object.
* Position \_position
  + Contains the position of the object.

AScene.hpp : AData

Methode :

* Float getIncX() const
  + Allows you to change the position X (3D).
* virtual void setIncX(float incX);
  + Allows you to change the position X (3D).
* Float getIncY() const
  + Retrieves position Y (3D).
* virtual void setIncY(float incY);
  + Retrieves position Y (3D).
* virtual void reset(void)
  + Allows reset.

Protected:

* Float \_incX
  + Contains the X position (3D).
* Float \_incY
  + Contains the Y position (3D).

AScene.hpp->AData Allows to retrieve the X, Y, and Z positons of an object for 3D viewing, stores its position X, Y and Z

Audio.hpp : AData

Methode :

* explicit Audio (std ::string const & audio, unsigned int intensity = 100, bool repeat = false)
  + Default constructor.
* ~Audio ()
  + Default destructor.
* std::string const &getAudio() const
  + Lets you retrieve the Audio file.
* bool getRepeat() const
  + Allows you to change the boolean repeat value.
* void setRepeat(bool repeat)
  + Allows you to change the value of the repeat boolean.
* unsigned int getIntensity() const
  + Used to retrieve the intensity value.
* void setIntensity(unsigned int intensity)
  + Allows you to change the intensity value.
* void reset(void)
  + Allows reset.

Protected:

* Std::string \_audio
  + Contains the name of the audio file.
* Unsigned int \_intensity
* Contains the intensity value.
* Bool \_repeat
  + Contains the value of the repeating boolean.

Audio.hpp->AData (Allows you to load an audio file and manage its intensity as well as repeat or not the song, store the song name, its intensity as well as a boolean of repetition)

AVisual.hpp : AData

Methode :

* explicit Audio (std ::string const & audio, unsigned int intensity = 100, bool repeat = false)
  + Default constructor.
* ~AVisual ()
  + Default destructor.
* std::string const &getTexture() const
  + Allows to retrieve the texture.
* virtual void setTexture(Texture const &texture)
  + Allows to modify the texture.
* unsigned int getZIndex() const
  + Retrieves the value Z (3D) .
* virtual void setZIndex(unsigned int zIndex)
  + Allows yo modify the Z position (3D)..
* unsigned int getId() const
  + Retrieves object ID.
* virtual void setId(unsigned int id)
  + Allows you to modify object ID.
* void reset(void)
  + Allows reset.

Protected:

* Texture \_texture
  + Contains the texture of the object.
* Unsigned int \_zIndex
* Contains the value of Z (3D).
* Unsigned int \_id
  + Contains the ID of the object.

AVisual.hpp->AData (Allows you to manage objects to add textures, get their positions and IDs, store texture, X, Y, Z (3D) position and ID)

Camera.hpp : AScene

Methode :

* explicit Camera (Position const &pos, float incX = 0, float incY = 0)
  + Default constructor.
* ~Camera ()
  + Default destructor.
* void reset(void)
  + Allows reset.

Camera.hpp->AScene (Place a camera for a 3D view according to position X, Y and Z recover thanks to the class AScene)

Cube.hpp : AVisual

Methode :

* explicit Cube(Position const &pos, Position const &size, Texture const &text = Texture(), unsigned int zIndex = 0, unsigned int id = 0)
  + Default constructor .
* ~Cube ()
  + Default destructor.
* Position const &getSize() const;
  + Retrieves the size of the object.
* void setSize(Position const &size)
  + Allows you to change the size of the object.
* bool inLine(double a, double new\_a, double size) const.
  + Allows to check the hitboxes.
* void reset(void)
  + Allows to reset.

Protected:

* Position \_size
  + Contiains the size of the object.

Cube.hpp->AVisual (Creates a cube at a position, a size and a selected texture and, stores its position.)

Light.hpp : AVisual

Methode :

* explicit Light(Position const &pos, float incX = 0, float incY = 0)
  + Default constructor.
* ~Light ()
  + Default destructor.
* void reset(void)
  + Allows to reset.

Light.hpp->AScene (Allows you to position a light in a 3D scene at position X, Y and Z)

Sphere.hpp : AData

Methode :

* explicit Sphere(Position const &pos, float radius, Texture const &text = Texture(), unsigned int zIndex = 0)
  + Default constructor.
* ~Sphere ()
  + Default destructor.
* float getRadius() const
  + Allows to recover the radiant of the object.
* void setRadius(float radius)
  + Allows to modify the radiant of the object..
* void reset(void)
  + Allows to reset.

Protected:

* float \_radius
  + Contains the value of the radiant of the object.

Sphere.hpp->AVisual (Creates a sphere at a position, a radiant, a texture and the selected X, Y and Z positions, stores its radiant)

Text.hpp : AVisual

Methode :

* enum Align
  + Contains the different possibilities of alignment of the text.
* explicit Text (std::string const &text, unsigned int size, Position const &pos = Position(), unsigned int zIndex = 0)
  + Default constructor.
* ~Text ()
  + Default destructor.
* std::string const &getText() const
  + Lets you retrieve text.
* void setText(std::string const &text)
  + Change the text.
* Align const &getAlign() const
  + Allows you to retrieve the text alignment method.
* void setAlign(Align const &align)
  + Allows you to change the text alignment value.
* unsigned int getSize() const
  + Retrieves the size of the text.
* void setSize(unsigned int size)
  + Allows you to change the size of the text.
* void reset(void)
  + Allows reset.

Protected:

* Std::string \_text
  + Contains the text.
* Align \_align
* Contains the alignment value.
* Unsigned int \_size
  + Contains text size.

Text.hpp->AVisual (Allows you to display text on the screen with a size and position, and a Z position. Stores text to display, alignment mode, and size)

Color.hpp

Methode :

* Color (unsigned char const r = 0, unsigned char const g = 0, unsigned char const b = 0, unsigned char const a = 0)
  + Contains the different possibilities of alignement of the text.
* ~Color ()
  + Default destructor.
* unsigned char getR() const
  + Retrieves the R value of the RGBA color code.
* unsigned char getG() const
  + Retrieves the G value of the RGBA color code.
* unsigned char getB() const
  + Retrieves the B value of the RGBA color code.
* unsigned char getA() const
  + Retrieves the A value of the RGBA color code.
* void setR(unsigned char const r);
  + Allows you to change the R value of the RGBA color code.
* void setR(unsigned char const g);
  + Allows you to change the G value of the RGBA color code.
* void setR(unsigned char const b);
* Allows you to change the B value of the RGBA color code.
* void setR(unsigned char const A);
* Allows you to change the A value of the RGBA color code.
* void setRGBA(unsigned char const x, unsigned char const y, unsigned char const z, unsigned char const a);
  + Allows you to change the RGBA color code value R, G, B, A.

Private:

* Unsigned char \_r
  + Contains the R value of the RGBA color code.
* Unsigned char \_g
  + Contains the G value of the RGBA color code.
* Unsigned char \_b
  + Contains the B value of the RGBA color code.
* Unsigned char \_a
  + Contains the A value of the RGBA color code.

Color.hpp (Takes a color code R, G, B, A)

Model3D.hpp

Methode :

* Model3D (std::string const &object = "", Position const &position = Position(), Position const &\_scale = Position(1, 1, 1), Position const &\_rotation = Position())
  + Default constructor.
* ~Model3D ()
  + Default destructor.
* void setObject(std::string const &object)
  + Set the type of an object.
* std::string const &getObject(void) const
  + Get the type of an object.
* void setScale(Position const &scale)
  + Allow you to modify the size of an 3D object.
* void setScaleX(double x)
  + Allow you to change the size X position of object 3D.
* void setScaleY(double y)
  + Allows you to change the size Y of the object 3D.
* void setScaleZ(double z)
  + Allows you to change the size Z of the object 3D.
* Position const &getScale(void) const
* Retrieves the size of the 3D object.
* void setPosition(Position const &position)
* Allows you to change the position of the 3D object.
* void setPositionX(double x)
* Allows you to change the X position of the 3D object.
* void setPositionY(double y)
  + Allows you to change the Y position of the 3D object.
* void setPositionZ(double z)
  + Allows you to change the Z position of the 3D object.
* Position const &getPosition(void) const
* Allows to retrieve the position of the object on the XYZ position of the 3D object.
* void setRotation(Position const &rotation)
  + Used to set the rotation of an object.
* void setRotationX(double x)
  + Used to set the rotation to the X position of the object.
* void setRotationY(double y)
  + Used to set the rotation to the Y position of the object.
* void setRotationZ(double z)
* Used to set the rotation to the Z position of the object.
* Position const &getRotation(void) const
  + Retrieves XYZ positions from object rotation.

Private:

* std::string \_object
  + 3D file path.

* Position \_scale;
  + Contains the size of the object.
* Position \_position;

Contains the position of the object

* Position \_rotation;
* Contains the position of the rotation of the object.

Position.hpp

Methode :

* Position (double const x = 0, double const y = 0, double const z = 0)
  + Default constructor.
* ~Position ()
  + Default destructor.
* double getX() const
  + Retrieves the position X of the object.
* double getY() const
  + Retrieves the position Y of the object.
* double getZ() const
  + Retrieves the position Z of the object.
* void setX(double const x)
  + Permet de modifier la position X de l’objet.
* void setY(double const Y)
  + Permet de modifier la position Y de l’objet.
* void setZ(double const z)
  + Permet de modifier la position Z de l’objet.

Protected:

* double \_x
* Contains the x value of the position
* double \_y
* Contains the y value of the position
* double \_z
  + Contains the z value of the position

Position.hpp (Prend une position X, Y, Z. Stock les position X, Y, Z

Texture.hpp

Methode :

* Texture (Color const & color = Color())
  + Default constructor.
* Texture (std::string const & sprite, Color const & color = Color(), int rotation = 0, Model3D const &model = Model3D())
  + Constructor for the texture of the object.
* ~Texture ()
  + Default destructor.
* Color const &getColor() const
  + Lets you retrieve the color of the object.
* void setColor(Color const &color)
  + Allows you to change the color of the object.
* std::string const &getSprite() const
  + Allows you to retrieve the sprite of the object.
* void setSprite(std::string const &sprite))
  + Allows to modify the sprite of the object.
* int const &getRotation() const
* Allows to retrieve the positions of the rotation of the object.
* void setRotation(int rotation)
  + Allows you to change the positions of the rotation of the object.
* Model3D const &getModel() const
  + Lets you retrieve the 3D model and this information.
* void setModel(Model3D const &model)
  + Allow you to modify the 3D object.

Private:

* Color \_color
  + Contains RGBA color values.
* Std::string \_sprite
  + Contains the sprite name.
* Model3D \_model
  + Contains model3D of object.
* Int \_rotation
* Contains the rotation positions of the object.

Texture.hpp (Allows to load a sprite or a 3D model or a color as well as to choose the rotation, stock its color, its sprite, its 3D model and / or its rotation)

Arcade.hpp

Methode :

* Arcade ()
  + Default constructor.
* ~Arcade ()
  + Default destructor.
* void Setup(std::string const &startingLib)
  + Load a Library.
* void Start()
  + Launch l’Arcade.
* void prevGraph() || void nextGraph()
  + Load the previous / next library.

* void prevGame() | void nextGame()
* Load the previous / next game
* void goUp() | void goDown() | void goLeft() | void goRight() | void pressEchap() | void shoot () || pressEight() | pressNine()

Use the keys: Up, Down, Left, Right, Esc, Space, Eight, New.

* void setInGameMode()
  + Allows to pass the arcade in game mode.
* void setMenuMode()
  + Allows to pass the arcade in menu mode.
* void initCurrentGame()
  + Initialize and launch the game.
* void setCurrentGameIndex(int gameIndex)
  + Change the ID of the current game to load another.
* void setCurrentGame(game\_ptr const &)
  + Load a new game.
* std::vector <game\_ptr> getLibGames();
  + Retrieves the libraries of all games and inserts them into a vector.
* std::vector<game\_ptr> getlibGame() const
* Retrieves the current game library.
* std::vector<std::string> getNameGame() const;
  + Retrieves the names of all games and inserts them into a vector.

Private:

* Arcade(const Arcade &obj)
  + Stock the Arcade.
* void FillGameVector()
  + Fill the game vector with the current games.
* void FillGraphVector(std::string const &startingLib)
  + Fill the library vector with the current librarys.
* bool \_isRunning
  + Boolean allowing to know the state of the arcade, True if the arcade is running, False if the arcade does not execute.
* int \_frameRate
  + Define the number of frame..
* IGraph\* \_currentGraph
  + Contains the currently running graphic library.
* IGame\* \_currentGame
  + Contains the game running..
* std::vector<lib\_ptr> \_libGraph
  + Vector containing all graphic libraries.
* std::vector<game\_ptr> \_libGame
  + Vector containing all graphic games.
* std::vector<std::string> \_nameGame
  + Vector containing all games names libraries.
* std::vector<ScoreManager> \_scoreVector
  + Vector containing all player’s score libraries.
* MenuController \*\_menu
  + Contains the menu.
* int \_graphIndex()
  + Variable containing the index of the current library.
* int \_gameIndex
  + Variable containing the index of the current game.
* void transfertData() const
  + Link the IData of the game and the graphic library.

Arcade->IArcadeBridge (Arcade allows you to load a game with a given graphic library, launch the game, change the library or game at any time, manage the move functions as well as the keys space, input, echap, 8 and 9 of the keyboard Status of the game (Menu or Game). Stock an IGraph of the used library, an IGame of the used game, a graphics library vector, a vector of the games, a game name vector, a score vector, the menu, a transfereData as well as the number of game and Library)

IArcadeBridge.hpp | INTERFACE

Methode :

* Virtual void prevGraph() || virtual void nextGraph()
  + Load the previous / next library.

* Virtual void prevGame() | virtual void nextGame()
  + Loads the previous / next game.
* Virtual void goUp() | Virtual void goDown() | Virtual void goLeft() | Virtual void goRight() | Virtual void pressEchap() | Virtual void shoot () | Virtual pressEight() | Virtual pressNine()
  + Allows you to use the keys: Up, Down, Left, Right, Esc, Space, Eight, New.

IGame.hpp | INTERFACE

Methode :

* Virtual ~IGame ()
  + Default destructor.
* Virtual void initGame()
  + Initialize and launch a selected game.
* virtual std::string const &getGameName() const = 0
* Retrieves the name of the game
* virtual int getFrameRatePerSecond() const = 0
  + Retrieves the number of frames per second.
* virtual std::vector <AData \*> const &getData() const = 0
  + Allows you to retrieve and store data in a vector.
* virtual std::vector <std::string> const & getSprite() const = 0
  + Retrieve sprites and store them in a vector.
* void goUp() | void goDown() | void goLeft() | void goRight() | void pressEchap() | void shoot () || pressEight() | pressNine()
  + Allows you to use the keys: Up, Down, Left, Right, Esc, Space, Eight, New.
* virtual void play() = 0
  + Launch the game.
* virtual void getMap() = 0
* Get the map.

virtual void whereIAm() = 0

* Gives player position on map.

IGraph.hpp | INTERFACE

Methode :

* Virtual ~IGraph ()
  + Default destructor.
* Virtual void initLib()
  + Initialize and launch a selected library.
* virtual std::string const &getLibName() const = 0
* Retrieves the name of the library.
* virtual void giveData(std::vector <AData \*> const &data) = 0
  + Allows to retrieve AData in a vector.
* virtual void giveSprite(std::vector <std::string> const &spriteList) = 0
  + Retrieve sprites stored in a vector.
* virtual void setBridge(IArcadeBridge \* bridge) = 0
  + Allows to link graphic lib to the game.
* virtual void handleData(AData const & data) = 0 | virtual void handleSphere(AData const & data) = 0 |virtual void handleCube(AData const & data) = 0 | virtual void handleCamera(AData const & data) = 0 |virtual void handleLight(AData const & data) = 0 |virtual void handleMusic(AData const & data) = 0 | virtual void handleText(AData const & data) = 0
  + sorting function AData.
* Virtual void prevGraph() = 0 || virtual void nextGraph() = 0
  + Load the previous / next library.

* Virtual void prevGame() = 0| virtual void nextGame() = 0
  + Loads the previous / next game.
* Virtual void goUp() = 0 | virtual void goDown() = 0 |virtual void goLeft() = 0| virtual void goRight() = 0 |virtual void pressEchap() = 0 | virtual void shoot () = 0 || virtual pressEight() = 0 | virtual pressNine() = 0
  + Allows you to use the keys: Up, Down, Left, Right, Esc, Space, Eight, New.
* virtual void toggleRunning() const = 0
  + Enable or disable (or pause) the game.

LoadController.hpp | INTERFACE

Methode :

* ~LoadController ()
  + Default destructor.
* void LoadController()
  + Default constructor.
* static std::vector <std::string> readDirectory(std::string const &path, std::string const &startingLib)
  + Allows you to read the folder and store all the libraries in a vector.