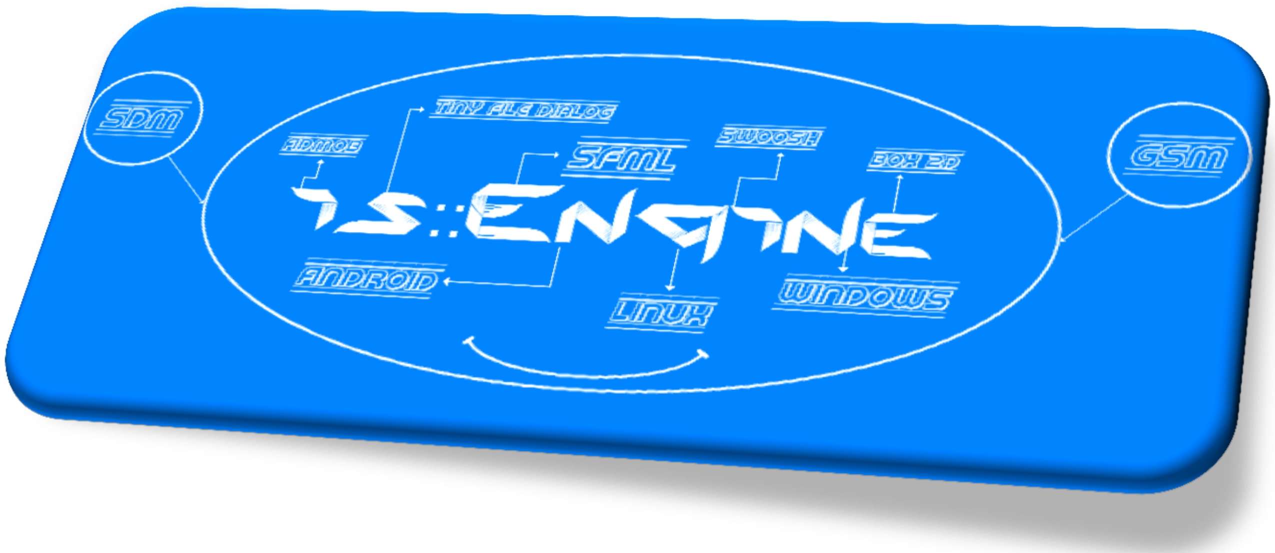
is::Engine v2.1

User Guide



Content

1. Introduction

Welcome to the **is::Engine v2.1** game engine user guide. The purpose of this guide is to detail how the API works. This is not a tutorial even if there is an example that shows you how to use the engine to create a game.

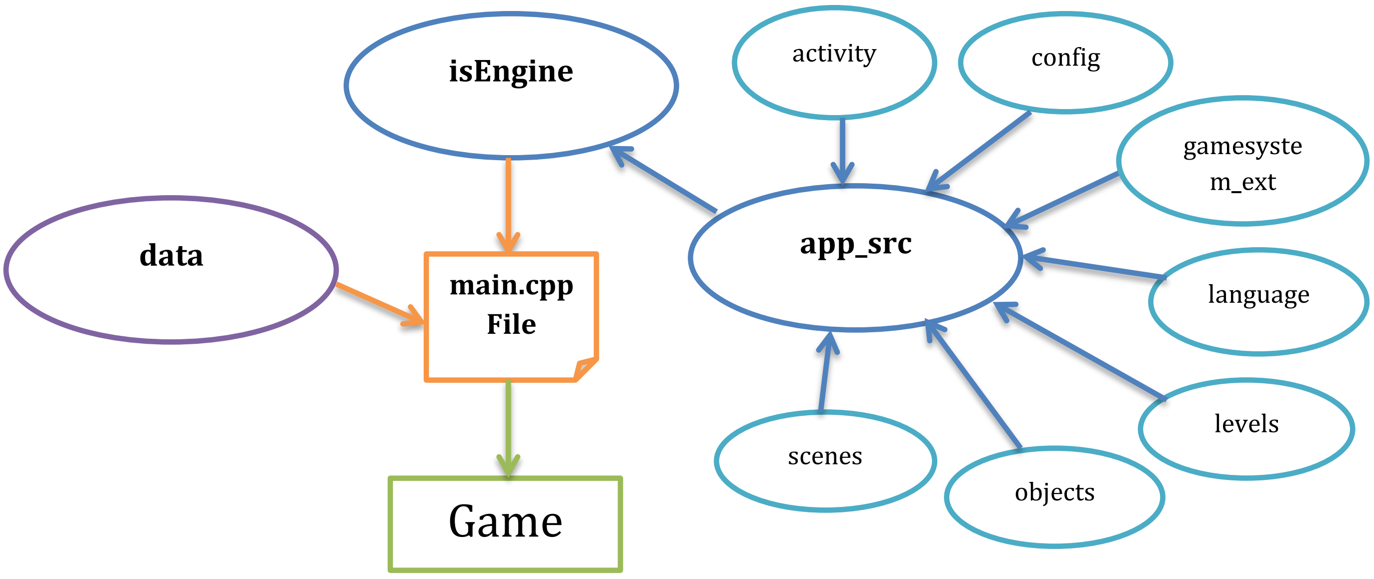
2. About the engine

is::Engine is a tool that relies on the mechanisms of the SFML library to work. So if you want to use this tool it is strongly advised to know at least the basics of SFML. The objective of this engine is to offer you features that allow you to create a game with the most flexibility possible and to easily carry it on various platforms (Windows, Linux, Android).

The engine is directly delivered with an IDE to avoid reconfigurations and to start quickly with this one. Note that each IDE with which the engine is delivered makes it possible to carry your project on a target platform. So the Android Studio project lets you use the engine to develop on Android.

The header which gives access to the engine is: ***isEngine/core/GameEngine.h.***

3. Engine structure



3.1 app_src

Directory that contains the source code of the game.

Description of these subdirectories:

- **activity**: Contains the **Activity** class ***(click here for more information: 1)*** which launches the different scenes of the game and ensures their interactions.
- **config**: Contains the **GameConfig.cpp** file ***(click here for more information: 1)*** which allows you to define the general parameters of the game.
- **gamesystem_ext**: Contains a class derived ***(click here for more information: 1)*** from **GameSystem *(click here for more information: 1)*** which allows you to manipulate the game data (save, load, etc.).
- **language**: Contains the **GameLanguage.cpp** file ***(click here for more information: 1)*** which allows you to manage everything related to the languages of the game.
- **levels**: Contains the levels and the **Level.h** file ***(click here for more information: 1)*** which allows you to integrate them into the game.
- **objects**: Contains the objects that will be used in the different scenes.

- **scenes**: Contains the different scenes of the game ***(click here for more information: 1)*** (Introduction, Main menu, ...).

3.2 isEngine

Directory that contains the source code of the game engine.

3.3 data

Directory of game resource files (music, sound effects, images, ...).

3.4 main.cpp file

This file contains the function which allows to launch the program.

3.4.1 main

int main()

Source Code

The main function that launches the game engine. Inside you will find **GameEngine game;** which initializes the game engine.

Return 0 when the program is finished and another value if there is an error during execution.

3.4.2 game.play

game.play()

Source Code

Allows to use the main loop of the game engine which allows you to launch the different game scenes (Introduction, Main menu, ...).

3.4.3 game.basicSFMLmain

game.basicSFMLmain()

Source Code

Displays a classic SFML window. This function allows you to use your own rendering loop with the engine. Very useful if you want to use an SFML project already under development with the engine or to integrate your own components into the engine.

Display

1. class GameDisplay

class GameDisplay;

Header : ***isEngine/system/display/GameDisplay.h***

Source Code

Abstract class that allows you to create the scene of a game. A scene is a place where the objects of the game come to life (Main Menu, Level, etc.). This class offers you functions that allow you to easily manipulate the view, apply window events on the scene, make animations on texts and sprites, display dialog boxes, etc.

2. Public methods

2.1 GameDisplay

GameDisplay(sf::RenderWindow &window, sf::View &view, sf::RenderTexture &surface, GameSystemExtended &gameSysExt, sf::Color bgColor)

Source Code

Constructor which allows you to create a GameDisplay object, it takes as parameter the window of the application, the surface of the SWOOSH library which allows to make transition effects, GameSystemExtended ***(click here for more information:* 1*)*** and the background color of the scene.

2.2 setAdmob

virtual void setAdmob(AdmobManager *admob)

Source Code

Allows you to integrate the Ad Manager (Admob) into a scene.

2.3 rewardVideoStep

virtual int rewardVideoStep()

Source Code

Allows you to launch a reward video ad.

Return 1 if the reward video is launched and 0 if there is an error (often occurs when the ad request did not work).

2.4 step

virtual void step() = 0

Source Code

Method which makes it possible to implement the part where the objects of the scene are updated (displacement of the objects, detection of collision, etc).

***Note:* When the SDM is activated and the user does not overload this function the SDM takes care of calling this method to automatically update the objects of the scene and the events of the window.**

2.5 draw

virtual void draw() = 0

Source Code

Method which makes it possible to implement the part where the objects of the scene will be draw.

***Note:* When the SDM is activated and the user does not overload this function the SDM takes care of calling this method to automatically draw the objects of the scene.**

2.6 drawScreen

virtual void drawScreen()

Source Code

Method for implementing the part where the objects of the game scene will be draw.

2.7 showTempLoading

virtual void showTempLoading(float time = 3.f * 59.f)

Source Code

Displays a false loading screen (Useful for making transitions in the same scene).

Parameter **time** represents the duration (in milliseconds) of the loading.

2.8 setOptionIndex

- ***First form :***

void setOptionIndex(int optionIndexValue, bool callWhenClick, float buttonScale = 1.3f)

Source Code

Allows to make animations on texts and play a sound when you change an option.

- ***Second form :***

void setOptionIndex(int optionIndexValue)

Source Code

Allows to change an option.

2.9 setTextAnimation

- ***First form :***

void setTextAnimation(sf::Text &txt, sf::Sprite &spr, int val)

Source Code

Allows to make an animation on a text and a sprite according to the choice of an option.

- ***Second form :***

void setTextAnimation(sf::Text &txt, int &var, int val)

Source Code

Allows to make an animation on a text according to the choice of an option.

2.10 setView

void setView()

Source Code

Updates the position of the view in the scene.

2.11 loadParentResources

virtual bool loadParentResources()

Source Code

Loads the resources that allow you to display dialog boxes in a scene.

It is generally used in the **loadResources** function of a scene.

2.12 loadResources

virtual bool loadResources() = 0

Source Code

Allows you to implement the loading of resources that are used in a scene.

2.13 isRunning

virtual bool isRunning() const

Source Code

Return **true** if a scene is running and **false** if not.

2.14 getView

virtual sf::View& getView() const

Source Code

Return the view of a scene.

2.15 getRenderWindow

virtual sf::RenderWindow& getRenderWindow()

Source Code

Return the scene execution window.

2.16 getRenderTexture

virtual sf::RenderTexture& getRenderTexture() const

Source Code

Return the surface on which we draw the objects of a scene.

2.17 getGameSystem

virtual GameSystemExtended& getGameSystem()

Source Code

Return the game system extended object.

2.18 getDeltaTime

float getDeltaTime()

Source Code

Return the execution time in seconds.

2.19 getDELTA_TIME

float getDELTA_TIME() const

Source Code

Return the variable DELTA_TIME.

2.20 getViewX

virtual float getViewX() const

Source Code

Return the x position of the view.

2.21 getViewY

virtual float getViewY() const

Source Code

Return the y position of the view.

2.22 getViewW

virtual float getViewW() const

Source Code

Return the width of the view.

2.23 getViewH

virtual float getViewH() const

Source Code

Return the height of the view.

2.24 getBgColor

virtual sf::Color& getBgColor()

Source Code

Return the background color of the scene.

2.25 inViewRec

virtual bool inViewRec(MainObject *obj, bool useTexRec = **true**)

Code Source

Return **true** if the object is in the field of vision of the view, **false** if not.

2.26 mouseCollision

- ***<u>First form</u>***

template <**class** T>

bool mouseCollision(T const &obj

#if defined(__ANDROID__)

, unsigned int finger = 0

#endif

)

Source Code

Windows, Linux : Detects if the mouse cursor collides with an object in the scene.

Android : Detects if the user touches an object in the scene.

Parameter:

> **obj** the object with which we want to test.
>
> **finger** represents the finger.

Return true if there is a collision and **false** if not.

Example :

```
if (mouseCollision(sprite))
{
   // do something
}
```

- ***Second form***

template <**class** T>

bool mouseCollision(T const &obj, sf::RectangleShape &cursor

```
                    #if defined(__ANDROID__)

                    , unsigned int finger = 0

                    #endif

                    )
```

Source Code

Windows, Linux: Detects if the mouse cursor collides with an object in the scene.

Android: Detects if the user touches an object in the scene.

Parameter:

> **obj** the object with which we want to test.
>
> **cursor** allows to recover the position of the collision point.
>
> **finger** represents the finger.

Return true if there is a collision and **false** if not.

Example :

```
sf::ReactangleShape rec;
if (mouseCollision(sprite, rec))
{
   float cursorXPosition = rec.getPosition.x();
   float cursorYPosition = rec.getPosition.y();
}
```

2.27 SDMstep

virtual void SDMstep()

Source Code

Allows to update the objects that are in the SDM container.

2.28 SDMdraw

virtual void SDMdraw()

Source Code

Allows to draw the objects in the SDM container.

2.29 GSMplaySound

virtual void GSMplaySound(std::string name)

Source Code

Allows to play a sound managed by GSM.

2.30 GSMpauseSound

virtual void GSMpauseSound(std::string name)

Source Code

Allows to pause a sound managed by GSM.

2.31 GSMplayMusic

virtual void GSMplayMusic(std::string name)

Source Code

Allows to play a music managed by GSM.

2.32 GSMpauseMusic

virtual void GSMpauseMusic(std::string name)

Source Code

Allows to pause a music managed by GSM.

3. Protected elements

3.1 enum MsgAnswer

enum MsgAnswer;

Enumerator	
YES	Response Yes
NO	Response No

Source Code

Represents the responses that the user can choose from the dialog box.

3.2 controlEventFocusClosing

void controlEventFocusClosing()

Source Code

Handles focus and window closing events. ***Used in an event loop!***

3.3 showMessageBox

template<**class** T>

void showMessageBox(T const &msgBody, bool mbYesNo = **true**)

Source Code

Define parameter and displays the dialog box.

Parameter:

> **msgBody** the message that will be displayed to the user.
>
> **mbYesNo true** displays a YES NO dialog box and **false** displays just an OK button.

3.4 updateMsgBox

void updateMsgBox(float const &DELTA_TIME, sf::Color textDefaultColor = sf::Color::White, sf::Color textSelectedColor = sf::Color::Red)

Source Code

Updates the information in the dialog box.

Parameter :

> **textDefaultColor** message text color.
>
> **textSelectedColor** button text color.

3.5 updateTimeWait

void updateTimeWait(float const &DELTA_TIME)

Source Code

Updates the counter which allows the user to wait after choosing an option. This avoids the choices in loops.

3.6 drawMsgBox

void drawMsgBox()

Source Code

Displays the dialog box.

SDM (Step and Draw Manager)

1. class SDM

class SDM;

Header : ***isEngine/system/display/SDM.h***

Source Code

Parent class that allows a scene to use the functions that automatically update and display the objects of a scene. It also allows to manage the display depth of objects.

2. Publics elements of SDM

2.1 m_SDMsceneObjects

```
std::vector<std::shared_ptr<MainObject>> m_SDMsceneObjects
```

Source Code

Container which allows to store the objects (derived from **MainObject** class) of the scene which will be managed by the SDM.

2.2 SDMgetObject

```
MainObject* SDMgetObject(std::string name)
```

Source Code

Return an object which is in the container according to its name.

<u>Exemple :</u>

```
auto player = SDMgetObject("Player");
player->setX(777.f);
```

2.3 SDMaddSceneObject

```
template <class T>

void SDMaddSceneObject(std::shared_ptr<T> obj, bool callStepFunction = true, bool callDrawFunction = true,
std::string name = "null")
```

Source Code

Allows to add an object to the container.

Parameter :

obj the object to add.

callStepFunction lets know if the SDM should update the object.

callDrawFunction lets know if the SDM should draw the object.

name allows to give a name to the object during the addition.

2.4 SDMaddSprite

```
virtual void SDMaddSprite(sf::Sprite &spr, std::string name, int depth = DepthObject::NORMAL_DEPTH)
```

Source Code

Allows you to add a SFML Sprite in the container. It will not be part of the objects to be updated but of those that will be displayed. The Sprite will be associated with a **MainObject** object.

2.5 SDMsetObjDepth

```
virtual void SDMsetObjDepth(std::string name, int depth)
```

Code Source

Allows to define the display depth of an object.

Game Sound

1. class GameSound

class GameSound;

Header : ***isEngine/system/sound/GameSound.h***

Source Code

Class that allows to use sounds in the game.

2. Publics elements of GameSound

2.1 GameSound

GameSound(**std::string** soundName, **std::string** filePath)

Source Code

Constructor that allows to load a sound and give it a name.

2.2 loadResources

void loadResources(**std::string** filePath)

Source Code

Allows to load the sound.

2.3 getSoundBuffer

sf::SoundBuffer& getSoundBuffer()

Source Code

Return Sound Buffer object.

2.4 getSound

sf::Sound& getSound()

Source Code

Return Sound object.

Game Music

1. class GameMusic

class GameMusic;

Header : ***isEngine/system/sound/GameMusic.h***

Source Code

Class that allows to use musics in the game.

2. Publics elements of GameMusic

2.1 GameMusic

GameMusic(**std::string** musicName, **std::string** filePath)

Source Code

Constructor that allows to load a music and give it a name.

2.2 loadResources

void loadResources(**std::string** filePath)

Source Code

Allows to load the music.

2.3 getMusic

sf::Music& getMusic()

Source Code

Return Music object.

GSM (Game Sound System)

1. class GSM

class GSM;

<u>*Header :*</u> ***isEngine/system/sound/GSM.h***

Source Code

Parent class that allows a scene to add and use sounds / musics without initializing SFML objects.

2. Publics elements of GSM

2.1 GSM Containers

std::vector<**std::shared_ptr**<GameSound>> m_GSMsound

std::vector<**std::shared_ptr**<GameMusic>> m_GSMmusic

Source Code

Container which allows to store the sounds / musics of the scene which will be managed by the GSM.

2.2 GSMaddSound

virtual void GSMaddSound(**std::string** filePath, **std::string** name)

Source Code

Allows to add a sound to the container.

Parameter :

name sound name.

filePath sound file path.

2.3 GSMaddMusic

virtual void GSMaddMusic(**std::string** filePath, **std::string** name)

Source Code

Allows to add a music to the container.

Parameter :

> **name** music name.
>
> **filePath** music file path.

2.4 GSMgetSound

virtual sf::Sound* GSMgetSound(**std::string** name)

Source Code

Return a sound that is in the container according to its name.

2.5 GSMgetMusic

virtual sf::Music* GSMgetMusic(**std::string** name)

Source Code

Return a music that is in the container according to its name.

Entities

1. class MainObject

class MainObject;

<u>*Header:*</u> ***isEngine/system/entity/MainObject.h***

Source Code

Basic class to create the objects (Character, Tiles, Button, etc) that will be used in the scenes. It offers you functions which allow you to control an object (displacements, detections of collision between objects, calculation of distance, etc) and many other things which are linked to the game play of the game.

2. Publics elements of MainObjet

2.1 MainObject

- ***<u>First form</u>***

MainObject()

Source Code

Default constructor of the class.

- ***<u>Second form</u>***

MainObject(float x, float y)

Source Code

Constructor that initializes the object with a starting point.

- ***Third form***

MainObject(sf::Sprite &spr, float x = 0.f, float y = 0.f)

Source Code

Constructor that initializes the object with a Sprite and a starting point.

2.2 instanceNumber

static int instanceNumber;

Source Code

Return the number of instances of the class.

2.3 m_SDMcallStep

bool m_SDMcallStep

Source Code

Lets know if SDM can use the object's **step()** (update) method.

2.4 m_SDMcallDraw

bool m_SDMcallDraw

Source Code

Lets know if SDM can use the object's **draw()** method.

2.5 setXStart

virtual void setXStart(float x)

Source Code

Defines the starting position x.

2.6 setYStart

virtual void setYStart(float y)

Source Code

Defines the starting position y.

2.7 setXPrevious

virtual void setXPrevious(float x)

Source Code

Defines the previous position x.

2.8 setYPrevious

virtual void setYPrevious(float y)

Source Code

Defines the previous position y.

2.9 setStartPosition

virtual void setStartPosition(float x, float y)

Source Code

Sets the x and y start position.

2.10 setX

virtual void setX(float x)

Source Code

Define position x.

2.11 setY

virtual void setY(float y)

Source Code

Define position y.

2.12 moveX

virtual void moveX(float x)

Source Code

Moves the object on the x-axis.

2.13 moveY

virtual void moveY(float y)

Source Code

Moves the object on the y-axis.

2.14 setPosition

virtual void setPosition(float x, float y)

Source Code

Set the x and y position.

2.15 setSpriteScale

virtual void setSpriteScale(float x, float y)

Source Code

Set the x and y scale of the object sprite.

2.16 setSpeed

virtual void setSpeed(float val)

Source Code

Set the speed of the object.

2.17 setHsp

virtual void setHsp(float val)

Source Code

Set horizontal speed.

2.18 setVsp

virtual void setVsp(float val)

Source Code

Set vertical speed.

2.19 setAngularMove

virtual void setAngularMove(float const &DELTA_TIME, float speed, float angle)

Source Code

Allows to move the object according to an angle and a speed.

2.20 setImageXscale

virtual void setImageXscale(float val)

Source Code

Set the x scale of the object.

2.21 setImageYscale

virtual void setImageYscale(float val)

Source Code

Set the y scale of the object.

2.22 setImageScale

virtual void setImageScale(float val)

Source Code

Set the x and y scale of the object with the same value.

2.23 setImageAngle

virtual void setImageAngle(float val)

Source Code

Set the angle of the object.

2.24 setXOffset

virtual void setXOffset(float val)

Source Code

Define the x offset of the object.

2.25 setYOffset

virtual void setYOffset(float val)

Source Code

Define the y offset of the object.

2.26 setXYOffset

virtual void setXYOffset()

Source Code

Defines the offset x and y of the object with the same value.

2.27 setTime

void setTime(float x)

Source Code

Set the value of the object's **m_time** variable.

2.28 setImageAlpha

virtual void setImageAlpha(int val)

Source Code

Set the alpha image of the object.

2.29 setImageIndex

virtual void setImageIndex(int val)

Source Code

Define the sub image of the object.

2.30 setMaskW

virtual void setMaskW(int val)

Source Code

Set the width of the object's collision mask.

2.31 setMaskH

virtual void setMaskH(int val)

Source Code

Set the height of the object's collision mask.

2.32 setIsActive

virtual void setIsActive(bool val)

Source Code

Defines the activity state of the object.

2.33 updateCollisionMask

- ***First form :***

virtual void updateCollisionMask()

Source Code

Updates the information (size, position, etc) of the collision mask.

- ***Second form:***

virtual void updateCollisionMask(int x, int y)

Source Code

Updates the position of the collision mask according to a point x and y different from that of the object.

2.34 centerCollisionMask

virtual void centerCollisionMask(int x, int y)

Source Code

Center the position of the collision mask according to a point x and y.

2.35 updateSprite

- ***First form***

virtual void updateSprite()

Source Code

Updates the sprite of the object with the values of the variables (alpha, scale, etc.) which are in the object.

- ***Second form***

virtual void updateSprite(float x, float y, float angle = 0.f, int alpha = 255, float xScale = 1.f, float yScale = 1.f)

Source Code

Updates the sprite of the object with external values.

2.36 draw

virtual void draw(sf::RenderTexture &surface)

Source Code

Displays the object.

2.37 getMask

virtual is::Rectangle getMask() const

Source Code

Return the collision mask.

2.38 getX

virtual float getX() const

Source Code

Return the x position of the object.

2.39 getY

virtual float getY() const

Source Code

Return the y position of the object.

2.40 getXStart

virtual float getXStart() const

Source Code

Return the x start position of the object.

2.41 getYStart

virtual float getYStart() const

Source Code

Return the y start position of the object.

2.42 getXPrevious

virtual float getXPrevious() const

Source Code

Return the previous position x of the object.

2.43 getYPrevious

virtual float getYPrevious() const

Source Code

Return the previous position y of the object.

2.44 distantToPoint

virtual float distantToPoint(float x, float y) const

Source Code

Return the distance between the object and a point x and y.

2.45 distantToObject

virtual float distantToObject(std::shared_ptr<MainObject> const &other, bool useSpritePosition) const

Source Code

Return the distance between the object and another.

Parameter if **useSpritePosition** is **true** we use the position of the sprite of the object to do the test **if not** we use the position x, y of the object.

2.46 pointDirection

- ***First form***

virtual float pointDirection(float x, float y) const

Source Code

Return the direction (angle) of the object relative to a point.

- ***Second form***

virtual float pointDirection(**std::shared_ptr**<MainObject> const &other) const

Source Code

Return the direction (angle) of the object relative to another. Here the other object is a smart pointer.

2.47 pointDirectionSprite

- ***First form***

virtual float pointDirectionSprite(float x, float y) const

Source Code

Return the direction (angle) of the object's sprite relative to a point.

- ***Second form***

virtual float pointDirectionSprite(**std::shared_ptr**<MainObject> const &other) const

Source Code

Return the direction (angle) of the object's sprite relative to another.

2.48 getSpeed

virtual float getSpeed() const

Source Code

Return object speed.

2.49 getHsp

virtual float getHsp() const

Source Code

Return the horizontal speed of the object.

2.50 getVsp

virtual float getVsp() const

Source Code

Return the vertical speed of the object

2.51 getFrame

virtual float getFrame() const

Source Code

Return the number of the sub-image that is being displayed.

2.52 getFrameStart

virtual float getFrameStart() const

Source Code

Return the number of the start sub picture.

2.53 getFrameEnd

virtual float getFrameEnd() const

Source Code

Return the number of the end sub picture.

2.54 getImageXscale

virtual float getImageXscale() const

Source Code

Return the object's x-scale.

2.55 getImageYscale

virtual float getImageYscale() const

Source Code

Return the object's y-scale.

2.56 getImageScale

virtual float getImageScale() const

Source Code

Return the object's scale.

2.57 getImageAngle

virtual float getImageAngle() const

Source Code

Return the angle of the object image.

2.58 getXOffset

virtual float getXOffset() const

Source Code

Return the object x offset.

2.59 getYOffset

virtual float getYOffset() const

Source Code

Return the object y offset.

2.60 getTime

virtual float getTime() const

Source Code

Return the value of the variable **m_time**.

2.61 getInstanceId

virtual int getInstanceId() const

Source Code

Return the object number.

2.62 getMaskWidth

virtual int getMaskWidth() const

Source Code

Return the width of the collision mask.

2.63 getMaskHeight

virtual int getSpriteHeight() const

Source Code

Return the height of the collision mask.

2.64 getIsActive

virtual bool getIsActive() const

Source Code

Return the state of the object.

2.65 getImageAlpha

virtual int getImageAlpha() const

Source Code

Return the alpha image of the object.

2.66 getImageIndex

virtual int getImageIndex() const

Source Code

Return the image index.

2.67 getSpriteWidth

virtual int getSpriteWidth() const

Source Code

Return the width of the sprite.

2.68 getSpriteHeight

virtual int getSpriteHeight() const

Source Code

Return the height of the sprite.

2.69 getSpriteX

virtual float getSpriteX() const

Source Code

Return the x position of the sprite.

2.70 getSpriteY

virtual float getSpriteY() const

Source Code

Return the y position of the sprite.

2.71 getSpriteCenterX

virtual int getSpriteCenterX() const

Source Code

Return the x center of the sprite.

2.72 getSpriteCenterY

virtual int getSpriteCenterY() const

Source Code

Return the y center of the sprite.

2.73 placeMetting

- ***<u>First form</u>***

virtual bool placeMetting(int x, int y, MainObject const *other)

Source Code

Return **true** if there is a collision with another object, **false** if not.

- ***<u>Second form</u>***

virtual bool placeMetting(int x, int y, **std::shared_ptr**<MainObject> const &other)

Source Code

Return **true** if there is a collision with another object, **false** if not. Here the other object is a smart pointer.

2.74 getSprite

virtual sf::Sprite& getSprite()

Source Code

Return the object sprite.

2.75 setFrame

virtual void setFrame(float frameStart, float frameEnd = -1.f)

Source Code

Defines the start and end image which will be used to animate the sprite of the object.

3. Other functions of MainObject

3.1 instanceExist

- ***First form***

template<**class** T>

bool instanceExist(**std::shared_ptr**<T> const &obj)

Source Code

Return **true** if the instance exists, **false** if not.

- ***Second form***

template<**class** T>

bool instanceExist(T const *obj)

Source Code

Return **true** if the instance exists, **false** if not.

3.2 operator()

- ***Position comparator***

class CompareX;

Source Code

Functor which is used to sort the objects compared to their position x.

bool **operator**()(**std::shared_ptr**<MainObject> const &a, **std::shared_ptr**<MainObject> const &b) const

Source Code

Used to sort objects according to their x positions.

- ***Depth comparator***

class CompareDepth;

Source Code

Functor which is used to sort objects according to their depth.

bool **operator()**(**std::shared_ptr**<MainObject> const &a, **std::shared_ptr**<MainObject> const &b) const

Source Code

Used to sort objects according to their depths.

3.3 sortObjArrayByX

template<class T>

void sortObjArrayByX(std::vector<std::shared_ptr<T>> &v)

Source Code

Sort an array (**std::vector**) of objects by x position.

3.4 sortObjArrayByDepth

template<class T>

void sortObjArrayByDepth(std::vector<std::shared_ptr<T>> &v)

Source Code

Sort an array (**std::vector**) of objects by depth.

3.5 operator>

bool operator<(std::shared_ptr<MainObject> const &a, const MainObject &b)

Source Code

Return true if the position of object A is greater than that of B, **false** if not.

3.6 operator<

bool operator<(const MainObject &b, std::shared_ptr<MainObject> const &a)

Source Code

Return true if the position of object A is less than that of B, **false** if not.

Forms for collision masks

Header : ***isEngine/system/entity/Form.h***

1. class Rectangle

class Rectangle;

Source Code

Represents the rectangle collision mask. These members **m_left, m_top, m_right, m_bottom** allow to define the size of the mask.

2. class Point

class Point;

Source Code

Represents the point collision mask. These members **m_x, m_y** allow to define the position of the point.

- ***First form***

Point()

Source Code

Default constructor.

- ***Second form***

Point(float x, float y)

Source Code

Constructor used to define the position of the point.

3. class Line

class Line;

Source Code

Represents the line collision mask. These members **m_x1, m_x2, m_y1, m_y2** allow to define the length of the line.

- ***First form***

Line()

Source Code

Default constructor.

- ***Second form***

Line(float x1, float y1, float x2, float y2)

Source Code

Constructor used to define the length of the line.

The Parent Classes of MainObject

1. class DepthObject

class DepthObject;

Header : ***isEngine/system/entity/parents/DephObject.h***

Source Code

Class that provides methods for managing the display depth of objects in a scene.

1.1 enum Depth

enum Depth;

Enumerator	
VERY_BIG_DEPTH	Very big depth
BIG_DEPTH	Big depth
NORMAL_DEPTH	Normal depth
SMALL_DEPTH	Small depth
VERY_SMALL_DEPTH	Very small depth

Source Code

Represents the depth level of an object.

1.2 DepthObject

DepthObject(int Depth)

Source Code

Constructor to define a depth.

1.3 setDepth

virtual void setDepth(int val)

Source Code

Set the depth of the object.

1.4 getDepth

virtual int getDepth() const

Source Code

Return the depth of the object.

2. class Destructible

class Destructible;

<u>*Header :*</u> ***isEngine/system/entity/parents/Destructible.h***

Source Code

Class that offers methods to manage the destruction of an object.

2.1 Destructible

Destructible()

Source Code

Default constructor.

2.2 setDestroyed

virtual void setDestroyed()

Source Code

Starts the destruction of an object.

2.3 isDestroyed

virtual bool isDestroyed() const

Source Code

Return the state of the object.

3. class Visibility

class Visibility;

Header : ***isEngine/system/entity/parents/Visibility.h***

Source Code

Class that offers methods to manage the visibility of an object.

3.1 Visibility

explicit Visibility(bool defaultVisibility = **true**)

Source Code

Class constructor.

3.2 setVisible

void setVisible(bool value)

Source Code

Set the visibility of the object.

3.3 getVisible

bool getVisible() const

Source Code

Return the state of the object.

4. class Health

class Health;

Header : ***isEngine/system/entity/parents/Health.h***

Source Code

Class that provides methods for managing the health of an object.

4.1 Health

- ***First form***

Health(int health)

Source Code

Constructor of the class takes as a parameter the health to be attributed to the object. Here the maximum health value is equal to the defined health.

- ***Second form***

Health(int health, int maxHealth)

Source Code

Class constructor takes as a parameter the health to be assigned to the object and the maximum value.

4.2 setHealth

virtual void setHealth(int val)

Source Code

Define the health of the object.

4.3 setMaxHealth

virtual void setMaxHealth(int val)

Source Code

Defines the maximum health (the limit not to be exceeded) of the object.

4.4 addHealth

virtual void addHealth(int val = 1)

Source Code

Add health to the object. Can also be used to retake it if you put a negative value.

4.5 getHealth

virtual int getHealth() const

Source Code

Return the health of the object.

4.6 getMaxHealth

virtual int getMaxHealth() const

Source Code

Return the maximum health (the limit not to be exceeded) of the object.

5. class HurtEffect

class HurtEffect;

Header : ***isEngine/system/entity/parents/HurtEffect.h***

Source Code

Class that offers methods to make an invulnerability effect on an object. That is, make the object blink for a certain time (e.g. when the player is attacked by an enemy he becomes invulnerable by blinking for a limited time).

5.1 HurtEffect

HurtEffect(sf::Sprite &sprParent) :

Source Code

Class constructor takes as parameter the sprite on which the invulnerability effect will be effected.

5.2 hurtStep

void hurtStep(float const &DELTA_TIME)

Source Code

Allows to make the invulnerability animation.

5.3 setIsHurt

void setIsHurt(float durration = 100.f)

Source Code

Defines the duration (in millisecond) of the object's invulnerability.

5.4 getIsHurt

bool getIsHurt() const

Source Code

Return **true** if the object is invulnerable, **false** if not.

6. class ScorePoint

class ScorePoint;

<u>*Header :*</u> ***isEngine/system/entity/parents/ScorePoint.h***

Source Code

Class that offers methods for managing the score to be assigned to an object (e.g. each enemy has a particular score point when it is created which is added to the player's overall score when he is defeated).

6.1 ScorePoint

explicit ScorePoint(int point = 0)

Source Code

Class constructor, takes as a parameter the point to assign to the object.

6.2 setScorePoint

virtual void setScorePoint(int point)

Source Code

Set object score point.

6.3 getScorePoint

virtual int getScorePoint() const

Source Code

Return the score point assigned to the object.

7. class Step

class Step;

<u>*Header :*</u> ***isEngine/system/entity/parents/Step.h***

Class that offers methods to manage the different steps of an object (e.g. to take off a rocket you have to go through several steps).

7.1 Step

explicit Step(int step = 0)

Source Code

Class constructor.

7.2 setStep

virtual void setStep(int val)

Source Code

Defines the step of the object.

7.3 addStep

virtual void addStep()

Source Code

Advance the object step.

7.4 reduceStep

virtual void reduceStep()

Source Code

Reduce the object step.

7.5 getStep

virtual int getStep() const

Source Code

Return the step at which the object is.

8. class Name

class Name;

Header : ***isEngine/system/entity/parents/Name.h***

Parent class that provides methods for managing the name of an object.

8.1 Name

explicit Name(**std::string** name = **""**)

Source Code

Constructor used to define the name of the object.

8.2 setName

void setName(**std::string** soundName)

Source Code

Allows to define the name of the object.

8.3 getName

std::string getName()

Source Code

Return the name of the object.

9. class FilePath

class FilePath;

<u>*Header :*</u> ***isEngine/system/entity/parents/FilePath.h***

Parent class that provides methods for managing the path of a file.

9.1 FilePath

FilePath(**std::string** filePath)

Source Code

Constructor of the class, it takes as parameter the path of the file to load.

9.2 setFilePath

void setFilePath(**std::string** filePath)

Source Code

Allows to define the file path.

9.3 getFilePath

std::string getFilePath()

Source Code

Return file path.

9.4 getFileIsLoaded

bool getFileIsLoaded()

Source Code

Return **true** when the file has been loaded **false** otherwise.

Admob

1. class AdmobManager

class AdmobManager;

<u>*Header:*</u> ***isEngine/system/android/AdmobManager.h***

Source Code

Class that allows you to use the Admob SDK in the game. It offers functions to manage banner and reward video ads.

2. Public methods

2.1 AdmobManager

AdmobManager(sf::RenderWindow &window, ANativeActivity* activity, JNIEnv* env, JavaVM* vm)

Source Code

Class constructor, it takes the window, Android activity and the virtual machine as parameters.

2.2 loadBannerAd

void loadBannerAd()

Source Code

Request for banner ad.

2.3 showBannerAd

void showBannerAd()

Source Code

Displays a banner ad provided the request has been successfully executed.

2.4 hideBannerAd

void hideBannerAd()

Source Code

Hide the banner ad.

2.5 loadRewardVideo

void loadRewardVideo()

Source Code

Request a reward video ad.

2.6 updateSFMLApp

auto updateSFMLApp(bool whiteColor)

Source Code

Updates the SFML application in the background when an ad is displayed. This avoids the main program crashing.

2.7 checkAdObjInit

void checkAdObjInit()

Source Code

Ensures the initialization of Admob components.

2.8 checkAdRewardObjReinitialize

void checkAdRewardObjReinitialize()

Source Code

Reset Admob components.

3. Other Functions of AdmobManager

3.1 ProcessEvents & WaitForFutureCompletion

static bool ProcessEvents(int msec)

static void WaitForFutureCompletion(firebase::FutureBase future)

Source Code

Ensures the proper functioning of tests on ad components.

3.2 checkAdState

static bool checkAdState(firebase::FutureBase future)

Source Code

Return **true** if the test on the ad component was successful, **false** if not.

Time

1. class GameTime

class GameTime;

Header: ***isEngine/system/function/GameTime.h***

Source Code

This Class allows you to manipulate the game time (the stopwatch). Very useful for platform games like Super Mario Bros or Sonic which uses a stopwatch in a level.

2. Public methods of GameTime

2.1 GameTime

- ***First form***

GameTime()

Source Code

Default constructor, initializes all counters (minute, second, millisecond) to zero (0).

- ***Second form***

GameTime(unsigned int ms)

Source Code

Constructor to initialize time with milliseconds which will be distributed later in minutes and seconds.

- ***Third form***

GameTime(unsigned int m, unsigned int s, unsigned int ms = 0)

Source Code

Constructor to initialize time with minutes, seconds and milliseconds.

2.2 step

void step(float const &DELTA_TIME, float const &VALUE_CONVERSION, float const &VALUE_TIME)

Source Code

Start the countdown timer so that it stops at zero (0).

2.3 addTimeValue

void addTimeValue(int m, int s = 0, int ms = 0)

Source Code

Add minutes, seconds and milliseconds to the current time.

2.4 setTimeValue

void setTimeValue(int m, int s = 0, int ms = 0)

Source Code

Set a new minute, second and millisecond for the current time.

2.5 setMSecond

void setMSecond(int ms)

Source Code

Set milliseconds which will be distributed in minutes and seconds.

2.6 getTimeString

std::string getTimeString() const

Source Code

Return current time as a string (example **00: 00.00**).

2.7 getTimeValue

unsigned int getTimeValue() const

Source Code

Return time in milliseconds.

2.8 getMinute

unsigned int getMinute() const

Source Code

Return the minute.

2.9 getSecond

unsigned int getSecond() const

Source Code

Return the second.

2.10 getMSecond

unsigned int getMSecond() const

Source Code

Return the millisecond.

2.11 operator=

GameTime& **operator**=(GameTime const &t)

Source Code

Equality operator to compare two objects.

2.12 operator<<

friend std::ostream& **operator**<<(**std::ostream** &flux, GameTime const &t)

Source Code

Operator to display the time with the **std::cout**.

3. Other functions of GameTime

bool **operator**==(GameTime const &t1, GameTime const &t2)

bool **operator**>(GameTime const &t1, GameTime const &t2)

bool **operator**<(GameTime const &t1, GameTime const &t2)

Source Code

These Operators allow you to make comparisons with objects.

Game control

class GameKeyData

<u>*Header:*</u> ***isEngine/system/function/GameKeyData.h***

Source Code

Class that allows to manage the controls of the game. It supports the keyboard and the mouse on PC and becomes a Virtual Game Pad on Android.

1. Elements of GameKeyData
2.1 enum VirtualKeyIndex

enum VirtualKeyIndex;

Enumerator	
V_KEY_LEFT	Represents the LEFT key
V_KEY_RIGHT	Represents the RIGHT key
V_KEY_UP	Represents the UP key
V_KEY_DOWN	Represents the DOWN key
V_KEY_A	Represents the A key
V_KEY_B	Represents the B key

V_KEY_NONE	No key

Source Code

Represents game controls key.

2.2 GameKeyData

GameKeyData(is::GameDisplay *scene)

Source Code

Constructor who takes the scene as a parameter.

2.3 loadResources

void loadResources(sf::Texture &tex)

Source Code

Allows to load the texture which will be used to create the keys of the Virtual Game Pad.

2.4 step

void step(float const &DELTA_TIME)

Source Code

Updates the position of the Virtual Game Pad on the screen and also detects the use of commands.

2.5 draw

void draw(sf::RenderTexture &surface)

Source Code

Displays the Virtual Game Pad.

2.6 m_keyPausePressed

bool m_keyPausePressed

Source Code

Determines if the pause key is pressed.

2.7 m_keyLeftPressed

bool m_keyLeftPressed

Source Code

Stores the state of the LEFT key.

2.8 m_keyRightPressed

bool m_keyRightPressed

Source Code

Stores the state of the RIGHT key.

2.9 m_keyUpPressed

bool m_keyUpPressed

Source Code

Stores the state of the UP key.

2.10 m_keyDownPressed

bool m_keyDownPressed

Source Code

Stores the state of the DOWN key.

2.11 m_keyAPressed

bool m_keyAPressed

Source Code

Stores the state of the A key.

2.12 m_keyBPressed

bool m_keyBPressed

Source Code

Stores the state of the B key.

2.13 m_keyAUsed

bool m_keyAUsed

Source Code

Stores the state of the A key when it is used.

2.14 m_keyBUsed

bool m_keyBUsed

Source Code

Stores the state of the B key when it is used.

2.15 m_disableAllKey

bool m_disableAllKey

Source Code

Disables all game controls.

2.16 m_hideGamePad

bool m_hideGamePad

Source Code

Allows to hide the Virtual Game Pad on Android.

2.17 m_keyboardA

sf::Keyboard::Key m_keyboardA

Source Code

Represents the keyboard key that serves as the A key.

2.18 m_keyboardB

sf::Keyboard::Key m_keyboardB

Source Code

Represents the keyboard key that serves as the B key.

2.19 m_keyboardLeft

sf::Keyboard::Key m_keyboardLeft

Source Code

Represents the keyboard key that serves as the LEFT key.

2.20 m_keyboardRight

sf::Keyboard::Key m_keyboardRight

Source Code

Represents the keyboard key that serves as the RIGHT key.

2.21 m_keyboardUp

sf::Keyboard::Key m_keyboardUp

Source Code

Represents the keyboard key that serves as the UP key.

2.22 m_keyboardDown

sf::Keyboard::Key m_keyboardDown

Source Code

Represents the keyboard key that serves as the DOWN key.

2.23 m_moveKeyPressed

VirtualKeyIndex m_moveKeyPressed

Source Code

Used to find out whether the virtual directional keys are pressed.

2.24 m_actionKeyPressed

VirtualKeyIndex m_actionKeyPressed

Source Code

Used to find out whether the virtual keys A, B are pressed.

2.25 keyLeftPressed

bool keyLeftPressed()

Source Code

Return **true** if the LEFT directional key is pressed, **false** if not.

2.26 keyRightPressed

bool keyRightPressed()

Source Code

Return **true** if the RIGHT directional key is pressed, **false** if not.

2.27 keyUpPressed

bool keyUpPressed()

Source Code

Return **true** if the UP directional key is pressed, **false** if not.

2.28 keyDownPressed

bool keyDownPressed()

Source Code

Return **true** if the DOWN directional key is pressed, **false** if not.

2.29 keyAPressed

bool keyAPressed()

Source Code

Return **true** if the key A is pressed, **false** if not.

2.30 keyBPressed

bool keyBPressed()

Source Code

Return **true** if the key B is pressed, **false** if not.

2.31 virtualKeyPressed

bool virtualKeyPressed(VirtualKeyIndex virtualKeyIndex)

Source Code

Return **true** if the corresponding virtual key is pressed, **false** if not.

2. Other functions of GameKeyData

These functions are found in **GameKeyName.h.**

Header : ***isEngine/system/function/GameKeyName.h***

- ***First form***

inline const char *getKeyName(const sf::Keyboard::Key key)

Source Code

Return the name of the keyboard key as a string.

- ***Second form***

inline std::wstring getKeyWName(const sf::Keyboard::Key key)

Source Code

Return the name of the keyboard key as a **std::wstring**.

Game System

1. class GameSystem

class GameSystem;

Header : ***isEngine/system/function/GameSystem.h***

Source Code

Base class which ensures the sharing of game information between the different components of the game engine. It contains the global variables and functions which ensure the proper functioning of the engine.

2. Elements of GameSystem

2.1 enum ValidationButton

enum ValidationButton;

Enemerator	
MOUSE	Represent the validation button of the mouse (if it is used, it becomes a touch action on Android)
KEYBOARD	Represent the validation key on the keyboard
ALL_BUTTONS	Represent the validation button of the mouse and the keyboard (if it is used, it becomes a touch action on Android)

Source Code

Represents the validation key on PC, It lets you know the button that will be used during a validation test.

2.2 GameSystem

GameSystem()

Source Code

Default constructor.

2.3 isPressed

bool isPressed(

#if defined(__ANDROID__)

int finger = 0

#else

ValidationButton validationButton = **ALL_BUTTONS**

#endif

) const

Source Code

- ***Windows, Linux:***

Checks if the validation key is pressed.

The validation key is defined in **GameConfig.h** ***(See here: 3.1)***.

- ***Android :***

Check if the screen is touched by the user.

Parameter :

finger finger index (on Android).

validationButton Represents the validation button to be used to perform the test.

Example:

- Check if the validation key of the keyboard is pressed, by default this key is **ENTER**.

```
if (m_gameSystem.isPressed(is::GameSystem::ValidationButton::KEYBOARD))
{
   // do something
}
```

- Check if the validation button of the mouse is pressed, by default this button is **LEFT**.

```
if (m_gameSystem.isPressed(is::GameSystem::ValidationButton::MOUSE)
{
   // do something
}
```

2.4 keyIsPressed

- ***First form***

bool keyIsPressed(sf::Keyboard::Key key) const

Source Code

Check if the keyboard key is pressed.

Return **true** if the key is pressed, **false** if not.

- ***Second form***

bool keyIsPressed(sf::Mouse::Button button) const

Source Code

Check if the mouse button is pressed.

Return **true** if the button is pressed, **false** if not.

2.5 fileExist

bool fileExist(std::string const &fileName) const

Source Code

Return **true** if the file exists, **false** if not.

2.6 playSound

template <**class** T>

void playSound(T &obj)

Source Code

Allows to play a sound / music if the option is activated.

2.7 stopSound

template <**class** T>

void stopSound(T &obj)

Source Code

Allows to stop a sound / music.

2.8 useVibrate

void useVibrate(short ms)

Source Code

Allows to use the vibrator if this option is activated (only for Android).

Parameter **ms** represents the duration of the vibrator in milliseconds.

2.9 saveConfig

void saveConfig(**std::string** const &fileName)

Source Code

Save game configuration data.

2.10 loadConfig

void loadConfig(**std::string** const &fileName)

Source Code

Load game configuration data.

2.11 savePadConfig

void savePadConfig(**std::string** const &fileName)

Source Code

Save the configuration data of the Virtual Game Pad.

2.12 loadPadConfig

void loadPadConfig(**std::string** const &fileName)

Source Code

Load the configuration data of the Virtual Game Pad.

2.13 m_disableKey

bool m_disableKey

Source Code

If it is **true** all the engine functions that manage the inputs (keyboard, mouse, touch) are disabled.

2.14 m_enableSound

bool m_enableSound

Source Code

Used to find out if the sound is activated.

2.15 m_enableMusic

bool m_enableMusic

Source Code

Used to find out if the music is activated.

2.16 m_enableVibrate

bool m_enableVibrate

Source Code

Used to find out if the vibrator is activated (only for Android).

2.17 m_keyIsPressed

bool m_keyIsPressed

Source Code

Used to find out if a key / button has been pressed.

2.18 m_firstLaunch

bool m_firstLaunch

Source Code

Check if the game has been launched at least once.

2.19 m_validationMouseKey

sf::Mouse::Button m_validationMouseKey

Source Code

Represent the variable that stores the validation button of the mouse.

2.20 m_validationKeyboardKey

sf::Keyboard::Key m_validationKeyboardKey

Source Code

Represents the variable that stores the keyboard validation key.

2.21 m_gameLanguage

int m_gameLanguage

Source Code

Represents the index of the chosen language.

2.22 m_padAlpha

int m_padAlpha

Source Code

Allows to modify the transparency of the Virtual Game Pad.

Game System Extended

1. class GameSystemExtended

class GameSystemExtended;

Header : ***app_src/gamesystem_ext/GameSystemExtended.h***

Source Code

Class derived from **GameSystem** ***(click here for more information: 1)***, it performs the same role as its parent. Its particularity is that it contains new elements which will be used to manage the game play and to manipulate the different game scenes.

2. Elements of GameSystemExtended
2.1 GameSystemExtended

GameSystemExtended()

Source Code

Default constructor.

2.2 initSystemData

void initSystemData()

Source Code

Initializes the data linked to the game engine.

2.3 initProgress

void initProgress()

Source Code

Initialize game progress data.

2.4 initData

void initData(bool clearCurrentLevel = **true**)

Source Code

Initializes the game play data (score, life, etc.).

2.5 saveData

void saveData(**std::string** const &fileName)

Source Code

Save game data.

2.6 loadData

void loadData(**std::string** const &fileName)

Source Code

Load game data.

2.7 m_launchOption

DisplayOption m_launchOption

Source Code

Determine the action ***(click here to see the actions: 1)*** that will be performed on the different scenes of the game.

2.8 game play variables

```
int  m_gameProgression
int  m_levelNumber
int  m_currentLevel
int  m_currentLives
int  m_currentBonus
int  m_currentScore
int  m_levelTime
```

Source Code

Global game variables.

Game Function

Header : ***isEngine/system/function/GameFunction.h***

These functions allow you to do conversions on strings, manipulate time, manipulate SFML objects, display special texts, use certain Android functions, perform geometric calculations, perform tests on variables, use functions to manipulate random values, etc.

1. General Function

1.1 VALUE_CONVERSION

static float const VALUE_CONVERSION(65.f);

Source Code

Acts on the timing of counters.

Example:

- This creates a counter in milliseconds when we put it in the update loop

```
// msCpt is an integer variable

msCpt += (is::VALUE_CONVERSION * 1.538f) * DELTA_TIME; // DELTA_TIME is the execution time returned by the machine
```

1.2 WITH

#define WITH(_SIZE)

Source Code

Allows to browse a vector array. **_I** is the counter.

Example:

```
WITH(vectoreArray.size())
{
  vectoreArray[_I]->function(...);
}
```

1.3 w_chart_tToStr

std::string w_chart_tToStr(wchar_t const *str)

Source Code

Convert **w_chart_t** to **std::string.**

1.4 strToWStr

std::wstring strToWStr(const **std::string** &str)

Source Code

Convert **std::string** to **std::wstring.**

1.5 numToStr

template <**class** T>

std::string numToStr(T val)

Source Code

Convert numeric to **std::string.**

1.6 strToNum

template <**typename** T>

T strToNum(const **std::string** &str)

Source Code

Convert **std::string** to numeric.

1.7 numToWStr

template <class T>

std::wstring numToWStr(T val)

Source Code

Convert numeric to **std::wstring.**

1.8 writeZero

template <class T>

std::string writeZero(T val, int zeroNumber = 1)

Source Code

Draw zeros in front of a number.

Parameter **zeroNumber** represents the number of zero to display.

Example:

```
int var(7);
std::cout << is::writeZero(var, 2) << std::endl; // its display in the console "007"
```

1.9 getMSecond

int getMSecond(float const &DELTA_TIME)

Source Code

Return execution time in milliseconds.

1.10 showLog

void showLog(std::string str)

Source Code

Displays messages in the console.

1.11 arraySize

template <size_t SIZE, class T>

inline size_t arraySize(T (&arr)[SIZE])

Source Code

Return the size of an array.

1.12 choose

template <typename T>

T choose(unsigned short valNumber, T x1, T x2, T x3 = 0, T x4 = 0, T x5 = 0, T x6 = 0, T x7 = 0, T x8 = 0, T x9 = 0)

Source Code

Selects a value randomly.

Parameter valNumber the number of values to test.

Example:

```
std::cout << is:: choose(3, 7, 12, 4) << std::endl; // its display in the console randomly 7 or 12 or 4
```

1.13 setVarLimit

template <**typename** T>

void setVarLimit(T &var, T valMin, T valMax)

Source Code

Allows to frame a value.

1.14 isIn

bool isIn(unsigned short valNumber, int const var, int x1, int x2, int x3 = 0, int x4 = 0, int x5 = 0, int x6 = 0, int x7 = 0, int x8 = 0, int x9 = 0)

Source Code

Check if the value of a variable is in a value set.

Example:

```
int year(2020);
if (is::isIn(3, year, 2020, 2012, 2000)) // this condition will be true because the value of year is found in the function
{
   // do something
}
```

1.15 isBetween

bool isBetween(float a, float b, float c)

Source Code

Check if a value is in an interval.

1.16 isCrossing

bool isCrossing(float l1, float r1, float l2, float r2)

Source Code

Checks if the point intersects another.

1.17 side

int side(Point m, Point a,Point b)

Source Code

Return -1 to the left, 1 to the right, 0 if **a b c** are aligned.

1.18 sign

int sign(float x)

Source Code

Return the sign of value.

1.19 pointDirection

template <**typename** T>

T pointDirection(float x1, float y1, float x2, float y2)

Source Code

Determine the angle between two points.

1.20 pointDistance

float pointDistance(float x1, float y1, float x2, float y2)

Source Code

Determine the distance between two points.

1.21 radToDeg

float radToDeg(float x)

Source Code

Convert radiant to degree.

1.22 degToRad

float degToRad(float x)

Source Code

Convert degree to radian.

1.23 lengthDirX

float lengthDirX(float dir, float angle)

Source Code

Return the component of x.

1.24 lengthDirY

float lengthDirY(float dir, float angle)

Source Code

Return the component of y.

1.25 increaseVar

template <**typename** T>

void increaseVar(const float &DELTA_TIME, T &var, T increaseValue, T varFinal, T varMax)

Source Code

Increment a variable with execution time.

Example:

```
int var(0);
is::increaseVar(DELTA_TIME, var, 1, 15, 10)); // the variable "var" will increment with the value 1. If it is
                                               // greater than 10 it becomes 15 and the increment stops
```

1.26 decreaseVar

template <**typename** T>

void decreaseVar(const float &DELTA_TIME, T &var, T decreaseValue, T varFinal = 0, T varMin = 0)

Source Code

Decrement a variable with execution time.

Example:

```
int var(40);
is::decreaseVar(DELTA_TIME, var, 1, 20, 25)); // the variable "var" will decrement with the value 1. If it is
                                               // less than 25 it becomes 20 and the decrement stops
```

1.27 collisionTest

bool collisionTest(Rectangle const &firstBox, Rectangle const &secondBox)

Source Code

Test the collision between two (2) rectangles.

2. Function on objects SFML

2.1 getSFMLObjAngle

template <**class** T>

float getSFMLObjAngle(T obj)

Source Code

Return the angle of the object.

2.2 getSFMLObjXScale

template <**class** T>

float getSFMLObjXScale(T obj)

Source Code

Return the x-scale of the object.

2.3 getSFMLObjYScale

template <**class** T>

float getSFMLObjYScale(T obj)

Source Code

Return the y-scale of the object.

2.4 getSFMLObjWidth

template <**class** T>

float getSFMLObjWidth(T obj)

Source Code

Return the width of the object.

2.5 getSFMLObjHeight

template <**class** T>

float getSFMLObjHeight(T obj)

Source Code

Return the height of the object.

2.6 getSFMLObjOriginX

template <**class** T>

float getSFMLObjOriginX(T obj)

Source Code

Return the origin x.

2.7 getSFMLObjOriginY

template <**class** T>

float getSFMLObjOriginY(T obj)

Source Code

Return the origin y.

2.8 getSFMLObjX

- ***First form***

template <**class** T>

float getSFMLObjX(T obj)

- ***Second form***

template <**class** T>

float getSFMLObjX(T *obj)

Source Code

Return position x.

2.9 getSFMLObjY

- ***First form***

template <**class** T>

float getSFMLObjY(T obj)

- ***Second form***

template <**class** T>

float getSFMLObjY(T *obj)

Source Code

Return position y.

2.10 setSFMLObjAngle

template <**class** T>

void setSFMLObjAngle(T &obj, float angle)

Source Code

Set the angle.

2.11 setSFMLObjRotate

template <**class** T>

void setSFMLObjRotate(T &obj, float rotationSpeed)

Source Code

Set the rotation of the object.

2.12 setSFMLObjScaleX_Y

template <**class** T>

void setSFMLObjScaleX_Y(T &obj, float x, float y)

Source Code

Define the scale x and y.

2.13 setSFMLObjScale

template <**class** T>

void setSFMLObjScale(T &obj, float scale)

Source Code

Set the scale x and y with the same value.

2.14 setSFMLObjOrigin

template <**class** T>

void setSFMLObjOrigin(T &obj, float x, float y)

Source Code

Set the origin x and y.

2.15 setSFMLObjX

template <**class** T>

void setSFMLObjX(T &obj, float x)

Source Code

Defines the position x.

2.16 setSFMLObjY

template <**class** T>

void setSFMLObjY(T &obj, float y)

Source Code

Defines the position y.

2.17 centerSFMLObj

template <**class** T>

void centerSFMLObj(T &obj)

Source Code

Center the object in x and y.

2.18 centerSFMLObjX

template <**class** T>

void centerSFMLObjX(T &obj)

Source Code

Center the object in x.

2.19 centerSFMLObjY

template <**class** T>

void centerSFMLObjY(T &obj)

Source Code

Center the object in y.

2.20 setSFMLObjX_Y

- ***First form***

template <**class** T>

void setSFMLObjX_Y(T &obj, sf::Vector2f position)

- ***Second form***

template <**class** T>

void setSFMLObjX_Y(T &obj, float x, float y)

Source Code

Defines the position x and y.

2.21 moveSFMLObjX

template <**class** T>

void moveSFMLObjY(T &obj, float speed)

Source Code

Moves the SFML object on the x-axis.

2.22 moveSFMLObjY

template <**class** T>

void moveSFMLObjY(T &obj, float speed)

Source Code

Moves the SFML object on the y-axis.

2.23 setSFMLObjSize

- ***First form***

template <**class** T>

void setSFMLObjSize(T &obj, float x, float y)

- ***Second form***

template <**class** T>

void setSFMLObjSize(T &obj, sf::Vector2f v)

Source Code

Set the size of the object.

2.24 setSFMLObjAlpha

- ***First form***

template <**class** T>

void setSFMLObjAlpha(T &obj, unsigned int alpha)

- ***Second form***

template <**class** T>

void setSFMLObjAlpha(T &obj, unsigned int alpha, sf::Uint8 r, sf::Uint8 g, sf::Uint8 b)

- ***Third form***

template <**class** T>

void setSFMLObjAlpha(T &obj, unsigned int alpha, sf::Uint8 rgb)

Source Code

Set transparency. ***Can generate WARNINGS if used on texts and geometric shapes!***

2.25 setSFMLObjAlpha2

template <class T>

void setSFMLObjAlpha2(T &obj, unsigned int alpha)

Source Code

Defines transparency for text, rectangles, etc. ***Does not work for sprites!***

2.26 setSFMLObjColor

template <class T>

void setSFMLObjColor(T &obj, sf::Color color)

Source Code

Set the color of the object (Sprite).

2.27 setSFMLObjFillColor

template <class T>

void setSFMLObjFillColor(T &obj, sf::Color color)

Source Code

Defines the color of the object (Text, Rectangle, etc.).

2.28 scaleAnimation

template <class T>

void scaleAnimation(float const &DELTA_TIME, float &var, T &obj, short varSign = 1, float scaleSize = 1.f)

Source Code

Allows you to make a stretch animation on an SFML object.

2.29 setFrame

void setFrame(sf::Sprite &sprite, float frame, int subFrame, int frameSize = 32, int recWidth = 32, int recHeight = 32)

Source Code

Defines the animation of a sprite ***(click here Figure 1 to see how it is used)***.

2.30 setSFMLObjOutlineColor

- ***First form***

template <class T>

void setSFMLObjOutlineColor(T &obj, sf::Color color)

Source Code

Set the outline color.

- ***Second form***

template <class T>

void setSFMLObjOutlineColor(T &obj, float thickness, sf::Color color)

Source Code

Set the outline color and its size.

2.31 setSFMLObjTexRec

template <**class** T>

void setSFMLObjTexRec(T &obj, int x, int y, int w = 32, int h = 32)

Source Code

Set the **intRect**.

2.32 setSFMLObjProperties

template <**class** T>

void setSFMLObjProperties(T &obj, float x, float y, float angle = 0.f, int alpha = 255, float xScale = 1.f, float yScale = 1.f)

Source Code

Defines the various properties of an SFML object.

2.33 loadSFMLObjResource

- ***First form***

template <**class** T>

bool loadSFMLObjResource(T &obj, **std::string** filePath, bool stopExecution = **false**)

Source Code

Allows to load a resource for an SFML object (Texture, Sound Buffer, etc.).

Parameter :

obj SFML object.

filePath resource file path.

stopExecution allows to stop the execution of the program in case of error.

Return **true** if the resource file has been loaded correctly and **false** if not.

- ***Second form***

template <**class** T>

inline bool loadSFMLObjResource(sf::SoundBuffer &sb, sf::Sound &snd, T &obj, **std::string** filePath, bool stopExecution = **false**)

Source Code

Allows to load a resource file for a Sound Buffer and associate it with a sound.

2.34 getSFMLSndState

template <**class** T>

bool getSFMLSndState(T &obj, sf::Sound::Status state)

Source Code

Return the state of the sound.

2.35 collisionTestSFML

template <**class** A, **class** B>

bool collisionTestSFML(A const &objA, B const &objB)

Source Code

Test the collision between two (2) SFML objects.

2.36 createRectangle

void createRectangle(sf::RectangleShape &rec, sf::Vector2f recSize, sf::Color color, float x = 0.f, float y = 0.f, bool center = **true**)

Source Code

Create a rectangle with various parameters.

2.37 textStyleConfig

void textStyleConfig(sf::Text &txt, bool underLined, bool boldText, bool italicText)

Source Code

Defines the style of a text.

2.38 createWText

void createWText(sf::Font const& fnt, sf::Text &txt, **std::wstring** const &text, float x, float y, sf::Color color, int txtSize = 20, bool underLined = **false**, bool boldText = **false**, bool italicText = **false**)

Source Code

Create a text with a **std::wstring**.

2.39 createText

- ***First form***

void createText(sf::Font const& fnt, sf::Text &txt, **std::string** const &text, float x, float y, int txtSize = 20, bool underLined = **false**, bool boldText = **false**, bool italicText = **false**)

- ***Second form***

void createText(sf::Font const& fnt, sf::Text &txt, **std::string** const &text, float x, float y, bool centerText, int txtSize = 20, bool underLined = **false**, bool boldText = **false**, bool italicText = **false**)

- ***Third form***

void createText(sf::Font const& fnt, sf::Text &txt, **std::string** const &text, float x, float y, sf::Color color, int txtSize = 20, bool underLined = **false**, bool boldText = **false**, bool italicText = **false**)

- ***Fourth form***

void createText(sf::Font const& fnt, sf::Text &txt, **std::string** const &text, float x, float y, sf::Color color, bool centerText, int txtSize = 20, bool underLined = **false**, bool boldText = **false**, bool italicText = **false**)

- ***Fifth form***

void createText(sf::Font const& fnt, sf::Text &txt, **std::string** const &text, float x, float y, sf::Color color, sf::Color outlineColor, int txtSize = 20, bool underLined = **false**, bool boldText = **false**, bool italicText = **false**)

Source Code

These functions allow to create text with various parameters.

2.40 createSprite

- ***First form***

void createSprite(sf::Texture &tex, sf::Sprite &spr, sf::Vector2f position, sf::Vector2f origin, bool smooth = **true**)

- ***Second form***

void createSprite(sf::Texture &tex, sf::Sprite &spr, sf::IntRect rec, sf::Vector2f position, sf::Vector2f origin, bool repeatTexture = **false**, bool smooth = **true**)

- ***Third form***

void createSprite(sf::Texture &tex, sf::Sprite &spr, sf::IntRect rec, sf::Vector2f position, sf::Vector2f origin, sf::Vector2f scale, unsigned int alpha = 255, bool repeatTexture = **false**, bool smooth = **true**)

Source Code

These functions allow to create a sprite with various parameters.

2.41 mouseCollision

- ***First form***

template <**class** T>

bool mouseCollision(sf::RenderWindow &window, T const &obj

#if defined(__ANDROID__)

, unsigned int finger = 0

#endif

)

Source Code

Windows, Linux: Detects if the mouse cursor collides with an object in the window.

Android: Detects if the user touches an object in the window.

Parameter:

obj the object with which we want to test.

finger represents the finger.

Return true if there is a collision and **false** if not.

Example :

```
if (mouseCollision(window, sprite))
{
  // do something
}
```

- ***Second form***

```
template <class T>

bool mouseCollision(sf::RenderWindow &window, T const &obj, sf::RectangleShape &cursor

                    #if defined(__ANDROID__)

                    , unsigned int finger = 0

                    #endif

                    )
```

Source Code

Windows, Linux: Detects if the mouse cursor collides with an object in the window.

Android: Detects if the user touches an object in the window.

Parameter:

obj the object with which we want to test.

cursor allows to recover the position of the collision point.

finger represents the finger.

Return **true** if there is a collision and **false** if not.

Example:

```
sf::ReactangleShape rec;
if (mouseCollision(window, sprite, rec))
{
   float cursorXPosition = rec.getPosition.x();
   float cursorYPosition = rec.getPosition.y();
}
```

3. Other functions

3.1 vibrate

```
int vibrate(sf::Time duration)
```

Source Code

Launches the Android vibrator.

3.2 openURL

```
void openURL(std::string urlStr)
```

Source Code

Open a URL in the browser (e.g www.website.com).

3.3 setScreenLock

```
void setScreenLock(bool disableLock)
```

Source Code

Set android screen lock.

3.4 jstring2string

static std::string jstring2string(JNIEnv *env, jstring jStr)

Source Code

Convert **jstring** to **std::string.**

3.5 getDeviceId

static std::string getDeviceId(JNIEnv *env, ANativeActivity *activity)

Source Code

Return Android device id.

External library

1. Swoosh

It is integrated by default to the engine. It is thanks to it that the engine manages to make transitions effects.

For more information please click here.

2. Tiny File Dialogs (only for Windows and Linux)

2.1 class TinyDialogBox

class TinyDialogBox;

Header : ***isEngine/ext_lib/TinyFileDialogs/TinyDialogsBox.h***

Source Code

A class that allows you to use the Tinyfiledialogs library in the simplest way. It allows you to use the dialog boxes of the operating system (Windows and Linux).

2.2 tinyString

#if !defined(SFML_SYSTEM_LINUX)

typedef wchar_t const* **tinyString**;

#else

typedef char const* **tinyString**;

#endif

Source Code

Custom type which allows to manipulate the data of tinyFileDialogs. When using tinyFileDialogs different data depending on the operating system. On windows the strings become wchar_t const* and on Linux char const*, which implies the use of two (2) different types having the same purpose for the same program. The **tinyString** type overcomes this problem by automatically determining the type that corresponds to the target operating system.

2.3 TINY_FILE_DIALOGBOX_PATH

static tinyString TINY_FILE_DIALOGBOX_PATH;

Source Code

Stores file path of dialog box.

2.4 enum FileDialogType

enum FileDialogType;

Enumerator	
SAVE_FILE	Save file
LOAD_FILE	Load file

Source Code

Represents the type of dialog box to display.

2.5 enum DialogType

enum DialogType;

Enumerator	
OK	Message with button OK
OKCANCEL	Message with button OK et CANCEL
YESNO	Message with button YES et NO

Source Code

Represents the buttons that will be displayed on the dialog box.

2.6 enum IconType

enum IconType;

Enumerator	
INFO	Dialog box with an INFO icon
WARNING	Dialog box with an WARNING icon
ERROR_ICO	Dialog box with an ERROR icon
QUESTION	Dialog box with an QUESTION icon

Source Code

Represents the icon that will be displayed on the dialog box.

2.7 enumDialogTypeToStr / enumIconTypeToStr

static tinyString const enumDialogTypeToStr(DialogType val)

static tinyString const enumIconTypeToStr(IconType val)

Source Code

These functions convert **enum** to **string** which will be used later in the library functions.

2.8 showDialogBox

static int showDialogBox(**tinyString** title,

```
                    tinyString msgError,
                    DialogType dialogType,
                    IconType iconType
                    )
```

Source Code

Displays a message dialog box.

Return 1 when the user clicks on the OK button and 0 when he clicks on CANCEL or NO.

2.9 showFileDialogBox

```
static std::string showFileDialogBox(FileDialogType type,
                              tinyString title,
                              tinyString filterPatterns[],
                              #if !defined(SFML_SYSTEM_LINUX)
                              tinyString fileName = L"file",
                              tinyString msgError = L"Unable to access file!",
                              tinyString errTitle = L"Error"
                              #else
                              tinyString fileName = "file",
                              tinyString msgError = "Unable to access file!",
                              tinyString errTitle = "Error"
                              #endif
                              )
```

Source Code

Displays a file dialog box.

Return file path if the function was successful and **"" (empty string)** if not.

2.10 showFolderDialogBox

```
static std::string showFolderDialogBox(tinyString title,
                              #if !defined(SFML_SYSTEM_LINUX)
                              tinyString defaultPath = L"C:\\" ,
                              tinyString msgError = L"Unable to access folder!",
                              tinyString errTitle = L"Error"
                              #else
                              tinyString defaultPath,
tinyString msgError = "Unable to access folder!",

tinyString errTitle = "Error"

#endif

)
```

Source Code

Displays a folder selection dialog box.

Return folder path if function was successful and **"" (empty string)** if not.

3. Box 2D

Box 2D is a physical engine integrated into the game engine. To use it in a scene you must include it this way:

#include "../../../isEngine/ext_lib/Box2D/Box2D.h"

Game Engine

1. class GameEngine

class GameEngine;

Header : ***isEngine/core/GameEngine.h***

Source Code

This Class ensures the interconnection of the different components of the engine and launches the rendering loop in which the game will take place.

2. Methods of GameEngine

2.1 GameEngine

GameEngine()

Source Code

Default constructor.

2.2 initEngine

bool initEngine()

Source Code

Initializes the game engine.

2.3 play

bool play()

Source Code

Game engine main render loop.

2.4 basicSFMLmain

bool basicSFMLmain()

Source Code

Classic SFML window rendering loop.

2.5 getRenderWindow

sf::RenderWindow& getRenderWindow()

Source Code

Return SFML window.

Game setup

namespace GameConfig;

*Header : **app_src/config/GameConfig.h***

Allows you to define parameters to preconfigure these parts of the game: The size of the window and the view, The keyboard and mouse keys to use to control the game, the game information (name, author, version), path resource files (sound, image, backup, etc.) and Admob information.

1. enum DisplayOption

enum DisplayOption;

Enumerator	
RESUME_GAME	When player close pause menu
GAME_OPTION_RESTART	Restart the scene with the restart option
QUIT_GAME	When player use quit option
INTRO	Access the Introduction scene
RESTART_LEVEL	Restart the scene when you lose
NEXT_LEVEL	Go to the next level
MAIN_MENU	Access the Main Menu scene
GAME_LEVEL	Access the Game Level scene
GAME_OVER	Access the Game Over scene
GAME_END_SCREEN	Go to the End of Game scene

Source Code

Allows to manipulate the different scenes and menu pause.

2. Window setting

2.1 WINDOW_WIDTH

static const unsigned int WINDOW_WIDTH

Source Code

Set window width.

2.2 WINDOW_HEIGHT

static const unsigned int WINDOW_HEIGHT

Source Code

Set window height.

2.3 VIEW_WIDTH

static const float VIEW_WIDTH

Source Code

Set view width.

2.4 VIEW_HEIGHT

static const float VIEW_HEIGHT

Source Code

Set view height.

2.5 FPS

static const float FPS

Source Code

Set the FPS (Frame Per Second) of the game.

2.6 WINDOW_SETTINGS

static const is::WindowStyle WINDOW_SETTINGS

Source Code

Set the window style.

3. Parameter of validation buttons

3.1 KEY_VALIDATION_MOUSE

static const sf::Mouse::Button KEY_VALIDATION_MOUSE

Source Code

Represents the button that validates the options with the mouse.

3.2 KEY_VALIDATION_KEYBOARD

static const sf::Keyboard::Key KEY_VALIDATION_KEYBOARD

Source Code

Represents the key that validates the options with the keyboard.

3.3 KEY_CANCEL

static const sf::Keyboard::Key KEY_CANCEL

Source Code

Represents the key that cancels options with the keyboard.

4. Keyboard key setting

4.1 KEY_A

static const sf::Keyboard::Key KEY_A

Source Code

Represents the key A.

4.2 KEY_B

static const sf::Keyboard::Key KEY_B

Source Code

Represents the key B.

4.3 KEY_LEFT

static const sf::Keyboard::Key KEY_LEFT

Source Code

Represents the key LEFT.

4.4 KEY_RIGHT

static const sf::Keyboard::Key KEY_RIGHT

Source Code

Represents the key RIGHT.

4.5 KEY_UP

static const sf::Keyboard::Key KEY_UP

Source Code

Represents the key UP.

4.6 KEY_DOWN

static const sf::Keyboard::Key KEY_DOWN

Source Code

Represents the key DOWN.

5. Game information

5.1 MAJOR

static const std::wstring MAJOR

Source Code

Set the major version.

5.2 MINOR

static const std::wstring MINOR

Source Code

Set the minor version.

5.3 getGameVersion

inline std::wstring getGameVersion()

Source Code

Return version of the game.

5.4 GAME_NAME

static std::wstring const GAME_NAME

Source Code

Set the name of the game.

5.5 GAME_AUTHOR

static std::wstring const GAME_AUTHOR

Source Code

Set the name of the author.

6. Admob setting

namespace AdmobConfig;

Allows you to define Admob information so that ads can be displayed in the game. ***This information is provided on the Google Admob platform!***

6.1 Ad Id

6.1.1 kAdMobAppID

static const char* kAdMobAppID

Source Code

Admob application code.

6.1.2 kBannerAdUnit

static const char* kBannerAdUnit

Source Code

Banner code.

6.1.3 kRewardedVideoAdUnit

static const char* kRewardedVideoAdUnit

Source Code

Reward video code.

6.2 Banner size

6.2.1 kBannerWidth

static const int kBannerWidth

Source Code

Set the width of the ad banner.

6.2.2 kBannerHeight

static const int kBannerHeight

Source Code

Set the height of the ad banner.

6.3 Target audience

6.3.1 kBirthdayDay

static const int kBirthdayDay

Source Code

Set users' birth day.

6.3.2 kBirthdayMonth

static const int kBirthdayMonth

Source Code

Set users' birth month.

6.3.3 kBirthdayYear

static const int kBirthdayYear

Source Code

Define users' year of birth.

6.3.4 kKeywords

static const char* kKeywords[]

Source Code

Keywords to use when requesting an ad.

7. Path of the resource files

7.1 GUI_DIR

static std::string const GUI_DIR

Source Code

Path of resource files that serve as a graphical interface.

7.2 FONT_DIR

static std::string const FONT_DIR

Source Code

Path of resource files that serve as font.

7.3 SPRITES_DIR

static std::string const SPRITES_DIR

Source Code

Path of resource files that serve as Sprite.

7.4 TILES_DIR

static std::string const TILES_DIR

Source Code

Path to resource files that serve as tiles and background.

7.5 SFX_DIR

static std::string const SFX_DIR

Source Code

Path of resource files that serve as SFX.

7.6 MUSIC_DIR

static std::string const MUSIC_DIR

Source Code

Path to resource files that serve as music.

8. Game package name (Android)

static std::string const PACKAGE_NAME

Source Code

Name of the game package. Represents the place where your data will be saved on Android.

You must apply this name for the **applicationId** in the **build.gradle** file

9. Backup file path

9.1 GAME_DATA_FILE

static std::string const GAME_DATA_FILE

Source Code

Path to the game save file.

9.2 CONFIG_FILE

static std::string const CONFIG_FILE

Source Code

Path to the game configuration file.

9.3 GAME_PAD_FILE

static std::string const GAME_PAD_FILE

Source Code

Path for the configuration file of the Virtual Game Pad on Android.

1. class GameActivity

class GameActivity;

Header : ***app_src/activity/GameActivity.h***

Source Code

Allows you to launch the different game scenes. Another special feature of this class is that it links the engine scenes and the SWOOSH library in order to be able to use the transition effects.

2. Elements of GameActivity

2.1 GameActivity

GameActivity(ActivityController& controller, GameSystemExtended &gameSysExt)

Source Code

Class constructor, it takes as parameter the activity controller (from the SWOOSH library) and game system manager ***(click here for more information: 1).***

2.2 m_gameScene

std::shared_ptr<**is::**GameDisplay> m_gameScene;

Source Code

Instance of the scene that will be used.

2.3 onStart

virtual void onStart()

Source Code

When the scene is launched.

2.4 onUpdate

virtual void onUpdate(double elapsed)

Source Code

Used to update scene information.

2.5 onLeave

virtual void onLeave()

Source Code

When the scene is no longer used (interruption).

2.6 onExit

virtual void onExit()

Source Code

When we leave the scene for another.

2.7 onEnter

virtual void onEnter()

Source Code

When the segue of the scene begins.

2.8 onResume

virtual void onResume()

Source Code

When we resume the scene after an interruption.

2.9 onDraw

virtual void onDraw(sf::RenderTexture& surface)

Source Code

Displays the scene.

2.10 onEnd

virtual void onEnd()

Source Code

When we leave the scene (destruction).

Game Level

1. Level

In is::Engine the game levels are integer arrays contained in header files (file.h). These levels are created thanks to **is::Level Editor** (project link) which is delivered with the engine.

Header : ***app_src/levels/Level.h***

2. Integration of a level

To integrate a level we include its header in the **Level.h** file in this way:

#include "../levels/level_1.h"

3. Elements to manage levels

3.1 namespace level

namespace level;

Source Code

Contains the elements which are used to manage the levels.

3.2 enum LevelId

```
enum LevelId
```

```
{
  LEVEL_1,
  LEVEL_2,
  /* ... */
  , LEVEL_MAX // Allows to know the total number of integrated level
}
```

Source Code

Represents the index of each level. Each time a new level is integrated into the engine, you must declare its index.

3.3 getLevelMap

```
inline short const* getLevelMap(int CURRENT_LEVEL)
```

Source Code

Return the level array entered in the parameter.

Each time a new level is integrated, you must enter the instruction that will return this level in the function.

Example:

- ***Integration in function:***

```
// Returns the level array found in level_1.h
inline short const* getLevelMap(int CURRENT_LEVEL)
{
  switch (CURRENT_LEVEL)
  {
    case LEVEL_1 : return LEVEL_1_MAP; break; // LEVEL_1_MAP is the name of the array found in level_1.h
  //...
```

- ***Use in an external source file:*** **(*This is a simple example just to explain the principle to you. To go further, please refer to the Engine Demo*)**

```
short *currentLevelArray = getLevelMap(LEVEL_1); // Return the array which is in level_1.h
```

Game language

1. Languages

Languages are represented in is::Engine by string arrays.

Header : app_src/language/GameLanguage.h

2. Elements to manage languages

2.1 namespace lang

```
namespace lang;
```

Source Code

Used to manage game languages.

2.2 enum GameLanguage

```
enum GameLanguage

{

   ENGLISH,  ///< Represents the English language

   FRANCAIS, ///< Represents the French language

   /* ... */

}
```

Source Code

This enumeration allows to implement the index of each language in order to provide the used more easily during the development.

Example:

- ***Create a sentence:***

```
static std::string hello_world[] = { "Hello World !", "Salut le monde !" }; // To put in GameLanguage.h
```

- ***Use:*** **(*This is a simple example just to explain the principle to you. To go further, please refer to the Engine Demo*)**

```
gameSystemExt.m_gameLanguage = is::lang::GameLanguage::ENGLISH; // Choice of English language

is::showLog(is::lang::hello_world[gameSystemExt.m_gameLanguage]); // we will have in the console: Hello World !

gameSystemExt.m_gameLanguage = is::lang::GameLanguage::FRANCAIS; // Choice of French language

is::showLog(is::lang::hello_world[gameSystemExt.m_gameLanguage]); // we will have in the console: Salut le monde !"
```

Game Dialog Box

1. class GameDialog

```
class GameDialog;
```

Header : ***app_src/objects/widgets/GameDialog.h***

Source Code

Class that allows you to display dialog boxes like in RPG games. It is closely related to the language part of the game ***(click here for more information: 1)***. To be able to display a dialog you must define a string array representing this dialog in **GameLanguage.h**

2. Elements of GameDialog

2.1 GameDialog

```
GameDialog(is::GameDisplay *scene)
```

Source Code

Constructor of the class, it takes as a parameter the scene in which it is used.

2.2 enum DialogIndex

```
enum DialogIndex

{

  DIALOG_NONE = -1,

  DIALOG_PLAYER_MOVE, // Represents the dialog that talks about how to move the player

  /* ... */

};
```

Source Code

Represents the different dialogs that will be displayed in the game. The information that is defined inside is linked to the language part of the game.

Each time an index is added we must declare its string array in **GameLanguage.h**.

Example:

- ***DIALOG_PLAYER_MOVE dialog declaration in GameLanguage.h :***

```
static std::wstring dialog_player_move[] = {L"Press LEFT or RIGHT to move.\n"

                                             "Press A to Jump.",

                                             L"Appuie sur GAUCHE ou DROITE pour te déplacer.\n"

                                             "Appuie sur A pour sauter."};
```

2.3 linkArrayToEnum

```
void linkArrayToEnum()
```

Source Code

Connect the string array found in **GameLanguage.h** and the dialogue index.

Example:

- ***Link an Index and its string array:*** **(*This is a simple example just to explain the principle to you. To go further, please refer to the Engine Demo*)**

```
void linkArrayToEnum()

{

// ...

switch (m_dialogIndex)

{

  case DIALOG_PLAYER_MOVE: // dialogue index

  m_msgIndexMax = is::arraySize(is::lang::dialog_player_move); // Determines the number of sentences
```

```
checkMsg(is::lang::dialog_player_move); // Define the dialogue thanks to its string array

break;

// ...
```

2.4 loadResources

void loadResources(sf::Texture &tex, sf::Font &fnt);

Source Code

Load the resource files of the dialog box.

2.5 step

void step(const float &DELTA_TIME)

Source Code

Updates the information in the dialog box.

2.6 setDialog

void setDialog(DialogIndex dialogIndex)

Source Code

Defines the dialog that will be launched.

2.7 setMouseInCollison

void setMouseInCollison(bool val)

Source Code

Force the collision of the mouse cursor or finger (on Android) with the dialog box.

2.8 draw

void draw(sf::RenderTexture &surface)

Source Code

Displays the dialog box.

2.9 getDialogIndex

DialogIndex getDialogIndex() const

Source Code

Return the enumerator of the dialog that is displayed.

2.10 getMouseInCollison

bool getMouseInCollison() const

Source Code

Return **true** when the mouse cursor or the user's finger (on Android) touches the dialog box, **false** if not.

2.11 showDialog

bool showDialog() const

Source Code

Return **true** when the dialog is open and **false** if not.

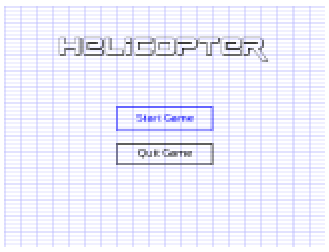
Game Example

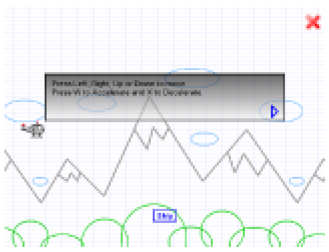
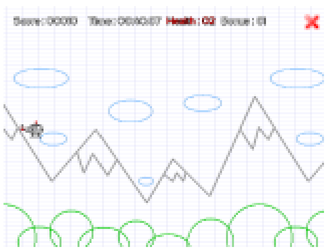
1. Introduction

In this part of the document we will find out how to use the functions of is::Engine to create a mini game. Note that this is just a short tutorial to get you started with the engine.

We are going to create an arcade game in which we control a Helicopter whose goal is to avoid obstacles and collect bonus items that increase time of the chronometer and its score. If the level clock reaches zero (0), it loses the game.

The game will be playable on Android and PC.



You can access the project here.

2. How the game will be created?

2.1 Here are the elements of the engine that the game will use

- GameDisplay class to create scenes
- MainObject class and these parents to create game play objects (Player, HUD, Bonus, etc.)
- GameKeyData class to control the player
- GameDialog class to display the tutorial
- GameLangague.h to add sentences to translate
- Some functions found in GameFunction.h
- Activity class to launch the different scenes and make them interact with each other

2.2 The objects that will be used in the game

- A Main Menu which will contain these objects:
  - A Text for the title of the game
  - Two (2) sprites which will serve as Buttons: One to start the game and another to exit
  - Two (2) texts which will serve as title for the Buttons
- A Scene called GameLevel where the game takes place and will have for content:
  - A player object that will serve as a helicopter
  - A HUD object
  - A cross-shaped sprite to exit the Level
  - A sprite for the background
  - Sounds
  - Text to display the game over message
  - An object container (**std::vector**) for Bonuses
  - An object container (**std::vector**) for Obstacles

2.3 The roles of objects

- Activity class

  - Launch the different scenes
  - Transition between Main Menu and Game Level and vice versa.

- MainMenu class
  - Navigate the menu with the mouse (touch on Android) and keyboard
  - Use the validation keys to choose an option
  - Exit the menu using a dialog box
- GameLevel class
  - Start the game
  - Restart the level when the player loses
  - Quit the level when the user clicks on the cross (sprite)
- Class Player will be a Helicopter:
  - The UP, DOWN, LEFT, RIGHT keys will be used to move the object
  - The key A to accelerate
  - The key B for normal speed
  - Animated the sprite
- Bonus class
  - Disappears when the player touches it
  - Increase the Score and time of the player and play a sound when it is destroyed
- Obstacle class
  - Collision with the player (remove health)
- HUD class
  - Displays the level timer
  - Displays the number of Bonuses
  - Displays the player's score
  - Displays the player's health

3. Integration of game sentences

3.1 Creation of sentences in GameLanguage.h

```
#include "../../isEngine/system/function/GameKeyName.h"

namespace is
{
/// Access to content that allows internationalization of the game
namespace lang
{
/// Represent the index of each language
enum GameLanguage
{
   ENGLISH,  ///< English language index
   FRANCAIS, ///< French language index
};

// ----------------------- message box answer -----------------------
static std::string pad_answer_ok[] = {"OK", "OK"};
static std::string pad_answer_yes[] = {"YES", "OUI"};
static std::string pad_answer_no[]  = {"NO", "NON"};

// ----------------------- intro -----------------------
static std::string pad_game_language[] = {"English", "French"};

// ----------------------- menu -----------------------
static std::string pad_main_menu[] = {"Main Menu", "Menu Principal"};
static std::string pad_new_game[]  = {"Start Game", "Nouvelle Partie"};
static std::string pad_quit_game[] = {"Quit Game", "Quitter le Jeu"};
static std::string msg_quit_game[] = {"Quit game?", "Quitter le jeu?"};

// ----------------------- level dialog -----------------------
static std::string pad_dialog_skip[] = {"Skip", "Passer"};

#if defined(__ANDROID__)
static std::wstring dialog_player_move[] = {L"Press LEFT, RIGHT, UP or DOWN to move.\n"
                                  "Press A to Accelerate and B to decelerate.",
                                     L"Appuie sur GAUCHE, DROITE, HAUT, BAS pour te déplacer.\n"
                                     "Appuie sur A pour Accélerer et B pour Ralentire."};
#else
static std::wstring dialog_player_move[] = {L"Press " + is::getKeyWName(is::GameConfig::KEY_LEFT) + L", " +
                                             is::getKeyWName(is::GameConfig::KEY_RIGHT) + L", " +
                                             is::getKeyWName(is::GameConfig::KEY_UP) + L" or " +
                                             is::getKeyWName(is::GameConfig::KEY_DOWN) + L" to move.\n"
                                  "Press " + is::getKeyWName(is::GameConfig::KEY_A) + L" to Accelerate and " +
                                        is::getKeyWName(is::GameConfig::KEY_B) + L" to Decelerate.",
                                     L"Appuie sur " + is::getKeyWName(is::GameConfig::KEY_LEFT) + L", " +
                                                is::getKeyWName(is::GameConfig::KEY_RIGHT) + L", " +
                                                is::getKeyWName(is::GameConfig::KEY_UP) + L" ou " +
                                                is::getKeyWName(is::GameConfig::KEY_DOWN) + L" pour te déplacer.\n"
                                     "Appuie sur " + is::getKeyWName(is::GameConfig::KEY_A) + L" pour Accélerer et " +
                                               is::getKeyWName(is::GameConfig::KEY_B) + L" pour Ralentire."};
#endif

// ----------------------- game level -----------------------
static std::string msg_game_over[] = {"Your score  : ", "Votre score : "};
static std::string msg_clic_restart[] = {"Click to restart", "Cliquer pour recommencer"};
}
}
```

- **Explanation**

This file makes it possible to define the sentences to be translated which will be used in the game. A sentence to be translated is represented by an array of strings (std::string or std::wstring). The first index of the array represents the first language, the next index the second and so on.

- static std::wstring dialog_player_move

Source Code

Sentence that will be used later in the dialog box to show the user how to control the helicopter.

On Android we display how to move the player relative to the keys of the Virtual Game Pad and on PC relative to the keys of the keyboard (which can change according to the parameters defined in **GameConfig.h**).

- is::getKeyWName(is::GameConfig::KEY_LEFT)

Source Code

Allows to obtain the name of the keyboard key (in the form of a std::wstring) thanks to its associated code.

This allows you to know the name of the keyboard key associated with each action.

3.2 Associate of the dialog box with the sentence of the game

The code below is part of the GameDialog class declaration.

```
//...
enum DialogIndex
{
   DIALOG_NONE = -1,
   DIALOG_PLAYER_MOVE
};

// ...
void linkArrayToEnum()
{
   auto setMsg = [this](std::wstring txt)
   {
      m_strDialog = txt;
   };
   auto checkMsg =[this, &setMsg](std::wstring txt[])
   {
      if (m_msgIndex < m_msgIndexMax) setMsg(txt[m_msgIndex + m_scene->getGameSystem().m_gameLanguage]);
   };

   // each enum with its array of string
   switch (m_dialogIndex)
   {
      case DIALOG_PLAYER_MOVE:
         m_msgIndexMax = is::arraySize(is::lang::dialog_player_move);
         checkMsg(is::lang::dialog_player_move);
      break;

      default:
      break;
   }
}
// ...
```

- **Explanation**

The DialogIndex **enum** and the **void linkArrayToEnum()** function are the two elements of the **GameDialog** class which allows us to display sentences from **GameLanguage.h** with the dialog box.

- DIALOG_PLAYER_MOVE

Source Code

Represents the sentence dialog_player_move of **GameLanguage.h.** The elements of the DialogIndex **enum** are used to link the sentences of **GameLanguage.h** and the **GameDialog** class.

-
  ```
  switch (m_dialogIndex)
  {
     case DIALOG_PLAYER_MOVE:
        m_msgIndexMax = is::arraySize(is::lang::dialog_player_move);
        checkMsg(is::lang::dialog_player_move);
     break;
  ```

Source Code

These instructions allow you to associate a sentence from **GameLanguage.h** with the **GameDialog** class. The procedure is the same for any other type of sentence but do not forget that for each sentence (string array) you must define its element in **enum DialogIndex.**

4. Creation of game classes

4.1 Obstacle class

4.1.1 Header

```
#include "../../../isEngine/system/entity/MainObject.h"
#include "../../../isEngine/system/entity/parents/ScorePoint.h"

class Obstacle : public is::MainObject, public is::ScorePoint
{
public:
   Obstacle(sf::Texture &tex, float x, float y);
   void step(float const& DELTA_TIME);
};
```

➢ **Explanation**

Obstacle class is a class which inherits from **MainObject** (offers functions to manage the movement and display of the object) and **ScorePoint** a class which allows to assign bonus points to objects.

void step(float const& DELTA_TIME) allows to update instances of Obstacle class.

4.1.2 Implementation

4.1.2.1 Obstacle

```
Obstacle::Obstacle(sf::Texture &tex, float x, float y):
   MainObject(x, y),
   ScorePoint(20)
{
   // define collision mask
   m_w = 32;
   m_h = 32;
   m_speed = -12.f;

   // load texture
   is::createSprite(tex, m_sprParent, sf::IntRect(0, 0, 32, 32), sf::Vector2f(m_x, m_y), sf::Vector2f(0.f, 0.f), false, false);
   updateCollisionMask();
}
```

➢ **Explanation**

Constructor of the class that takes the texture and position of the object in the scene as parameters.

ScorePoint(20) represents the score that is assigned to the object. Inside the block there is the definition of the size of the collision mask, the speed of the object and the function which makes it possible to create the sprite of the object.

4.1.2.2 step

```
void Obstacle::step(float const& DELTA_TIME)
{
   m_x += ((m_speed * is::VALUE_CONVERSION) * DELTA_TIME);
   updateCollisionMask();
   updateSprite();
}
```

➢ **Explanation**

This method allows you to move the object to the left depending on its speed, update the position of the collision mask and the sprite.

4.2 Bonus class

4.2.1 Header

```
#include "../../../isEngine/system/entity/MainObject.h"
#include "../../../isEngine/system/entity/parents/Destructible.h"
#include "../../../isEngine/system/entity/parents/ScorePoint.h"
#include "../../../isEngine/system/entity/parents/Step.h"
#include "../../gamesystem_ext/GameSystemExtended.h"

class Bonus : public is::MainObject, public is::Destructible, public is::ScorePoint, public is::Step
{
public:
   Bonus(sf::Texture &tex, float x, float y);
   void step(float const &DELTA_TIME);
};
```

➢ **Explanation**

Class daughter of **MainObject**, it also inherits from **Destructible** which offers functions to manage the destruction of these instances explicitly. **ScorePoint** to assign a score point to the object which will be counted later. **Step** allows to manage the different steps of the object: collision with the player and destruction.

4.2.2 Implementation

4.2.2.1 Bonus

```
Bonus::Bonus(sf::Texture &tex, float x, float y):
   MainObject(x, y),
   Destructible(),
   ScorePoint(10),
   Step(0)
{
   m_w = 32;
   m_h = 32;
   m_speed = -15.f;
   is::createSprite(tex, m_sprParent, sf::IntRect(0, 0, 32, 32), sf::Vector2f(m_x, m_y), sf::Vector2f(16.f, 16.f));
}
```

➢ **Explanation**

Constructor that takes the texture of the sprite and the position of the object in the scene.

Inside, the size of the collision mask was defined with the speed of the object, followed by the function which creates the sprite of the object.

4.2.2.2 step

```
void Bonus::step(float const &DELTA_TIME)
{
   m_x += ((m_speed * is::VALUE_CONVERSION) * DELTA_TIME);
   if (m_step == 1) m_destroy = true;
   updateSprite();
   updateCollisionMask();
}
```

➢ **Explanation**

This method makes it possible to move the object and to start the destruction of the object when its step passes to 1. It also updates the properties of the sprite and that of the collision mask.

4.3 Player class

4.3.1 Header

```
#include "../../../isEngine/system/entity/MainObject.h"
#include "../../../isEngine/system/entity/parents/Health.h"
#include "../../../isEngine/system/entity/parents/HurtEffect.h"
#include "../../../isEngine/system/function/GameKeyData.h"

class Player : public is::MainObject, public is::Health, public is::HurtEffect
{
public:
   Player(GameKeyData &gameKey);
   void loadResources(sf::Texture &tex);
   void step(float const &DELTA_TIME);

private:
   GameKeyData &m_gameKey;
};
```

➢ **Explanation**

MainObjet's daughter class, **Health** offers methods to manage the player's health; **HurtEffect** allows to make an invincibility effect (make the object blink when it is hurted).

- void loadResources(sf::Texture &tex)

Source Code

Allocate external resources (used in the scene) to the object.

- GameKeyData &m_gameKey;

Source Code

Used to manage game controls.

4.3.2 Implementation

4.3.2.1 Player

```
Player::Player(GameKeyData &gameKey):
   MainObject(),
   Health(3),
   HurtEffect(m_sprParent),
   m_gameKey(gameKey)
{
   // define collision mask
   m_w = 40;
   m_h = 40;
   m_isActive = true;

   // initialize collision mask
   updateCollisionMask();
}
```

➢ **Explanation**

Constructor takes as a parameter the instance of the object which manages the controls of the game. It also allows to define the number of health of the player and to choose the sprite which will be used to make the invincibility effect when the player is hit by an obstacle.

Inside there is the definition of the size of the collision mask. The variable **m_isActive = true** allows the user to control the object when it has not lost.

4.3.2.2 loadResources

```
void Player::loadResources(sf::Texture &tex)
{
   is::createSprite(tex, m_sprParent, sf::IntRect(0, 0, 48, 48), sf::Vector2f(m_x, m_y), sf::Vector2f(0.f, 0.f));
}
```

- **Explanation**

Use the texture loaded in the scene to create the player sprite.

4.3.2.3 step

```
void Player::step(float const &DELTA_TIME)
{
   if (m_isActive)
   {
      // allow accelerate / decelerate player
      if (m_gameKey.m_keyBPressed) m_speed = 0.f;
      else if (m_gameKey.m_keyAPressed) m_speed = 200.f;

      // move
      float const SPEED(2.f);
      m_hsp = 0.f;
      m_vsp = 0.f;
      if (m_gameKey.m_keyRightPressed)      m_hsp = SPEED;
      else if (m_gameKey.m_keyLeftPressed)  m_hsp = -SPEED;
      else if (m_gameKey.m_keyDownPressed)  m_vsp = SPEED;
      else if (m_gameKey.m_keyUpPressed)    m_vsp = -SPEED;

      // animation
      m_frame += (0.33f * is::VALUE_CONVERSION) * DELTA_TIME; // image speed
      setFrame(0.f, 3.6f);

      // update collision mask (position, size, ...)
      updateCollisionMask();

      // update object position
      m_x += (m_hsp * is::VALUE_CONVERSION) * DELTA_TIME;
      m_y += (m_vsp * is::VALUE_CONVERSION) * DELTA_TIME;
   }
   else m_frame = 0.f;

   is::setFrame(m_sprParent, m_frame, 4, 48, 48, 48); // update sprite and animation
   updateSprite();
   hurtStep(DELTA_TIME);
}
```

- **Explanation**

Method in which we manage the behavior of the object. Here when the variable **m_isActive** is **true** then the user can make the helicopter speed up when he presses the **A key** and make it slow down when he presses the **B key**. He can also move the object with **the four (4) keys directional**. The animation of the Helicopter ***(which will be detailed below***) is also done in this block.

Note: when the user accelerates or slows down it also affects the other objects in the scene (Obstacles, Bonuses, Background).

- hurtStep(DELTA_TIME)

Source Code

Allows you to make the invincibility animation (flash the sprite).

Here's how the sprite is animated:

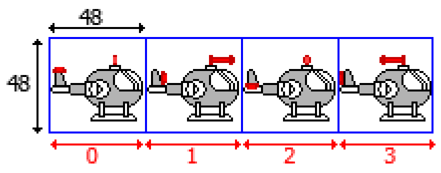


Figure 1

To animate the sprite we use a texture (**Figure 1**) composed of several sub-images having the same sizes. Each sub-image represents a value (in red) that the variable **m_frame** can take. Below the elements which make it possible to make an animation

- is::setFrame(m_sprParent, m_frame, 4, 48, 48, 48)

Source Code

The function to animate the sprite. It takes as a parameter the sprite which will be used, the sub image which will be displayed, the number of sub images ***on a line (here which is 4)*** and the 3 other parameters which are the size of the sub images (they are similar but have different purposes).

Note: The function automatically cuts the image.

- m_frame

Source Code

Allows to define the sprite sub-image which will be displayed.

- setFrame(0.f, 3.6f);

Source Code

Allows to define the interval of **m_frame**, i.e. the sub-images to choose.

4.4 HUD class

4.4.1 Header

```
#include "../../../isEngine/system/entity/MainObject.h"
#include "../../../isEngine/system/function/GameTime.h"
#include "../../gamesystem_ext/GameSystemExtended.h"
#include "Player.h"

class HUD : public is::MainObject
{
public:
  HUD(is::GameDisplay &scene, is::GameTime &gameTime, Player &player);
  void loadResources(sf::Font const &fnt);
  void step(float const &DELTA_TIME);
  void draw(sf::RenderTexture &surface);
  void setScore(int val);

private:
  is::GameDisplay &m_scene;
```

```
    is::GameTime &m_gameTime;
    Player &m_player;
    sf::Text m_txtHealth, m_txtBonus, m_txtLevelTime, m_txtScore;
};
```

- **Explanation**

Class to display game play information on screen.

- is::GameDisplay &m_scene

Source Code

Allows to have access to the scene in which the object is used to be able to position it and use the game play variables (score, number of bonuses collected).

- is::GameTime &m_gameTime

Source Code

Allows to display the chronometer.

- Player &m_player

Source Code

Allows to displays player health.

4.4.2 Implementation
4.4.2.1 HUD

```
HUD::HUD(is::GameDisplay &scene, is::GameTime &gameTime, Player &player) :
    m_scene(scene),
    m_gameTime(gameTime),
    m_player(player)
{}
```

- **Explanation**

Constructor who takes the scene, the object that manages the chronometer and the player's instance as parameters.

4.4.2.2 loadResources

```
void HUD::loadResources(sf::Font const &fnt)
{
    int const TXT_SIZE(20);
    is::createText(fnt, m_txtScore,     " ", 0.f, 0.f, sf::Color(255, 255, 255, 255), sf::Color(0, 0, 0, 255), TXT_SIZE);
    is::createText(fnt, m_txtLevelTime, " ", 0.f, 0.f, sf::Color(255, 255, 255, 255), sf::Color(0, 0, 0, 255), TXT_SIZE);
    is::createText(fnt, m_txtHealth,    " ", 0.f, 0.f, sf::Color(255, 0, 0, 255),     sf::Color(0, 0, 0, 255), TXT_SIZE);
    is::createText(fnt, m_txtBonus,     " ", 0.f, 0.f, sf::Color(255, 255, 255, 255), sf::Color(0, 0, 0, 255), TXT_SIZE);
}
```

- **Explanation**

Allows to use the font loaded in the scene to create the texts.

4.4.2.3 step

```
void HUD::step(float const &DELTA_TIME)
{
    float const TXT_X_POS(-300.f), TXT_Y_POS(16.f);
    m_txtScore.setString("Score : " + is::writeZero(m_scene.getGameSystem().m_currentScore, 4));
```

```
   is::setSFMLObjX_Y(m_txtScore, m_scene.getViewX() + TXT_X_POS, (m_scene.getViewY() - m_scene.getViewH() / 2.f) +
TXT_Y_POS);
   m_txtLevelTime.setString("Time : " + m_gameTime.getTimeString());
   is::setSFMLObjX_Y(m_txtLevelTime, m_scene.getViewX() + TXT_X_POS + 150.f, (m_scene.getViewY() -
m_scene.getViewH() / 2.f) + TXT_Y_POS);
   m_txtHealth.setString("Health : " + is::writeZero(m_player.getHealth()));
   is::setSFMLObjX_Y(m_txtHealth, m_scene.getViewX() + TXT_X_POS + 305.f, (m_scene.getViewY() -
m_scene.getViewH() / 2.f) + TXT_Y_POS);
   m_txtBonus.setString("Bonus : " + is::writeZero(m_scene.getGameSystem().m_currentBonus));
   is::setSFMLObjX_Y(m_txtBonus, m_scene.getViewX() + TXT_X_POS + 415.f, (m_scene.getViewY() -
m_scene.getViewH() / 2.f) + TXT_Y_POS);
}
```

- **Explanation**

This method is used to position the texts on the screen and to update their information.

4.4.2.4 draw

```
void HUD::draw(sf::RenderTexture &surface)
{
   surface.draw(m_txtScore);
   surface.draw(m_txtLevelTime);
   surface.draw(m_txtHealth);
   surface.draw(m_txtBonus);
}
```

- **Explanation**

Displays the various texts on the screen. ***This method is an overload!***

4.5 MainMenu class

4.5.1 Header

```
#include "../../../isEngine/system/function/GameFunction.h"
#include "../../../isEngine/system/display/GameDisplay.h"

class GameMenu : public is::GameDisplay
{
public:
   GameMenu(sf::RenderWindow &window, sf::View &view, sf::RenderTexture &surface, GameSystemExtended
&gameSysExt);
   void step();
   void componentsController();
   void draw();
   bool loadResources();

private:
   sf::Font m_fontTitle;
   sf::Texture m_texPad, m_texScreenBG;
   sf::Sprite m_sprPad1, m_sprPad2, m_sprScreenBG;
   sf::Text m_txtGameTitle, m_txtStartGame, m_txtQuit;
   bool m_isStart, m_closeApplication;
};
```

- **Explanation**

Class declaration that allows to create the scene of the Main Menu.

void componentsController()

Source Code

Method where the main menu buttons will be managed.

4.5.2 Implementation

4.5.2.1 MainMenu

```
GameMenu::GameMenu(sf::RenderWindow &window, sf::View &view, sf::RenderTexture &surface,
GameSystemExtended &gameSysExt):
   GameDisplay(window, view, surface, gameSysExt, sf::Color::White),
   m_isStart(true),
   m_closeApplication(false)
{}
```

➢ **Explanation**

Class constructor, takes the window, view, surface and manager of the game system as a parameter. It also allows to define the background color of the stage (here which is White).

4.5.2.2 loadResources

```
bool GameMenu::loadResources()
{
   if (!GameDisplay::loadParentResources()) return false;

   m_gameSysExt.m_gameLanguage = is::lang::GameLanguage::ENGLISH; // set default language

   // load textures
   if (!m_texPad.loadFromFile(is::GameConfig::GUI_DIR + "main_menu_pad.png"))           return false;
   if (!m_texScreenBG.loadFromFile(is::GameConfig::GUI_DIR + "screen_background.png"))    return false;
   if (!m_fontTitle.loadFromFile(is::GameConfig::FONT_DIR + "space_ranger_3d_mp_pv.otf")) return false;

   // game title
   is::createWText(m_fontTitle, m_txtGameTitle, is::GameConfig::GAME_NAME, 65.f, 32.f, sf::Color(0, 0, 0), 64);

   // create sprites
   float const XPOS(225.f), YPOS(200.f), BTYSIZE(0.9f);
   is::createSprite(m_texPad, m_sprPad1, sf::IntRect(0, 0, 192, 48), sf::Vector2f(XPOS, YPOS), sf::Vector2f(96.f, 24.f));
   is::createSprite(m_texPad, m_sprPad2, sf::IntRect(0, 0, 192, 48), sf::Vector2f(XPOS, YPOS + 70.f), sf::Vector2f(96.f,
24.f));
   is::createSprite(m_texPad, m_sprButtonSelect, sf::IntRect(192, 0, 192, 48), sf::Vector2f(XPOS, YPOS), sf::Vector2f(96.f,
24.f));
   is::setSFMLObjScaleX_Y(m_sprPad1, 1.f, BTYSIZE);
   is::setSFMLObjScaleX_Y(m_sprPad2, 1.f, BTYSIZE);

   // sprite background
   is::createSprite(m_texScreenBG, m_sprScreenBG, sf::IntRect(0, 0, 672, 512),sf::Vector2f(0.f, 0.f), sf::Vector2f(0.f , 0.f),
true);

   // create text for main menu
   float const TXT_Y_ON_BT(8.f);
   int const _PAD_TXT_SIZE(22);
   is::createText(m_fontSystem, m_txtStartGame, is::lang::pad_new_game[m_gameSysExt.m_gameLanguage],
              is::getSFMLObjX(m_sprPad1), is::getSFMLObjY(m_sprPad1) - TXT_Y_ON_BT, sf::Color::Blue, true,
_PAD_TXT_SIZE);
   is::createText(m_fontSystem, m_txtQuit, is::lang::pad_quit_game[m_gameSysExt.m_gameLanguage],
              is::getSFMLObjX(m_sprPad2), is::getSFMLObjY(m_sprPad2) - TXT_Y_ON_BT, true, _PAD_TXT_SIZE);
   return true;
}
```

➢ **Explanation**

This method loads the resources that will be used in the menu and positions the objects.

- m_gameSysExt.m_gameLanguage = **is::lang::**GameLanguage::ENGLISH;

Source Code

Allows you to define the default language of the game. If you change the value to **is::lang::GameLanguage::FRENCH** the French language will be chosen.

- **is::lang::**pad_new_game[m_gameSysExt.m_gameLanguage]

Source Code

- **is::lang::pad_new_game** : allows to use the array string found in **GameLanguage.h**.
- **[m_gameSysExt.m_gameLanguage]** : allows to choose the sentence that corresponds to the language.

4.5.2.3 componentsController

```
void GameMenu::componentsController()
{
  const short OP_START_GAME(0), OP_QUIT(1);

  // allow to know is mouse is in collision with sprite
  bool mouseInCollisonPad(false);

  // allows activated use of buttons
  if (!m_gameSysExt.keyIsPressed(is::GameConfig::KEY_UP) &&
    !m_gameSysExt.keyIsPressed(is::GameConfig::KEY_DOWN) &&
    !m_gameSysExt.isPressed())
      m_gameSysExt.m_keyIsPressed = false;

  // m_isClose allow to deactivate scene object
  if (!m_isClose)
  {
    if (mouseCollision(m_sprPad1) || mouseCollision(m_sprPad2)) mouseInCollisonPad = true;

    // change option with mouse (touch on Android)
    if (mouseCollision(m_sprPad1) && m_optionIndex != OP_START_GAME)  setOptionIndex(OP_START_GAME, true,
1.4f);
    if (mouseCollision(m_sprPad2) && m_optionIndex != OP_QUIT)    setOptionIndex(OP_QUIT, true, 1.4f);

    // avoid the long pressing button effect
    if (!mouseInCollisonPad && m_gameSysExt.isPressed(is::GameSystem::ValidationButton::MOUSE))
      m_gameSysExt.m_keyIsPressed = true;

    // change option with keyboard (only for PC)
    if (!m_gameSysExt.m_keyIsPressed && !mouseInCollisonPad)
    {
      if (m_gameSysExt.keyIsPressed(is::GameConfig::KEY_UP)) setOptionIndex(-1, false, 1.4f);
      else if (m_gameSysExt.keyIsPressed(is::GameConfig::KEY_DOWN)) setOptionIndex(1, false, 1.4f);
      if (m_optionIndex < OP_START_GAME) m_optionIndex = OP_QUIT;
      if (m_optionIndex > OP_QUIT) m_optionIndex = OP_START_GAME;
    }

    // launch a dialog box which allow to quit the game
    auto lauchDialogBox = [this]()
    {
      showMessageBox(is::lang::msg_quit_game[m_gameSysExt.m_gameLanguage]);
      m_closeApplication = true;
      m_keyBackPressed = false;
    };

    // validate menu option
    if ((m_gameSysExt.isPressed(is::GameSystem::ValidationButton::KEYBOARD) ||
      (m_gameSysExt.isPressed(is::GameSystem::ValidationButton::MOUSE) && mouseInCollisonPad)) &&
      (m_waitTime == 0 && !m_gameSysExt.m_keyIsPressed))
    {
      auto playSelectSnd = [this]()
      {
        m_gameSysExt.playSound(m_sndSelectOption);
        m_sprButtonSelectScale = 1.4f;
        m_gameSysExt.useVibrate(m_vibrateTimeDuration);
      };
      switch (m_optionIndex)
      {
      case OP_START_GAME:
        playSelectSnd();
        m_gameSysExt.m_launchOption = is::DisplayOption::GAME_LEVEL;
        m_isClose = true;
      break;
      case OP_QUIT: lauchDialogBox(); break;
      }
      m_keyBackPressed = false;
    }

    // Quit game
    if (m_keyBackPressed) lauchDialogBox();

    // change the color of the texts according to the chosen option
    setTextAnimation(m_txtStartGame, m_sprPad1, OP_START_GAME);
    setTextAnimation(m_txtQuit,  m_sprPad2, OP_QUIT);

    // PAD animation
    is::scaleAnimation(DELTA_TIME, m_sprButtonSelectScale, m_sprButtonSelect, is::getSFMLObjXScale(m_sprPad1));
  }
}
```

➢ **Explanation**

This method is a subfunction of **step()**. It allows to use the game keys and the mouse (becomes the touch function on Android) to navigate the menu and choose an option. It also allows to animate the main menu objects when you perform an action (mouse over, click, press a key).

- setOptionIndex(-1, **false**, 1.4f);

Source Code

Animate text, sprite and play a sound when changing an option.

- m_gameSysExt.m_launchOption = **is::**DisplayOption::GAME_LEVEL

Source Code

Inform the engine that the next scene to launch will be the Level scene.

4.5.2.4 step

```
void GameMenu::step()
{
  DELTA_TIME = getDeltaTime();
  updateTimeWait(DELTA_TIME);

  // even loop
  while (m_window.pollEvent(m_event))
  {
     controlEventFocusClosing();
     if (m_event.type == sf::Event::KeyReleased)
     {
        if (m_event.key.code == is::GameConfig::KEY_CANCEL) m_keyBackPressed = true;
     }
  }

  // starting mechanism
  if (m_isStart)
  {
     // window has focus
     if (m_windowIsActive)
     {
        if (!m_showMsg)
        {
           componentController();
        }
       // MESSAGE BOX
        else
        {
           updateMsgBox(DELTA_TIME);

           // when user closes message box in update function execute this instruction
           // "m_waitTime" allow to disable clicks on objects during a moment when user closes message box
           if (!m_showMsg)
           {
              if (m_closeApplication) // quit game
              {
                 if (m_msgAnswer == MsgAnswer::YES)
                 {
                    m_window.close();
                    m_isRunning = false;
                 }
                 else
                 {
                    m_waitTime = 20;
                    m_closeApplication = false;
                 }
              }
           }
        }
     }
  }

  if (m_isClose)
  {
     m_isStart = false;
     m_isRunning = false;
  }
}
```

- **Explanation**

This method manages the event part associated with the scene and the dialog box of the game engine ***(not that of the tutorial but the one that displays a YES - NO button),*** as well as the closing of the application.

- m_isRunning = **false**;

Source Code

Stops the execution of the scene in order to leave it.

4.5.2.5 draw

```
void GameMenu::draw()
{
   const short OP_START_GAME(0), OP_QUIT(1);

   // draw background
   m_surface.draw(m_sprScreenBG);

   // draw game title
   m_surface.draw(m_txtGameTitle);

   // draw button
   if (m_optionIndex != OP_START_GAME) m_surface.draw(m_sprPad1);
   if (m_optionIndex != OP_QUIT) m_surface.draw(m_sprPad2);
   m_surface.draw(m_sprButtonSelect);
   m_surface.draw(m_txtStartGame);
   m_surface.draw(m_txtQuit);

   // message box
   drawMsgBox();
}
```

➢ **Explanation**

Displays the components of the Main Menu.

4.6 GameLevel class

4.6.1 Header

```
#include <memory>

#include "../../../isEngine/system/display/GameDisplay.h"
#include "../../../isEngine/system/function/GameKeyData.h"
#include "../../objects/gamelevel/Player.h"
#include "../../objects/gamelevel/Obstacle.h"
#include "../../objects/gamelevel/HUD.h"
#include "../../objects/gamelevel/Bonus.h"
#include "../../objects/widgets/GameDialog.h"
#include "../../language/GameLanguage.h"

class GameLevel : public is::GameDisplay
{
public:
   GameLevel(sf::RenderWindow &window, sf::View &view, sf::RenderTexture &surface, GameSystemExtended
&gameSysExt);
   void step();
   void draw();
   bool loadResources();

private:
   void gamePlay();
   void updateObjObstacleList();
   void updateObjBonusList();
   void playerLose();
   void updateObjPlayer();
   void updateBackground();

private:
   std::vector<std::shared_ptr<Obstacle>> m_obstacleList;
```

```
    std::vector<std::shared_ptr<Bonus>> m_bonusList;
    sf::Texture m_texButtonClose, m_texPlayer, m_texObstacle, m_texBonus, m_texDialog, m_texJoystick, m_texLevelBg;
    sf::Sprite m_sprLevelBg, m_sprButtonClose;
    sf::Text m_txtGameInfo;
    sf::SoundBuffer m_sbHurt, m_sbLose, m_sbHaveBonus;
    sf::Sound m_sndHurt, m_sndLose, m_sndHaveBonus;
    sf::Music m_mscLevel;
    GameKeyData m_gameKey;
    is::GameTime m_gameTime;
    GameDialog m_gameDialog;
    Player m_player;
    HUD m_gameHud;
    int m_timeCreateOstacle, m_timeCreateBonus;
};
```

- **Explanation**

Declaration of the class that represents the level.

- **std::vector**<**std::shared_ptr**<Obstacle>> m_obstacleList
  std::vector<**std::shared_ptr**<Bonus>> m_bonusList

Source Code

Container of Bonus and Obstacle objects.

- GameKeyData m_gameKey

Source Code

Object that allows to manage the game commands to control the player: keyboard key, mouse and Virtual Game Pad.

- **is::**GameTime m_gameTime;

Source Code

Level chronometer.

- sf::Text m_txtGameInfo

Source Code

Displays a message and the player's score when he loses the game.

- int m_timeCreateOstacle, m_timeCreateBonus

Source Code

Counter variable (in millisecond) to generate random objects in the scene.

4.6.2 Implementation

4.6.2.1 GameLevel

```
GameLevel::GameLevel(sf::RenderWindow &window, sf::View &view, sf::RenderTexture &surface,
GameSystemExtended &gameSysExt):
    GameDisplay(window, view, surface, gameSysExt, sf::Color::White),
    m_gameKey(this),
    m_gameDialog(this),
    m_player(m_gameKey),
    m_gameHud(*this, m_gameTime, m_player),
    m_timeCreateOstacle(59 * is::choose(2, 3, 5)),
    m_timeCreateBonus(59 * is::choose(2, 4, 9))
```

```
{}
```

➢ **Explanation**

We define a default time for counters that allow to generate objects in random ways in the level.

4.6.2.2 loadResources

```
bool GameLevel::loadResources()
{
   if (!GameDisplay::loadParentResources()) return false;

   // load buffers
   if (!m_sbHurt.loadFromFile(is::GameConfig::SFX_DIR + "hurt.ogg"))          return false;
   if (!m_sbLose.loadFromFile(is::GameConfig::SFX_DIR + "lose.ogg"))          return false;
   if (!m_sbHaveBonus.loadFromFile(is::GameConfig::SFX_DIR + "have_bonus.ogg")) return false;

   // sound
   m_sndHurt.setBuffer(m_sbHurt);
   m_sndLose.setBuffer(m_sbLose);
   m_sndHaveBonus.setBuffer(m_sbHaveBonus);

   // GUI resources
   if (!m_texButtonClose.loadFromFile(is::GameConfig::GUI_DIR + "button_close.png")) return false;
   if (!m_texDialog.loadFromFile(is::GameConfig::GUI_DIR + "dialog_box.png"))      return false;
   if (!m_texJoystick.loadFromFile(is::GameConfig::GUI_DIR + "game_pad.png")) return false;
   m_gameKey.loadResources(m_texJoystick);

   // sprites
   if (!m_texPlayer.loadFromFile(is::GameConfig::SPRITES_DIR + "player.png"))  return false;
   if (!m_texBonus.loadFromFile(is::GameConfig::SPRITES_DIR + "bonus.png"))    return false;
   if (!m_texObstacle.loadFromFile(is::GameConfig::SPRITES_DIR + "obstacle.png")) return false;

   // background
   if (!m_texLevelBg.loadFromFile(is::GameConfig::TILES_DIR + "level_bg.png"))   return false;

   // CREATION OF THE LEVEL
   // place the player
   m_player.loadResources(m_texPlayer);
   m_player.setPosition(32.f, 220.f);

   // set time
   m_gameTime.setTimeValue(0, 29, 59);

   // create game over text
   is::createText(m_fontMsg, m_txtGameInfo, "", 240.f, 200.f, false, 24);

   // create close button
   is::createSprite(m_texButtonClose, m_sprButtonClose, sf::IntRect(0, 0, 32, 32), sf::Vector2f(600.f, 16.f),
sf::Vector2f(0.f, 0.f), true);

   // build background
   // We enlarge the size of the background to make it repeat in game endlessly
   is::createSprite(m_texLevelBg, m_sprLevelBg, sf::IntRect(0, 0, m_texLevelBg.getSize().x * 2.5, 480), sf::Vector2f(0.f,
0.f), sf::Vector2f(0.f, 0.f), true);

   // load HUD resources
   m_gameHud.setPosition(m_viewX, m_viewY);
   m_gameHud.loadResources(m_fontSystem);

   // load Dialog Box resources
```

```
  m_gameDialog.loadResources(m_texDialog, m_fontSystem);
  m_gameDialog.setDialog(GameDialog::DialogIndex::DIALOG_PLAYER_MOVE);

  // load level music
  m_mscLevel.openFromFile(is::GameConfig::MUSIC_DIR + "world_1_music.ogg");
  m_mscLevel.setLoop(true);
  m_mscLevel.play();
  return true;
}
```

➢ **Explanation**

Method to load the resources of the game (music, sounds, sprites, etc.), define the parameters for creating certain objects and position the objects in the scene.

- m_gameTime.setTimeValue(0, 29, 59)

Source Code

Set the chronometer time.

- is::createSprite(m_texLevelBg, m_sprLevelBg, sf::IntRect(0, 0, m_texLevelBg.getSize().x * 2.5, 480), sf::Vector2f(0.f, 0.f), sf::Vector2f(0.f, 0.f), true)

Source Code

Allows to create the background of the level by repeating its size over the length 2.5 times. This allows to scroll the background infinitely on the x-axis.

- m_gameDialog.setDialog(GameDialog::DialogIndex::DIALOG_PLAYER_MOVE)

Source Code

Displays the dialog box with the message that shows how to control the player.

4.6.2.3 updateObjPlayer

```
void GameLevel::updateObjPlayer()
{
  m_player.step(DELTA_TIME);
}
```

➢ **Explanation**

Method that updates the player.

4.6.2.4 playerLose

```
void GameLevel::playerLose()
{
  m_mscLevel.stop();
  m_gameSysExt.playSound(m_sndLose);
  m_txtGameInfo.setString(is::lang::msg_game_over[m_gameSysExt.m_gameLanguage] +
                 is::numToStr(m_gameSysExt.m_currentScore) + "\n" +
                 is::lang::msg_clic_restart[m_gameSysExt.m_gameLanguage]);
  m_player.setIsActive(false);
}
```

➢ **Explanation**

This method allows to stop the game when the player is no longer healthy. It stops the game music, defines the game over text with the player score that will be displayed and deactivates the player (which means that he lost).

4.6.2.5 updateObjObstacleList

```
void GameLevel::updateObjObstacleList()
{
  WITH(m_obstacleList.size())
  {
    if (is::instanceExist(m_obstacleList[_I]))
    {
      // apply player acceleration on the object
      m_obstacleList[_I]->moveX(-m_player.getSpeed() * DELTA_TIME);

      // If the player touches the obstacle, his health is removed. if he is no longer healthy then game over
      if (m_player.placeMetting(0, 0, m_obstacleList[_I]))
      {
        if (m_player.getHealth() > 1)
        {
          m_gameSysExt.playSound(m_sndHurt);
          m_player.setIsHurt(30.f); // make blink
          m_player.addHealth(-1);
          m_obstacleList[_I].reset();
          break;
        }
        else playerLose();
      }
      m_obstacleList[_I]->step(DELTA_TIME); // update object

      // We destroy the object when it leaves to the left of the view
      if (m_obstacleList[_I]->getX() < -32.f)
      {
        m_gameSysExt.m_currentScore += m_obstacleList[_I]->getScorePoint(); // add score point
        m_obstacleList[_I].reset();
      }
    }
  }
}
```

➢ **Explanation**

Method that updates the Obstacles. Inside the **WITH** loop we check if the player is in collision with the object, if yes we remove the obstacle and we remove a health, but if he no longer has health then the game is over.

- **if** (m_obstacleList[_I]->getX() < -32.f)

Source Code

Lets know if the object is out on the left side of the window. If so we destroy it to free the memory space and add a score point to the player.

4.6.2.6 updateObjBonusList

```
void GameLevel::updateObjBonusList()
{
  WITH(m_bonusList.size())
  {
    if (is::instanceExist(m_bonusList[_I]))
    {
      // apply player acceleration on the object
      m_bonusList[_I]->moveX(-m_player.getSpeed() * DELTA_TIME);
      if (m_player.placeMetting(0, 0, m_bonusList[_I]) && m_bonusList[_I]->getStep() == 0)
      {
        m_gameSysExt.m_currentBonus++;
        m_gameTime.addTimeValue(0, 15, 0); // add 10 second
```

```
            m_gameSysExt.m_currentScore += m_bonusList[_I]->getScorePoint(); // add score point
            m_gameSysExt.playSound(m_sndHaveBonus);
            m_bonusList[_I]->addStep();
         }
         m_bonusList[_I]->step(DELTA_TIME); // update object

         // destruction
         if (m_bonusList[_I]->isDestroyed() || m_bonusList[_I]->getX() < -32.f) m_bonusList[_I].reset();
      }
   }
}
```

➢ **Explanation**

Method that updates the Bonuses. Inside the **WITH** loop we check if the player is in collision with the object if yes we add a score point and we increase the time of the level.

Then we check if the Bonus is out on the left side of the window, if yes we destroy it to free the memory space.

- m_bonusList[_I]->getStep() == 0

Source Code

Allows to execute actions in the collision once and to be able to delete the Bonus later.

4.6.2.7 updateBackground

```
void GameLevel::updateBackground()
{
   // Allows you to repeat the background endlessly
   if (is::getSFMLObjX(m_sprLevelBg) < -static_cast<float>(m_texLevelBg.getSize().x)) is::setSFMLObjX(m_sprLevelBg,
0.f);
   is::moveSFMLObjX(m_sprLevelBg, -(1.f * is::VALUE_CONVERSION + m_player.getSpeed()) * DELTA_TIME);
}
```

➢ **Explanation**

This method updates the background by simulating an infinite scroll animation.

4.6.2.8 gamePlay

```
void GameLevel::gamePlay()
{
   // GAME CONTROLLER
   if (!m_gameSysExt.isPressed()) m_gameSysExt.m_keyIsPressed = false;
   m_gameKey.step(DELTA_TIME);

   // LEVEL CHRONOMETER
   if (m_gameTime.getTimeValue() != 0) m_gameTime.step(DELTA_TIME, is::VALUE_CONVERSION, is::VALUE_TIME);
else playerLose();

   // We create a second counter which creates objects randomly
   m_timeCreateOstacle -= is::getMSecond(DELTA_TIME);
   if (m_timeCreateOstacle == 0)
   {
      m_obstacleList.push_back(std::shared_ptr<Obstacle>(new Obstacle(m_texObstacle, m_viewW + 10.f,
m_player.getY())));
      m_timeCreateOstacle = 59 * is::choose(3, 10, 3, 5);
   }
   m_timeCreateBonus -= is::getMSecond(DELTA_TIME);
   if (m_timeCreateBonus == 0)
   {
      m_bonusList.push_back(std::shared_ptr<Bonus>(new Bonus(m_texBonus, m_viewW + 10.f, m_player.getY())));
      m_timeCreateBonus = 59 * is::choose(3, 10, 20, 25);
   }

   // OBSTACLE
   updateObjObstacleList();

   // BONUS
   updateObjBonusList();

   // PLAYER
   updateObjPlayer();

   // HUD
   m_gameHud.step(DELTA_TIME);

   // BACKGROUND
   updateBackground();
}
```

➢ **Explanation**

Sub-function **step()**, it manages the level stopwatch, game control, the counters that generate the Bonus and Obstacle objects and to call the functions that update the game play objects.

4.6.2.9 step

```
void GameLevel::step()
{
    DELTA_TIME = getDeltaTime();
    updateTimeWait(DELTA_TIME);

    // even loop
    while (m_window.pollEvent(m_event))
    {
        controlEventFocusClosing();
        if (m_event.type == sf::Event::KeyReleased)
        {
            if (m_event.key.code == is::GameConfig::KEY_CANCEL) m_keyBackPressed = true;
        }
    }

    // if the window is activated launch the game
    if (m_windowIsActive)
    {
        // If the player loses and clicks on the screen then restart the level
        if (m_gameSysExt.isPressed() && !m_player.getIsActive())
        {
            m_gameSysExt.playSound(m_sndSelectOption);
            m_gameSysExt.m_launchOption = is::DisplayOption::RESTART_LEVEL;
            m_isRunning = false;
        }

        // if player clicks on close button sprite then quit game
        if (mouseCollision(m_sprButtonClose) && m_gameSysExt.isPressed())
        {
            m_mscLevel.stop();
            m_gameSysExt.playSound(m_sndSelectOption);
            m_gameSysExt.m_launchOption = is::DisplayOption::MAIN_MENU;
            m_isRunning = false;
        }
        if (!m_gameDialog.showDialog())
    {
      if (m_player.getIsActive()) gamePlay();
    }
    else
    {
      if (!mouseCollision(m_gameDialog.getSprite()) && m_gameSysExt.isPressed()) m_gameSysExt.m_keyIsPressed =
true;
      m_gameDialog.setPosition(m_viewX, m_viewY + 32.f);
    }
    m_gameDialog.step(DELTA_TIME);
  }
}
```

➢ **Explanation**

This method manages the event part associated with the scene, the dialog box for the tutorial and the options which allow to start the level again or leave it for another one.

- m_gameSysExt.m_launchOption = **is::**DisplayOption::MAIN_MENU
- m_gameSysExt.m_launchOption = **is::**DisplayOption::RESTART_LEVEL

Source Code

The action that will be performed on a scene.

4.6.2.10 draw

```
void GameLevel::draw()
{
  // draw background
  m_surface.draw(m_sprLevelBg);

  // draw bonus
  WITH(m_bonusList.size())
  {
    if (is::instanceExist(m_bonusList[_I]))
    {
      if (m_bonusList[_I]->inViewRec(*this)) m_bonusList[_I]->draw(m_surface);
    }
  }

  // draw blocks
  WITH(m_obstacleList.size())
  {
    if (is::instanceExist(m_obstacleList[_I]))
    {
      if (m_obstacleList[_I]->inViewRec(*this)) m_obstacleList[_I]->draw(m_surface);
    }
  }
  m_player.draw(m_surface);
  m_gameHud.draw(m_surface);

  // draw close button
  m_surface.draw(m_sprButtonClose);
  if (!m_player.getIsActive()) m_surface.draw(m_txtGameInfo);

  // draw dialog box
  m_gameDialog.draw(m_surface);
}
```

➢ **Explanation**

Displays objects from the scene.

5. Integration and use of scenes in Activity

```
#include <memory>
#include "SwooshFiles.h"
#include "../scenes/GameMenu/GameMenu.h"
#include "../scenes/GameLevel/GameLevel.h"

using namespace swoosh::intent;

class GameActivity : public Activity
{
private:
  std::shared_ptr<is::GameDisplay> m_gameScene;

public:
  GameActivity(ActivityController& controller, GameSystemExtended &gameSysExt) :
    Activity(&controller)
  {
    m_gameScene = nullptr;
    switch (gameSysExt.m_launchOption)
    {
    case is::DisplayOption::MAIN_MENU:
      m_gameScene = std::shared_ptr<is::GameDisplay>(new GameMenu(controller.getWindow(),
                                                    getView(),
                                                    *(this->controller->getSurface()),
                                                    gameSysExt));
    break;

    case is::DisplayOption::GAME_LEVEL:
        m_gameScene = std::shared_ptr<is::GameDisplay>(new GameLevel(controller.getWindow(),
                                                    getView(),
                                                    *(this->controller->getSurface()),
                                                    gameSysExt));
    break;

    default:
        is::showLog("Error : Scene not found !");
        std::terminate();
    break;
    }

   if (!m_gameScene->loadResources())
   {
        is::showLog("Error in loadResources function !");
        std::terminate();
   }
    this->setBGColor(m_gameScene->getBgColor());
  }

  virtual void onUpdate(double elapsed)
  {
    if (m_gameScene->isRunning()) m_gameScene->step();
    else
    {
      switch (m_gameScene->getGameSystem().m_launchOption)
      {
        case is::DisplayOption::MAIN_MENU:
          {
            using transition = segue<VerticalSlice, sec<2>>;
            using action = transition::to<GameActivity>;
```

```
                getController().replace<action>(m_gameScene->getGameSystem());
            }
          break;

          case is::DisplayOption::GAME_LEVEL:
            {
                using transition = segue<VerticalSlice, sec<2>>;
                using action = transition::to<GameActivity>;
                getController().replace<action>(m_gameScene->getGameSystem());
            }
          break;

          case is::DisplayOption::RESTART_LEVEL : // restart level (when player loses)
            m_gameScene->getGameSystem().initData(false);
            m_gameScene->getGameSystem().m_launchOption = is::DisplayOption::GAME_LEVEL;
            using transition = segue<BlackWashFade>;
            using action = transition::to<GameActivity>;
            getController().replace<action>(m_gameScene->getGameSystem());
          break;

          default:
            is::showLog("Error : Scene not found !");
            std::terminate();
          break;
        }
      }
    }

    virtual void onDraw(sf::RenderTexture& surface)
    {
      m_gameScene->drawScreen();
    }

    virtual void onStart() {}
    virtual void onLeave(){}
    virtual void onExit(){}
    virtual void onEnter(){}
    virtual void onResume(){}
    virtual void onEnd() {}
};
```

- **Explanation**
- **#include "../scenes/GameMenu/GameMenu.h"**
  #include "../scenes/GameLevel/GameLevel.h"

Source Code

Allows to include the two scenes in order to use them in the **Activity** class.

- std::shared_ptr<is::GameDisplay> m_gameScene;

Source Code

Represents the instance that will store the scene to be executed. ***Please note this is a variable that adapts to the scene!***

-
  ```
  case is::DisplayOption::MAIN_MENU:
  m_gameScene = std::shared_ptr<is::GameDisplay>(new GameMenu(controller.getWindow(),
                                                                 getView(),
                                                                 *(this->controller->getSurface()),
                                                                 gameSysExt));
  ```

Source Code

Launches the Main Menu scene. If the **switch (m_gameScene->getGameSystem().m_launchOption)** is equivalent to **case is::DisplayOption::MAIN_MENU.**

- `if (m_gameScene->isRunning()) m_gameScene->step();`

Source Code

Launches the **step()** part (content update) of a scene.

-
  ```
  using transition = segue<VerticalSlice, sec<2>>;
  using action = transition::to<GameActivity>;
  getController().replace<action>(m_gameScene->getGameSystem());
  ```

Source Code

These instructions allow to pass from one scene to another by making a transition effect (Swoosh).

Remember that you can determine the scene that will be changed by another one by: **switch (m_gameScene->getGameSystem().m_launchOption**) and the **case is::Displayoption:: scene_name:**

Click here for more information on using the SWOOSH library functions.

- `m_gameScene->drawScreen();`

Source Code

Launches the **draw()** (content display) part of a scene.

6. Improvement

There are still a lot of features that we can bring to this mini game, here are some of them:

- Avoid the player leaving the screen when moving it
- An interface in the Main Menu which allows you to change the language of the game
- An interface in the Main Menu which enables / disables the game sound
- Increase the speed of Obstacles and Bonuses as the score increases
- Add a button to pause the game
- Etc.

Now, it's your turn to play!