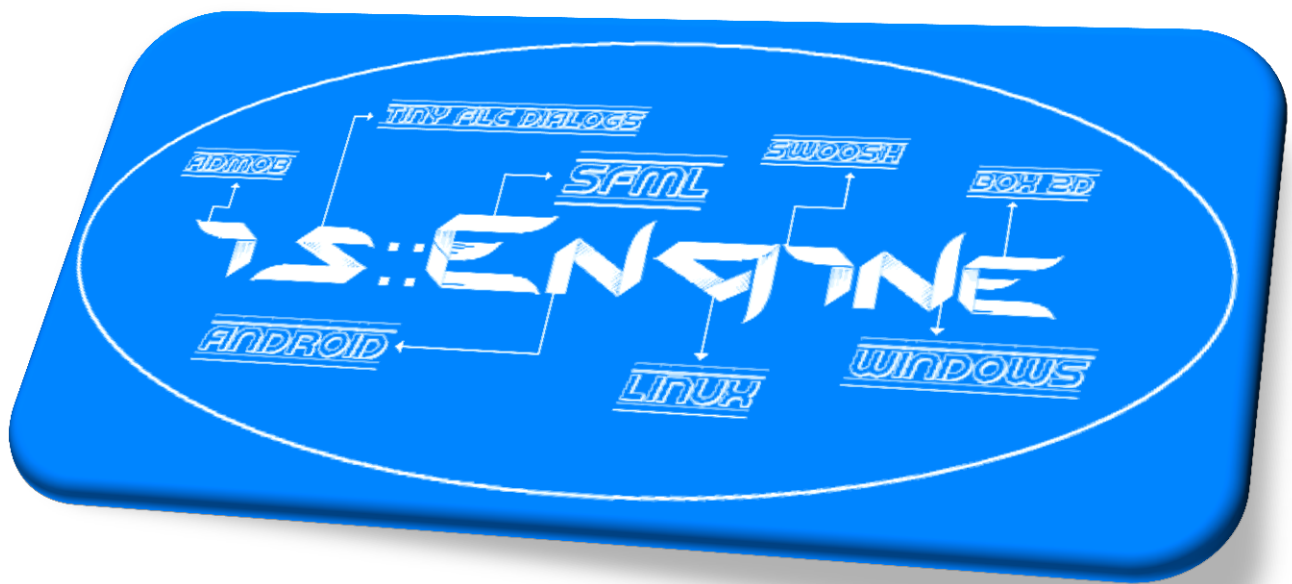


# Guide utilisateur de **is::Engine v2.0**



## Contenu

Guide de démarrage.....	14
1. Introduction.....	14
2. A propos du moteur.....	14
3. Structure du moteur .....	14
3.1 app_src .....	14
3.2 isEngine .....	15
3.3 data .....	15
3.4 Fichier main.cpp.....	15
3.4.1 main.....	15
3.4.2 game.play.....	15
3.4.3 game.basicSFMLmain .....	15
Display .....	15
1. class GameDisplay .....	15
2. Les méthodes publiques.....	16
2.1 GameDisplay .....	16
2.2 setAdmob.....	16
2.3 rewardVideoStep .....	16
2.4 step .....	16
2.5 drawScreen.....	16
2.6 showTempLoading.....	16
2.7 loadParentResources .....	16
2.8 loadResources .....	17
2.9 isRunning.....	17
2.10 getView .....	17
2.11 getRenderWindow .....	17
2.12 getRenderTexture .....	17
2.13 getGameSystem .....	17
2.14 getSndCancel.....	17
2.15 getSndSwitch .....	18
2.16 getSndSelectOption.....	18
2.17 getDeltaTime.....	18
2.18 getViewX.....	18
2.19 getViewY .....	18
2.20 getViewW .....	18
2.21 getViewH.....	18
2.22 getBgColor.....	18
2.23 mouseCollision.....	19
3. Éléments protégés .....	20
3.1 enum MsgAnswer .....	20
3.2 controlEventFocusClosing.....	20
3.3 showMessageBox.....	20
3.4 updateMsgBox.....	20
3.5 updateTimeWait.....	21

3.6	drawMsgBox.....	21
3.7	setOptionIndex.....	21
3.8	setTextAnimation .....	21
3.9	setView.....	21
	Entités.....	21
1.	class MainObject.....	21
2.	Les éléments publics de MainObjet.....	22
2.1	MainObject .....	22
2.2	instanceNumber .....	22
2.3	inViewRec.....	22
2.4	setXStart.....	22
2.5	setYStart.....	22
2.6	setXPrevious.....	22
2.7	setYPrevious.....	23
2.8	setStartPosition .....	23
2.9	setX .....	23
2.10	setY .....	23
2.11	moveX.....	23
2.12	moveY.....	23
2.13	setPosition.....	23
2.14	setSpriteScale .....	23
2.15	setSpeed .....	24
2.16	setHsp.....	24
2.17	setVsp .....	24
2.18	setAngularMove .....	24
2.19	setImageXscale.....	24
2.20	setImageYscale.....	24
2.21	setImageScale .....	24
2.22	setImageAngle.....	24
2.23	setXOffset.....	25
2.24	setYOffset.....	25
2.25	setXYOffset.....	25
2.26	setTime .....	25
2.27	setImageAlpha.....	25
2.28	setImageIndex.....	25
2.29	setMaskW .....	25
2.30	setMaskH .....	25
2.31	setIsActive.....	26
2.32	updateCollisionMask.....	26
2.33	centerCollisionMask.....	26
2.34	updateSprite.....	26
2.35	draw .....	26
2.36	getMask.....	26

2.37	getX.....	27
2.38	getY.....	27
2.39	getXStart.....	27
2.40	getYStart.....	27
2.41	getXPrevious.....	27
2.42	getYPrevious.....	27
2.43	distantToPoint.....	27
2.44	distantToObject.....	27
2.45	pointDirection.....	28
2.46	pointDirectionSprite.....	28
2.47	getSpeed.....	28
2.48	getHsp.....	28
2.49	getVsp.....	28
2.50	getFrame.....	29
2.51	getFrameStart.....	29
2.52	getFrameEnd.....	29
2.53	getImageXscale.....	29
2.54	getImageYscale.....	29
2.55	getImageScale.....	29
2.56	getImageAngle.....	29
2.57	getXOffset.....	29
2.58	getYOffset.....	30
2.59	getTime.....	30
2.60	getInstanceId.....	30
2.61	getMaskWidth.....	30
2.62	getMaskHeight.....	30
2.63	getIsActive.....	30
2.64	getImageAlpha.....	30
2.65	getImageIndex.....	30
2.66	getSpriteWidth.....	31
2.67	getSpriteHeight.....	31
2.68	getSpriteX.....	31
2.69	getSpriteY.....	31
2.70	getSpriteCenterX.....	31
2.71	getSpriteCenterY.....	31
2.72	placeMetting.....	31
2.73	getSprite.....	32
2.74	setFrame.....	32
3.	Autres fonctions de MainObject.....	32
3.1	instanceExist.....	32
3.2	operator().....	32
3.3	sortObjArray.....	32
3.4	operator>.....	33

3.5	operator< .....	33
	Les formes pour les masques collisions .....	33
1.	class Rectangle .....	33
2.	class Point .....	33
3.	class Line .....	33
	Les Classes Parentes de MainObject .....	34
1.	class DepthObject .....	34
1.1	enum Depth .....	34
1.2	DepthObject .....	34
1.3	setDepth .....	35
1.4	getDepth .....	35
2.	class Destructible .....	35
2.1	Destructible .....	35
2.2	setDestroyed .....	35
2.3	isDestroyed .....	35
3.	class Visibility .....	35
3.1	Visibility .....	35
3.2	setVisible .....	36
3.3	getVisible .....	36
4.	class Health .....	36
4.1	Health .....	36
4.2	setHealth .....	36
4.3	setMaxHealth .....	36
4.4	addHealth .....	37
4.5	getHealth .....	37
4.6	getMaxHealth .....	37
5.	class HurtEffect .....	37
5.1	HurtEffect .....	37
5.2	hurtStep .....	37
5.3	setIsHurt .....	37
5.4	getIsHurt .....	37
6.	class ScorePoint .....	38
6.1	ScorePoint .....	38
6.2	setScorePoint .....	38
6.3	getScorePoint .....	38
7.	class Step .....	38
7.1	Step .....	38
7.2	setStep .....	38
7.3	addStep .....	39
7.4	reduceStep .....	39
7.5	getStep .....	39
	Admob .....	39
1.	class AdmobManager .....	39

2.	Les méthodes publiques .....	39
2.1	AdmobManager .....	39
2.2	loadBannerAd .....	39
2.3	showBannerAd .....	39
2.4	hideBannerAd .....	40
2.5	loadRewardVideo .....	40
2.6	updateSFMLApp .....	40
2.7	checkAdObjInit .....	40
2.8	checkAdRewardObjReinitialize .....	40
3.	Autres Fonction d'AdmobManager .....	40
3.1	ProcessEvents & WaitForFutureCompletion .....	40
3.2	checkAdState .....	40
	Temps .....	41
1.	class GameTime .....	41
2.	Les methods publiques de GameTime .....	41
2.1	GameTime .....	41
2.2	step .....	41
2.3	addTimeValue .....	41
2.4	setTimeValue .....	41
2.5	setMSecond .....	42
2.6	getTimeString .....	42
2.7	getTimeValue .....	42
2.8	getMinute .....	42
2.9	getSecond .....	42
2.10	getMSecond .....	42
2.11	operator= .....	42
2.12	operator<< .....	42
3.	Autres Fonctions de GameTime .....	43
	Commande du jeu .....	43
1.	class GameKeyData .....	43
2.	Les éléments de GameKeyData .....	43
2.1	enum VirtualKeyIndex .....	43
2.2	GameKeyData .....	43
2.3	loadResources .....	43
2.4	step .....	44
2.5	draw .....	44
2.6	m_keyPausePressed .....	44
2.7	m_keyLeftPressed .....	44
2.8	m_keyRightPressed .....	44
2.9	m_keyUpPressed .....	44
2.10	m_keyDownPressed .....	44
2.11	m_keyAPressed .....	44
2.12	m_keyBPressed .....	45

2.13	m_keyAUsed .....	45
2.14	m_keyBUsed .....	45
2.15	m_disableAllKey .....	45
2.16	m_hideGamePad .....	45
2.17	m_keyboardA .....	45
2.18	m_keyboardB .....	45
2.19	m_keyboardLeft .....	45
2.20	m_keyboardRight .....	46
2.21	m_keyboardUp .....	46
2.22	m_keyboardDown .....	46
2.23	m_moveKeyPressed .....	46
2.24	m_actionKeyPressed .....	46
2.25	keyLeftPressed .....	46
2.26	keyRightPressed .....	46
2.27	keyUpPressed .....	46
2.28	keyDownPressed .....	47
2.29	keyAPressed .....	47
2.30	keyBPressed .....	47
2.31	virtualKeyPressed .....	47
3.	Autres fonctions de GameKeyData .....	47
	Game Système .....	47
1.	class GameSystem .....	47
2.	Les éléments de GameSystem .....	48
2.1	enum ValidationButton .....	48
2.2	GameSystem .....	48
2.3	isPressed .....	48
2.4	keyIsPressed .....	49
2.5	fileExist .....	49
2.6	playSound .....	49
2.7	stopSound .....	49
2.8	useVibrate .....	50
2.9	saveConfig .....	50
2.10	loadConfig .....	50
2.11	savePadConfig .....	50
2.12	loadPadConfig .....	50
2.13	m_disableKey .....	50
2.14	m_enableSound .....	50
2.15	m_enableMusic .....	50
2.16	m_enableVibrate .....	51
2.17	m_keyIsPressed .....	51
2.18	m_firstLaunch .....	51
2.19	m_validationMouseKey .....	51
2.20	m_validationKeyboardKey .....	51

2.21	m_gameLanguage .....	51
2.22	m_padAlpha .....	51
	Game System Extended .....	52
1.	class GameSystemExtended .....	52
2.	Les éléments de GameSystemExtended .....	52
2.1	GameSystemExtended .....	52
2.2	enum DisplayOption .....	52
2.3	initSystemData .....	52
2.4	initProgress .....	52
2.5	initData .....	52
2.6	saveData .....	53
2.7	loadData .....	53
2.8	m_launchOption .....	53
2.9	game play variables .....	53
	Game Function .....	53
1.	Fonction Générale .....	53
1.1	VALUE_CONVERSION .....	53
1.2	WITH .....	54
1.3	w_chart_tToStr .....	54
1.4	strToWStr .....	54
1.5	numToStr .....	54
1.6	strToNum .....	54
1.7	numToWStr .....	54
1.8	writeZero .....	55
1.9	getMSecond .....	55
1.10	showLog .....	55
1.11	arraySize .....	55
1.12	choose .....	55
1.13	setVarLimit .....	56
1.14	isIn .....	56
1.15	isBetween .....	56
1.16	isCrossing .....	56
1.17	side .....	56
1.18	sign .....	56
1.19	pointDirection .....	57
1.20	pointDistance .....	57
1.21	radToDeg .....	57
1.22	degToRad .....	57
1.23	lengthDirX .....	57
1.24	lengthDirY .....	57
1.25	increaseVar .....	57
1.26	decreaseVar .....	58
1.27	collisionTest .....	58



2.	Fonction sur les objets SFML.....	58
2.1	getSFMLObjAngle .....	58
2.2	getSFMLObjXScale.....	58
2.3	getSFMLObjYScale.....	58
2.4	getSFMLObjWidth .....	58
2.5	getSFMLObjHeight .....	59
2.6	getSFMLObjOriginX .....	59
2.7	getSFMLObjOriginY .....	59
2.8	getSFMLObjX.....	59
2.9	getSFMLObjY .....	59
2.10	setSFMLObjAngle.....	60
2.11	setSFMLObjRotate.....	60
2.12	setSFMLObjScaleX_Y.....	60
2.13	setSFMLObjScale .....	60
2.14	setSFMLObjOrigin .....	60
2.15	setSFMLObjX.....	60
2.16	setSFMLObjY .....	61
2.17	centerSFMLObj.....	61
2.18	centerSFMLObjX.....	61
2.19	centerSFMLObjY.....	61
2.20	setSFMLObjX_Y .....	61
2.21	moveSFMLObjX.....	62
2.22	moveSFMLObjY.....	62
2.23	setSFMLObjSize .....	62
2.24	setSFMLObjAlpha .....	62
2.25	setSFMLObjAlpha2.....	62
2.26	setSFMLObjColor .....	63
2.27	setSFMLObjFillColor.....	63
2.28	scaleAnimation .....	63
2.29	setFrame .....	63
2.30	setSFMLObjOutlineColor.....	63
2.31	setSFMLObjTexRec .....	64
2.32	setSFMLObjProperties .....	64
2.33	getSFMLSndState .....	64
2.34	collisionTestSFML .....	64
2.35	createRectangle .....	64
2.36	textStyleConfig.....	64
2.37	createWText.....	65
2.38	createText.....	65
2.39	createSprite .....	65
2.40	mouseCollision.....	65
3.	Autres fonctions.....	67
3.1	vibrate .....	67

3.2	openURL.....	67
3.3	setScreenLock.....	67
3.4	jstring2string.....	67
3.5	getDeviceId.....	67
	Bibliothèque externe.....	67
1.	Swoosh.....	67
2.	Tiny File Dialogs (uniquement pour Windows et Linux).....	68
2.1	class TinyDialogBox.....	68
2.2	tinyString.....	68
2.3	TINY_FILE_DIALOGBOX_PATH.....	68
2.4	enum FileDialogType.....	68
2.5	enum DialogType.....	68
2.6	enum IconType.....	69
2.7	enumDialogTypeToStr / enumIconTypeToStr.....	69
2.8	showDialogBox.....	69
2.9	showFileDialogBox.....	69
2.10	showFolderDialogBox.....	70
3.	Box 2D.....	70
	Game Engine.....	70
1.	class GameEngine.....	70
2.	Les méthodes de GameEngine.....	71
2.1	GameEngine.....	71
2.2	initEngine.....	71
2.3	play.....	71
2.4	basicSFMLmain.....	71
2.5	getRenderWindow.....	71
	Configuration du jeu.....	71
1.	Paramètre de la fenêtre.....	71
1.1	WINDOW_WIDTH.....	72
1.2	WINDOW_HEIGHT.....	72
1.3	VIEW_WIDTH.....	72
1.4	VIEW_HEIGHT.....	72
2.	Paramètre des boutons de validation.....	72
2.1	KEY_VALIDATION_MOUSE.....	72
2.2	KEY_VALIDATION_KEYBOARD.....	72
2.3	KEY_CANCEL.....	72
3.	Paramètre des touches du clavier.....	72
3.1	KEY_A.....	72
3.2	KEY_B.....	73
3.3	KEY_LEFT.....	73
3.4	KEY_RIGHT.....	73
3.5	KEY_UP.....	73
3.6	KEY_DOWN.....	73

4.	Les informations du jeu .....	73
4.1	MAJOR .....	73
4.2	MINOR .....	73
4.3	getGameVersion .....	73
4.4	GAME_NAME .....	74
4.5	GAME_AUTHOR .....	74
5.	Paramètre Admob.....	74
5.1	Id de la PUB.....	74
5.1.1	kAdMobAppID.....	74
5.1.2	kBannerAdUnit.....	74
5.1.3	kRewardedVideoAdUnit.....	74
5.2	Taille de la bannière de PUB.....	74
5.2.1	kBannerWidth.....	74
5.2.2	kBannerHeight.....	74
5.3	Publique cible de la PUB.....	75
5.3.1	kBirthdayDay.....	75
5.3.2	kBirthdayMonth .....	75
5.3.3	kBirthdayYear .....	75
5.3.4	kKeywords .....	75
6.	Chemin des fichiers ressources du jeu .....	75
6.1	GUI_DIR.....	75
6.2	FONT_DIR .....	75
6.3	SPRITES_DIR .....	75
6.4	TILES_DIR.....	76
6.5	SFX_DIR.....	76
6.6	MUSIC_DIR .....	76
7.	Nom du package du jeu (Android).....	76
8.	Chemin des fichiers de sauvegarde .....	76
8.1	GAME_DATA_FILE .....	76
8.2	CONFIG_FILE .....	76
8.3	GAME_PAD_FILE.....	76
	Activité .....	76
1.	class GameActivity.....	77
2.	Les éléments de GameActivity; .....	77
2.1	GameActivity.....	77
2.2	m_gameScene.....	77
2.3	onStart.....	77
2.4	onUpdate.....	77
2.5	onLeave.....	77
2.6	onExit.....	77
2.7	onEnter .....	77
2.8	onResume .....	78
2.9	onDraw .....	78

2.10	onEnd.....	78
	Niveau.....	78
1.	Les niveaux.....	78
2.	Intégration d'un niveau.....	78
3.	Les éléments pour gérer les niveaux.....	78
3.1	namespace level .....	78
3.2	enum LevelId .....	78
3.3	getLevelMap .....	79
	Langage du jeu.....	79
1.	Les langues.....	79
2.	Les éléments pour gérer les langues .....	79
2.1	namespace Lang.....	79
2.2	enum GameLanguage.....	80
	Boite de Dialogue du jeu .....	80
1.	class GameDialog.....	80
2.	Les éléments de GameDialog.....	80
2.1	GameDialog.....	80
2.2	enum DialogIndex.....	81
2.3	linkArrayToEnum .....	81
2.4	loadResources .....	82
2.5	step .....	82
2.6	setDialog.....	82
2.7	setMouseInCollison.....	82
2.8	draw .....	82
2.9	getDialogIndex.....	82
2.10	getMouseInCollison .....	82
2.11	showDialog .....	82
	Exemple de jeu .....	83
1.	Introduction.....	83
2.	Comment le jeu sera créé ?.....	83
2.1	Voici les éléments du moteur que le jeu utilisera .....	83
2.2	Les objets qui seront utilisés dans le jeu .....	83
2.3	Les rôles des objets.....	83
3.	Intégration des phrases dans le jeu .....	84
3.1	Création des phrases dans GameLanguage.h .....	84
3.2	Association de la boite de dialogue avec la phrase du jeu .....	85
4.	Création des classes du jeu.....	87
4.1	Classe Obstacle.....	87
4.1.1	Entête .....	87
4.1.2	Implémentation .....	87
4.1.2.1	Obstacle .....	87
4.1.2.2	step .....	87
4.2	Classe Bonus.....	88

4.2.1	Entête .....	88
4.2.2	Implémentation .....	88
4.2.2.1	Bonus .....	88
4.2.2.2	step .....	88
4.3	Classe Player .....	89
4.3.1	Entête .....	89
4.3.2	Implémentation .....	89
4.3.2.1	Player .....	89
4.3.2.2	loadResources .....	90
4.3.2.3	step .....	90
4.4	Classe HUD .....	91
4.4.1	Entête .....	91
4.4.2	Implémentation .....	92
4.4.2.1	HUD .....	92
4.4.2.2	loadResources .....	92
4.4.2.3	step .....	92
4.4.2.4	draw .....	93
4.5	Classe MainMenu .....	93
4.5.1	Entête .....	93
4.5.2	Implémentation .....	94
4.5.2.1	MainMenu .....	94
4.5.2.2	loadResources .....	94
4.5.2.3	componentsController .....	95
4.5.2.4	step .....	96
4.5.2.5	draw .....	98
4.6	Classe GameLevel .....	98
4.6.1	Entête .....	98
4.6.2	Implémentation .....	99
4.6.2.1	GameLevel .....	99
4.6.2.2	loadResources .....	100
4.6.2.3	updateObjPlayer .....	101
4.6.2.4	playerLose .....	101
4.6.2.5	updateObjObstacleList .....	102
4.6.2.6	updateObjBonusList .....	102
4.6.2.7	updateBackground .....	103
4.6.2.8	gamePlay .....	103
4.6.2.9	step .....	104
4.6.2.10	draw .....	105
5.	Intégration et utilisation des scènes dans Activity .....	106
6.	Amélioration .....	108

## 1. Introduction

Bienvenue dans le guide utilisateur du moteur de jeu **is::Engine v2.0**. Ce guide a pour objective de vous détailler comment l'API fonctionne. Ce n'est pas un tutoriel même s'il y a un exemple qui vous montre comment utiliser le moteur pour créer un jeu.

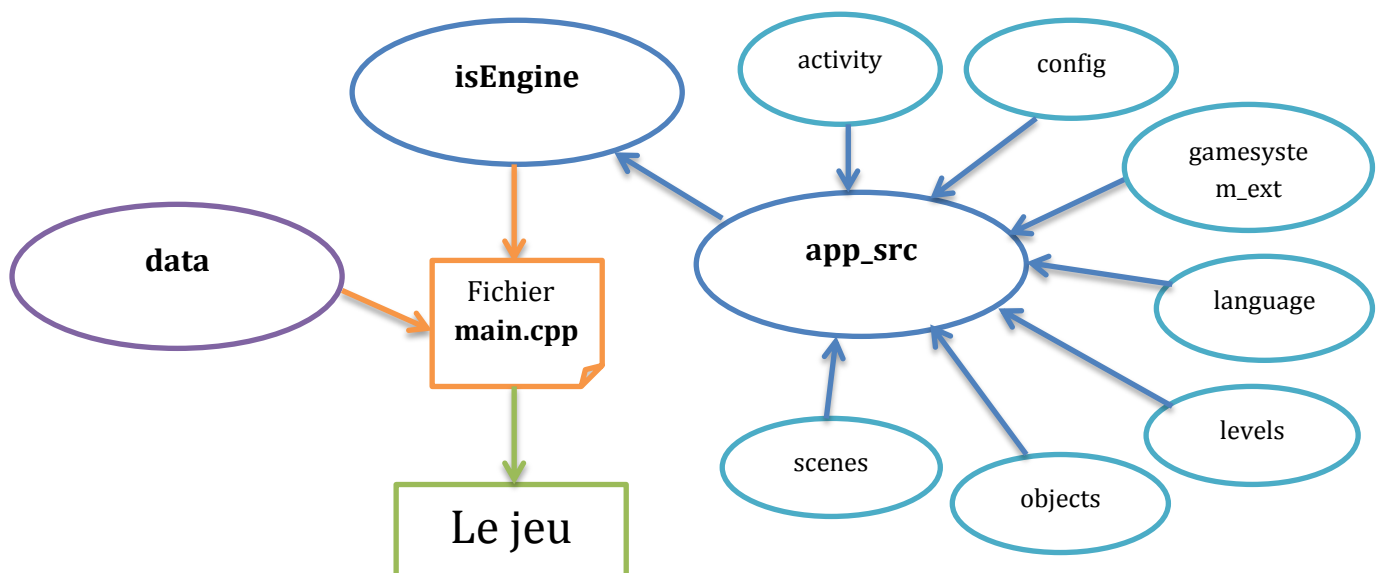
## 2. A propos du moteur

is::Engine est un outil qui se base sur les mécanismes de la bibliothèque SFML pour fonctionner. Donc si vous voulez bien utiliser cet outil il est vivement conseillé de connaître au minimum les bases de SFML. L'objectif de ce moteur est de vous offrir des fonctionnalités qui vous permettent de créer un jeu avec le plus de souplesse possible et de le porter facilement sur diverses plateformes (Windows, Linux, Android).

Le moteur est directement livré avec un IDE pour vous éviter les reconfigurations et de démarrer rapidement avec celui-ci. Notez que chaque IDE avec lequel le moteur est livré permet de porter votre projet sur une plateforme cible. Donc le projet Android Studio vous permet d'utiliser le moteur pour développer sous Android.

L'entête qui permet d'avoir accès à la bibliothèque est : **isEngine/core/GameEngine.h**.

## 3. Structure du moteur



### 3.1 app\_src

Repertoire qui contient le code source du jeu.

Description de ces sous-répertoires:

- **activity**: Contient la classe **Activity** ([cliquer ici 1 pour plus d'info](#)) qui permet de lancer les différentes scènes du jeu et d'assurer leurs interactions.
- **config**: Contient le fichier **GameConfig.cpp** ([cliquer ici 1 pour plus d'info](#)) qui permet de définir les paramètres généraux du jeu.
- **gamesystem\_ext**: Contient une classe dérivée ([cliquer ici 1 pour plus d'info](#)) de **GameSystem** ([cliquer ici 1 pour plus d'info](#)) qui permet de manipuler les données du jeu (sauvegarde, chargement, ...).
- **language**: Contient le fichier **GameLanguage.cpp** ([cliquer ici 1 pour plus d'info](#)) qui permet de gérer tout ce qui concerne les langues du jeu.

- **levels:** Contient les niveaux et le fichier **Level.h** (*cliquer ici 1 pour plus d'info*) qui permet de les intégrer dans le jeu.
- **objects:** Contient les objets qui seront utilisés dans les différentes scènes.
- **scenes:** Contient les différentes scènes du jeu (*cliquer ici 1 pour plus d'info*) (Introduction, Menu principal, ...).

### 3.2 isEngine

Repertoire qui contient le code source du moteur de jeu.

### 3.3 data

Repertoire des fichiers ressources du jeu (musiques, effets sonores, images, ...).

### 3.4 Fichier main.cpp

Ce fichier contient la fonction qui permet de lancer le programme.

#### 3.4.1 main

```
int main()
```

#### Code Source

La fonction principale qui permet de lancer le moteur de jeu. A l'intérieur d'elle vous trouverez **GameEngine game;** qui permet d'initialiser le moteur de jeu.

**Retourne 0** quand le programme c'est bien terminé et une autre valeur s'il y a une erreur lors de l'exécution.

#### 3.4.2 game.play

```
game.play()
```

#### Code Source

Permet d'utiliser la boucle principale du moteur qui permet de lancer les différentes scènes du jeu (Introduction, Menu principal, ...).

#### 3.4.3 game.basicSFMLmain

```
game.basicSFMLmain()
```

#### Code Source

Permet d'afficher une fenêtre SFML classique. Cette fonction vous permet d'utiliser votre propre boucle de rendu avec le moteur. Très utile si vous désirez utiliser un projet SFML déjà en cours de développement avec le moteur ou d'intégrer vos propres composants au moteur.

## Display

---

### 1. class GameDisplay

```
class GameDisplay;
```

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

#### Code Source

Classe abstraite qui permet de créer la scène d'un jeu. Une scène est un endroit où les objets du jeu prennent vie (Menu Principal, Niveau, ...). Cette classe vous offre des fonctions qui vous permettent de manipuler facilement la vue,

appliquer les événements de la fenêtre sur la scène, de faire des animations sur des textes et sprites, afficher des boîtes de dialogue, ...

## 2. Les méthodes publiques

### 2.1 GameDisplay

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

#### Code Source

Constructeur qui permet de créer un objet GameDisplay, elle prend en paramètre la fenêtre de l'application, la surface de la bibliothèque SWOOSH qui permet de faire des effets transitions, GameSystemExtended ([cliquer ici 1 pour plus d'info](#)) et la couleur de fond de la scène.

### 2.2 setAdmob

`virtual void setAdmob(AdmobManager *admob)`

#### Code Source

Permet d'intégrer le gestionnaire d'annonce (Admob) dans une scène.

### 2.3 rewardVideoStep

`virtual int rewardVideoStep()`

#### Code Source

Permet de lancer une annonce de type vidéo récompense.

**Retourne 1** si la vidéo récompense c'est bien lancé et **0** s'il y a une erreur (se produit souvent quand la requête de l'annonce n'a pas fonctionné).

### 2.4 step

`virtual void step() = 0`

#### Code Source

Méthode qui permet d'implémenter la partie où les objets de la scène sont mise à jour (déplacement des objets, détections de collision, ...).

### 2.5 drawScreen

`virtual void drawScreen()`

#### Code Source

Méthode qui permet d'implémenter la partie où les objets de la scène seront dessinés.

### 2.6 showTempLoading

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

#### Code Source

Permet d'afficher un faux écran de chargement (Utile pour faire des transitions dans la même scène).

**Paramètre time** représente la durée (en milliseconde) du chargement.

### 2.7 loadParentResources



**virtual bool** loadParentResources()

#### Code Source

Charge les ressources qui permettent d'afficher les boites de dialogues dans une scène.

Elle est généralement utilisée dans la fonction **loadResources** d'une scène.

### 2.8 loadResources

**virtual bool** loadResources() = 0

#### Code Source

Permet d'implémenter le chargement des ressources qui sont utilisés dans une scène.

### 2.9 isRunning

**virtual bool** isRunning() **const**

#### Code Source

**Retourne vrai** si une scène est en cours d'exécution et **faux** si non.

### 2.10 getView

**virtual sf::View&** getView() **const**

#### Code Source

**Retourne** la vue d'une scène.

### 2.11 getRenderWindow

**virtual sf::RenderWindow&** getRenderWindow()

#### Code Source

**Retourne** la fenêtre d'exécution de la scène.

### 2.12 getRenderTexture

**virtual sf::RenderTexture&** getRenderTexture() **const**

#### Code Source

**Retourne** la surface sur laquelle on dessine les objets d'une scène.

### 2.13 getGameSystem

**virtual GameSystemExtended&** getGameSystem()

#### Code Source

**Retourne** l'objet game system extended.

### 2.14 getSndCancel

**virtual sf::Sound&** getSndCancel()

#### Code Source

**Retourne** le son qui permet de simuler une annulation.

### 2.15      **getSndSwitch**

**virtual** sf::Sound& getSndSwitch()

#### **Code Source**

**Retourne** le son qui permet de simuler une modification.

### 2.16      **getSndSelectOption**

**virtual** sf::Sound& getSndSelectOption()

#### **Code Source**

**Retourne** le son qui permet de simuler une sélection (un clic).

### 2.17      **getDeltaTime**

**float** getDeltaTime()

#### **Code Source**

**Retourne** le temps d'exécution en seconde.

### 2.18      **getViewX**

**virtual float** getViewX() **const**

#### **Code Source**

**Retourne** la position x de la vue.

### 2.19      **getViewY**

**virtual float** getViewY() **const**

#### **Code Source**

**Retourne** la position y de la vue.

### 2.20      **getViewW**

**virtual float** getViewW() **const**

#### **Code Source**

**Retourne** la largeur de la vue.

### 2.21      **getViewH**

**virtual float** getViewH() **const**

#### **Code Source**

**Retourne** la hauteur de la vue.

### 2.22      **getBgColor**

**virtual** sf::Color& getBgColor()

#### **Code Source**

**Retourne** la couleur de fond de la scène.

## 2.23 mouseCollision

- Première forme

template <class T>

```
bool mouseCollision(T const &obj
    #if defined(__ANDROID__)
    , unsigned int finger = 0
    #endif
)
```

### Code Source

**Windows, Linux** : Détecte si le curseur de la souris est en collision avec un objet de la scène.

**Android** : Détecte si l'utilisateur touche un objet de la scène.

### Paramètre :

**obj** l'objet avec lequel on veut faire le test.

**finger** représente l'index du doigt.

**Retourne** **vrais** si il y a collision et **faux** si non.

### Exemple :

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

- Deuxième forme

template <class T>

```
bool mouseCollision(T const &obj, sf::RectangleShape &cursor
    #if defined(__ANDROID__)
    , unsigned int finger = 0
    #endif
)
```

### Code Source

**Windows, Linux** : Détecte si le curseur de la souris est en collision avec un objet de la scène.

**Android** : Détecte si l'utilisateur touche un objet de la scène.

### Paramètre :

**obj** l'objet avec lequel on veut faire le test.

**cursor** permet de récupérer la position du point de collision.

**finger** représente l'index du doigt.

**Retourne vrais** si il y a collision et **faux** si non.

**Exemple :**

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

### 3. Éléments protégés

#### 3.1 enum MsgAnswer

**enum** MsgAnswer;

Enumérateur	
YES	Réponse Oui
NO	Réponse Non

#### Code Source

Représente les réponses que l'utilisateur peut choisir par rapport à la boite de dialogue.

#### 3.2 controlEventFocusClosing

**void** controlEventFocusClosing()

#### Code Source

Gère les événements focus et fermeture de la fenêtre. *S'utilise dans une boucle d'événement !*

#### 3.3 showMessageBox

**template<class T>**

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

#### Code Source

Definie les paramètres et affiche la boite de dialogue.

#### Paramètre :

**msgBody** le message qui sera affiché à l'utilisateur.

**mbYesNo vrais** affiche une boite de dialogue de type OUI NON et **faux** affiche juste un bouton OK.

#### 3.4 updateMsgBox

**void** updateMsgBox(**float const** &DELTA\_TIME, sf::Color textDefaultColor = sf::Color::White, sf::Color textSelectedColor = sf::Color::Red)

#### Code Source

Met à jour les informations de la boite de dialogue.

#### Paramètre :

**textDefaultColor** couleur du text du message.

**textSelectedColor** couleur des textes des boutons.

### 3.5 updateTimeWait

```
void updateTimeWait(float const &DELTA_TIME)
```

#### Code Source

Met à jour le compteur qui permet de faire patienter l'utilisateur après le choix d'une option. Ceci permet d'éviter les choix en boucles.

### 3.6 drawMsgBox

```
void drawMsgBox()
```

#### Code Source

Affiche la boîte de dialogue.

### 3.7 setOptionIndex

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

#### Code Source

Permet de faire des animations sur des textes et de jouer un son quand on change une option.

### 3.8 setTextAnimation

- Première forme :

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

#### Code Source

Permet de faire une animation sur un texte et un sprite en fonction du choix d'une option.

- Deuxième forme :

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

#### Code Source

Permet de faire une animation sur un texte en fonction du choix d'une option.

### 3.9 setView

```
void setView()
```

#### Code Source

Met à jour la position de la vue dans la scène.

## Entités

---

### 1. class MainObject

```
class MainObject;
```

Entête : *isEngine/system/entity/MainObject.h*

## Code Source

Classe de base pour créer les objets (Personnage, Tuiles, Bouton, ...) qui seront utilisées dans les scènes. Elle vous offre des fonctions qui vous permettent de contrôler un objet (les déplacements, détections de collision entre les objets, calcul de distance, ...) et plein d'autres choses qui sont liées au game play du jeu.

### 2. Les éléments publics de MainObject

#### 2.1 MainObject

- Première forme

MainObject()

## Code Source

Constructeur par défaut de la classe.

- Deuxième forme

MainObject(float x, float y)

## Code Source

Constructeur qui permet d'initialiser l'objet avec un point de départ.

#### 2.2 instanceNumber

static int instanceNumber;

## Code Source

Retourne le nombre d'instance de la classe.

#### 2.3 inViewRec

virtual bool inViewRec(GameDisplay const &app, bool useTexRec = true)

## Code Source

Retourne  **vrais**  si l'objet se trouve dans le champ de vision de la vue,  **faux**  si non.

#### 2.4 setXStart

virtual void setXStart(float x)

## Code Source

Definie la position x de début.

#### 2.5 setYStart

virtual void setYStart(float y)

## Code Source

Definie la position y de début.

#### 2.6 setXPrevious

virtual void setXPrevious(float x)

## Code Source

Définie la position x précédente.

## 2.7 setYPrevious

**virtual void** setYPrevious(float y)

### Code Source

Définie la position y précédente.

## 2.8 setStartPosition

**virtual void** setStartPosition(float x, float y)

### Code Source

Définie la position x et y de début.

## 2.9 setX

**virtual void** setX(float x)

### Code Source

Définie la position x.

## 2.10 setY

**virtual void** setY(float y)

### Code Source

Définie la position y.

## 2.11 moveX

**virtual void** moveX(float x)

### Code Source

Permet de bouger l'objet sur l'axe des x.

## 2.12 moveY

**virtual void** moveY(float y)

### Code Source

Permet de bouger l'objet sur l'axe des y.

## 2.13 setPosition

**virtual void** setPosition(float x, float y)

### Code Source

Définie la position x et y.

## 2.14 setSpriteScale

**virtual void** setSpriteScale(float x, float y)

### Code Source

Définie l'échelle x et y du spirite de l'objet.

### 2.15      **setSpeed**

**virtual void** setSpeed(**float** val)

#### **Code Source**

Définie la vitesse de l'objet.

### 2.16      **setHsp**

**virtual void** setHsp(**float** val)

#### **Code Source**

Définie la vitesse horizontal.

### 2.17      **setVsp**

**virtual void** setVsp(**float** val)

#### **Code Source**

Définie la vitesse vertical.

### 2.18      **setAngularMove**

**virtual void** setAngularMove(**float const** &DELTA\_TIME, **float** speed, **float** angle)

#### **Code Source**

Permet de bouger l'objet en fonction d'un angle et d'une vitesse.

### 2.19      **setImageXscale**

**virtual void** setImageXscale(**float** val)

#### **Code Source**

Définie l'échelle x de l'objet.

### 2.20      **setImageYscale**

**virtual void** setImageYscale(**float** val)

#### **Code Source**

Définie l'échelle y de l'objet.

### 2.21      **setImageScale**

**virtual void** setImageScale(**float** val)

#### **Code Source**

Définie l'échelle x et y de l'objet avec une même valeur.

### 2.22      **setImageAngle**

**virtual void** setImageAngle(**float** val)

#### **Code Source**



Définie l'angle de l'objet.

#### 2.23      **setXOffset**

**virtual void** setXOffset(**float** val)

#### **Code Source**

Définie le décalage x de l'objet.

#### 2.24      **setYOffset**

**virtual void** setYOffset(**float** val)

#### **Code Source**

Définie le décalage y de l'objet.

#### 2.25      **setXYOffset**

**virtual void** setXYOffset()

#### **Code Source**

Définie le décalage x et y de l'objet avec une même valeur.

#### 2.26      **setTime**

**void** setTime(**float** x)

#### **Code Source**

Définie la valeur de la variable **m\_time** de l'objet.

#### 2.27      **setImageAlpha**

**virtual void** setImageAlpha(**int** val)

#### **Code Source**

Définie l'image alpha de l'objet.

#### 2.28      **setImageIndex**

**virtual void** setImageIndex(**int** val)

#### **Code Source**

Définie la sous image de l'objet.

#### 2.29      **setMaskW**

**virtual void** setMaskW(**int** val)

#### **Code Source**

Définie la largeur du masque de collision de l'objet.

#### 2.30      **setMaskH**

**virtual void** setMaskH(**int** val)

#### **Code Source**

Définie la hauteur du masque de collision de l'objet.

### 2.31 `setIsActive`

**virtual void** setIsActive(**bool** val)

#### Code Source

Définie l'état d'activité de l'objet.

### 2.32 `updateCollisionMask`

- Première forme :

**virtual void** updateCollisionMask()

#### Code Source

Met à jour les informations (taille, position, ...) du masque de collision.

- Deuxième forme:

**virtual void** updateCollisionMask(**int** x, **int** y)

#### Code Source

Met à jour la position du masque de collision en fonction d'un point x et y différents de celui de l'objet.

### 2.33 `centerCollisionMask`

**virtual void** centerCollisionMask(**int** x, **int** y)

#### Code Source

Centre la position du masque de collision en fonction d'un point x et y.

### 2.34 `updateSprite`

- Premier forme

**virtual void** updateSprite()

#### Code Source

Met à jour le sprite de l'objet avec les valeurs des variables (alpha, scale, ...) qui se trouvent dans l'objet.

- Deuxième forme

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

#### Code Source

Met à jour le sprite de l'objet avec des valeurs externes.

### 2.35 `draw`

**virtual void** draw(sf::RenderTexture &surface)

#### Code Source

Permet d'afficher l'objet.

### 2.36 `getMask`

**virtual** **is::Rectangle** getMask() **const**

#### Code Source

**Retourne** le masque de collision.

2.37      **getX**

**virtual float** getX() **const**

#### Code Source

**Retourne** la position x de l'objet.

2.38      **getY**

**virtual float** getY() **const**

#### Code Source

**Retourne** la position y de l'objet.

2.39      **getXStart**

**virtual float** getXStart() **const**

#### Code Source

**Retourne** la position x de début de l'objet.

2.40      **getYStart**

**virtual float** getYStart() **const**

#### Code Source

**Retourne** la position y de début de l'objet.

2.41      **getXPrevious**

**virtual float** getXPrevious() **const**

#### Code Source

**Retourne** la position x précédente de l'objet.

2.42      **getYPrevious**

**virtual float** getYPrevious() **const**

#### Code Source

**Retourne** la position y précédente de l'objet.

2.43      **distantToPoint**

**virtual float** distantToPoint(**float** x, **float** y) **const**

#### Code Source

**Retourne** la distance entre l'objet et un point x et y.

2.44      **distantToObject**

**virtual float** distantToObject(**std::shared\_ptr**<MainObject> **const** &other, **bool** useSpritePosition) **const**

#### Code Source

**Retourne** la distance entre l'objet et un autre.

**Paramètre** si **useSpritePosition** est **vrais** on utilise la position du sprite de l'objet pour faire le test **si non** on utilise la position **x, y** de l'objet.

#### 2.45 pointDirection

- Première forme

**virtual float** pointDirection(**float** x, **float** y) **const**

#### Code Source

**Retourne** la direction (angle) de l'objet par rapport à un point.

- Deuxième forme

**virtual float** pointDirection(**std::shared\_ptr**<MainObject> **const** &other) **const**

#### Code Source

**Retourne** la direction (angle) de l'objet par rapport à un autre. Ici l'autre objet est un pointeur.

#### 2.46 pointDirectionSprite

- Première forme

**virtual float** pointDirectionSprite(**float** x, **float** y) **const**

#### Code Source

**Retourne** la direction (angle) du sprite de l'objet par rapport à un point.

- Deuxième forme

**virtual float** pointDirectionSprite(**std::shared\_ptr**<MainObject> **const** &other) **const**

#### Code Source

**Retourne** la direction (angle) du sprite de l'objet par rapport à un autre.

#### 2.47 getSpeed

**virtual float** getSpeed() **const**

#### Code Source

**Retourne** la vitesse de l'objet.

#### 2.48 getHsp

**virtual float** getHsp() **const**

#### Code Source

**Retourne** la vitesse horizontale de l'objet.

#### 2.49 getVsp

**virtual float** getVsp() **const**

### Code Source

**Retourne** la vitesse verticale de l'objet

**2.50**      **getFrame**

**virtual float** getFrame() **const**

### Code Source

**Retourne** le numéro du sous image qui est en cours d'affichage.

**2.51**      **getFrameStart**

**virtual float** getFrameStart() **const**

### Code Source

**Retourne** le numéro du sous image de début.

**2.52**      **getFrameEnd**

**virtual float** getFrameEnd() **const**

### Code Source

**Retourne** le numéro du sous image de fin.

**2.53**      **getImageXscale**

**virtual float** getImageXscale() **const**

### Code Source

**Retourne** l'échelle x de l'objet.

**2.54**      **getImageYscale**

**virtual float** getImageYscale() **const**

### Code Source

**Retourne** l'échelle y de l'objet.

**2.55**      **getImageScale**

**virtual float** getImageScale() **const**

### Code Source

**Retourne** l'échelle de l'objet.

**2.56**      **getImageAngle**

**virtual float** getImageAngle() **const**

### Code Source

**Retourne** l'angle de l'image de l'objet.

**2.57**      **getXOffset**

**virtual float** getXOffset() **const**

### Code Source

**Retourne** le décallage en x de l'objet.

**2.58**      **getYOffset**

**virtual float** getYOffset() **const**

### Code Source

**Retourne** le décallage en y de l'objet.

**2.59**      **getTime**

**virtual float** getTime() **const**

### Code Source

**Retourne** la valeur de la variable **m\_time**.

**2.60**      **getInstanceId**

**virtual int** getInstanceId() **const**

### Code Source

**Retourne** le numéro de l'objet.

**2.61**      **getMaskWidth**

**virtual int** getMaskWidth() **const**

### Code Source

**Retourne** la largeur du masque de collision.

**2.62**      **getMaskHeight**

**virtual int** getSpriteHeight() **const**

### Code Source

**Retourne** la hauteur du masque de collision.

**2.63**      **getIsActive**

**virtual bool** getIsActive() **const**

### Code Source

**Retourne** l'état de l'objet.

**2.64**      **getImageAlpha**

**virtual int** getImageAlpha() **const**

### Code Source

**Retourne** l'image alpha de l'objet.

**2.65**      **getImageIndex**

**virtual int** getImageIndex() **const**

### Code Source

**Retourne** l'index de l'image.

2.66      **getSpriteWidth**

**virtual int** getSpriteWidth() **const**

### Code Source

**Retourne** la largeur du sprite.

2.67      **getSpriteHeight**

**virtual int** getSpriteHeight() **const**

### Code Source

**Retourne** la hauteur du sprite.

2.68      **getSpriteX**

**virtual float** getSpriteX() **const**

### Code Source

**Retourne** la position x du sprite.

2.69      **getSpriteY**

**virtual float** getSpriteY() **const**

### Code Source

**Retourne** la position y du sprite.

2.70      **getSpriteCenterX**

**virtual int** getSpriteCenterX() **const**

### Code Source

**Retourne** le centre x du sprite.

2.71      **getSpriteCenterY**

**virtual int** getSpriteCenterY() **const**

### Code Source

**Retourne** le centre y du sprite.

2.72      **placeMetting**

- **Première forme**

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

### Code Source

**Retourne** **vrai** s'il y a collision avec un autre objet, **faux** si non.

- **Deuxième forme**

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

#### Code Source

**Retourne vrai** s'il y a collision avec un autre objet, **faux** si non. Ici l'autre objet est un pointeur.

#### 2.73 getSprite

**virtual sf::Sprite&** getSprite()

#### Code Source

**Retourne** le sprite de l'objet.

#### 2.74 setFrame

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

#### Code Source

Définie l'image de début et de fin qui servira à faire l'animation du sprite de l'objet.

### 3. Autres fonctions de MainObject

#### 3.1 instanceExist

- Première forme

**template<class T>**

**bool** instanceExist(**std::shared\_ptr**<T> **const** &obj)

#### Code Source

**Retourne vrai** si l'instance existe, **faux** si non.

- Deuxième forme

**template<class T>**

**bool** instanceExist(T **const** \*obj)

#### Code Source

**Retourne vrai** si l'instance existe, **faux** si non.

#### 3.2 operator()

**class** CompareX;

#### Code Source

Foncteur qui sert à trier des objets.

**bool operator()**(**std::shared\_ptr**<MainObject> **const** &a, **std::shared\_ptr**<MainObject> **const** &b) **const**

#### Code Source

Sert à trier les objets en fonction de leurs positions x.

#### 3.3 sortObjArray

**template<class T>**



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

#### Code Source

Trie un tableau (`std::vector`) d'objets.

### 3.4 operator>

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

#### Code Source

**Retourne vrai** si la position de l'objet A est supérieur à celui de B, **faux** si non.

### 3.5 operator<

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

#### Code Source

**Retourne vrai** si la position de l'objet A est inférieur à celui de B, **faux** si non.

## *Les formes pour les masques collisions*

---

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

### 1. class Rectangle

```
class Rectangle;
```

#### Code Source

Représente le masque de collision rectangle. Ces membres **m\_left**, **m\_top**, **m\_right**, **m\_bottom** permettent de définir la taille du masque.

### 2. class Point

```
class Point;
```

#### Code Source

Représente le masque de collision point. Ces membres **m\_x**, **m\_y** permettent de définir la position du point.

- Première forme

```
Point()
```

#### Code Source

Constructeur par défaut.

- Deuxième forme

```
Point(float x, float y)
```

#### Code Source

Constructeur qui permet de définir la position du point.

### 3. class Line

**class** Line;

### Code Source

Représente le masque de collision ligne. Ces membres **m\_x1**, **m\_x2**, **m\_y1**, **m\_y2** permettent de définir la longueur de la ligne.

- Première forme

Line()

### Code Source

Constructeur par défaut.

- Deuxième forme

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

### Code Source

Constructeur qui permet de définir la longueur de la ligne.

## Les Classes Parentes de MainObject

---

### 1. class DepthObject

**class** DepthObject;

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

### Code Source

Classe qui offre des méthodes pour gérer la profondeur d’affichage des objets dans une scène.

#### 1.1 enum Depth

**enum** Depth;

Enumérateur	
VERY_BIG_DEPTH	Très Grande profondeur
BIG_DEPTH	Grande profondeur
NORMAL_DEPTH	Profondeur normal
SMALL_DEPTH	Petite profondeur
VERY_SMALL_DEPTH	Très petite profondeur

### Code Source

Représente le niveau de profondeur d’un objet.

#### 1.2 DepthObject

DepthObject(Depth Depth)

### Code Source

Constructeur pour définir une profondeur.

### 1.3 setDepth

**virtual void** setDepth(Depth val)

#### Code Source

Définie la profondeur de l'objet.

### 1.4 getDepth

**virtual** Depth getDepth() **const**

#### Code Source

**Retourne** la profondeur de l'objet.

## 2. class Destructible

**class** Destructible;

Entête : *isEngine/system/entity/parents/Destructible.h*

#### Code Source

Classe qui offre des méthodes pour gérer la destruction d'un objet.

### 2.1 Destructible

Destructible()

#### Code Source

Constructeur par défaut.

### 2.2 setDestroyed

**virtual void** setDestroyed()

#### Code Source

Enclenche la destruction d'un objet.

### 2.3 isDestroyed

**virtual bool** isDestroyed() **const**

#### Code Source

**Retourne** l'état de l'objet.

## 3. class Visibility

**class** Visibility;

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

#### Code Source

Class qui offre des méthodes pour gérer la visibilité d'un objet.

### 3.1 Visibility

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

## Code Source

Constructeur de la classe.

### 3.2 setVisible

```
void setVisible(bool value)
```

## Code Source

Définit la visibilité de l'objet.

### 3.3 isVisible

```
bool isVisible() const
```

## Code Source

Retourne l'état de l'objet.

## 4. class Health

```
class Health;
```

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

## Code Source

Class qui offre des méthodes pour gérer la santé d'un objet.

### 4.1 Health

- Première forme

```
Health(int health)
```

## Code Source

Constructeur de la classe, prend en paramètre la santé à attribuer à l'objet. Ici la valeur maximale de la santé est égale à la santé définie.

- Deuxième forme

```
Health(int health, int maxHealth)
```

## Code Source

Constructeur de la classe prend en paramètre la santé à attribuer à l'objet et la valeur maximale.

### 4.2 setHealth

```
virtual void setHealth(int val)
```

## Code Source

Définit la santé de l'objet.

### 4.3 setMaxHealth

```
virtual void setMaxHealth(int val)
```

## Code Source

Définie la santé maximale (la limite à ne pas dépasser) de l'objet.

#### 4.4 addHealth

```
virtual void addHealth(int val = 1)
```

##### Code Source

Ajoute la santé à l'objet. Peut servir aussi à le retancher si on met une valeur négative.

#### 4.5 getHealth

```
virtual int getHealth() const
```

##### Code Source

Retourne la santé de l'objet.

#### 4.6 getMaxHealth

```
virtual int getMaxHealth() const
```

##### Code Source

Retourne la santé maximale (la limite à ne pas dépasser) de l'objet.

### 5. class HurtEffect

```
class HurtEffect;
```

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

##### Code Source

Classe qui offre des méthodes pour faire un effet invulnérabilité sur un objet. C'est-à-dire faire clignoter l'objet pendant un certain temps (e.g. quand le joueur est attaqué par un ennemi il devient invulnérable en clignotant pendant une durée limitée).

#### 5.1 HurtEffect

HurtEffect(sf::Sprite &sprParent) :

##### Code Source

Constructeur de la classe prend en paramètre le sprite sur lequel l'effet invulnérabilité sera effectué.

#### 5.2 hurtStep

```
void hurtStep(float const &DELTA_TIME)
```

##### Code Source

Permet de faire l'animation d'invulnérabilité.

#### 5.3 setHurt

```
void setHurt(float durratation = 100.f)
```

##### Code Source

Définie la durée (en milliseconde) de l'invulnérabilité de l'objet.

#### 5.4 getHurt

```
bool getIsHurt() const
```

#### Code Source

**Retourne** vrai si l'objet est invulnérable, faux si non.

### 6. class ScorePoint

```
class ScorePoint;
```

Entête : *isEngine/system/entity/parents/ScorePoint.h*

#### Code Source

Classe qui offre des méthodes pour gérer le score à attribuer à un objet (e.g. chaque ennemi possède un point particulier lors de sa création qu'on ajoute au score global du joueur quand il est vaincu).

#### 6.1 ScorePoint

```
explicit ScorePoint(int point = 0)
```

#### Code Source

Constructeur de la classe, prend en paramètre le point à attribuer à l'objet.

#### 6.2 setScorePoint

```
virtual void setScorePoint(int point)
```

#### Code Source

Définit le point de l'objet.

#### 6.3 getScorePoint

```
virtual int getScorePoint() const
```

#### Code Source

**Retourne** le point attribué à l'objet.

### 7. class Step

```
class Step;
```

Entête : *isEngine/system/entity/parents/Step.h*

Classe qui offre des méthodes pour gérer les différentes étapes d'un objet (e.g. pour faire décoller une fusée il faut passer par plusieurs étapes).

#### 7.1 Step

```
explicit Step(int step = 0)
```

#### Code Source

Constructeur de la classe.

#### 7.2 setStep

```
virtual void setStep(int val)
```

#### Code Source

Définie l'étape de l'objet.

### 7.3 addStep

**virtual void** addStep()

#### Code Source

Avance l'étape de l'objet.

### 7.4 reduceStep

**virtual void** reduceStep()

#### Code Source

Réduit l'étape de l'objet.

### 7.5 getStep

**virtual int** getStep() **const**

#### Code Source

**Retourne** l'étape à laquelle l'objet est.

## Admob

---

### 1. class AdmobManager

**class** AdmobManager;

Entête: *isEngine/system/android/AdmobManager.h*

#### Code Source

Classe qui permet d'utiliser le SDK Admob dans le jeu. Elle offre des fonctions pour gérer les annonces de type bannière et vidéo récompense.

### 2. Les méthodes publiques

#### 2.1 AdmobManager

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

#### Code Source

Constructeur de la classe, elle prend en paramètre la fenêtre, l'activité Android et la machine virtuel.

#### 2.2 loadBannerAd

**void** loadBannerAd()

#### Code Source

Lance une requête pour l'annonce de type bannière.

#### 2.3 showBannerAd

**void** showBannerAd()

## Code Source

Affiche une bannière d'annonce à condition que la requête ait été bien exécutée.

### 2.4 hideBannerAd

```
void hideBannerAd()
```

## Code Source

Cache la bannière d'annonce.

### 2.5 loadRewardVideo

```
void loadRewardVideo()
```

## Code Source

Lance une requête pour l'annonce de type vidéo récompense.

### 2.6 updateSFMLApp

```
auto updateSFMLApp(bool whiteColor)
```

## Code Source

Met à jour l'application SFML en arrière-plan lorsqu'une annonce est affichée. Ceci permet d'éviter le plantage du programme principal.

### 2.7 checkAdObjInit

```
void checkAdObjInit()
```

## Code Source

Assure l'initialisation des composants d'Admob.

### 2.8 checkAdRewardObjReinitialize

```
void checkAdRewardObjReinitialize()
```

## Code Source

Réinitialise les composants d'Admob.

## 3. Autres Fonction d'AdmobManager

### 3.1 ProcessEvents & WaitForFutureCompletion

```
static bool ProcessEvents(int msec)
```

```
static void WaitForFutureCompletion(firebase::FutureBase future)
```

## Code Source

Assure le bon fonctionnement des tests sur les composants d'annonces.

### 3.2 checkAdState

```
static bool checkAdState(firebase::FutureBase future)
```

## Code Source

**Retourne** **vrais** si le test sur le composant d'annonce a été bien effectué, **faux** si non.



### 1. class gameTime

**class** gameTime;

Entête : *isEngine/system/function/gameTime.h*

#### Code Source

Cette Classe permet de manipuler le temps du jeu (le chronomètre). Très utilisée pour les jeux de plateforme style Super Mario Bros ou Sonic qui utilise un chronomètre dans un niveau.

### 2. Les méthodes publiques de gameTime

#### 2.1 gameTime

- Première forme

gameTime()

#### Code Source

Constructeur par défaut, initialise tous les compteurs (minute, seconde, milliseconde) à zéro (0).

- Deuxième forme

gameTime(unsigned int ms)

#### Code Source

Constructeur pour initialiser le temps avec les millisecondes qui seront distribuées plus tard en minutes et secondes.

- Troisième forme

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

#### Code Source

Constructeur pour initialiser le temps avec les minutes, secondes et millisecondes.

#### 2.2 step

void step(float const &DELTA\_TIME, float const &VALUE\_CONVERSION, float const &VALUE\_TIME)

#### Code Source

Lancer le compte à rebours du chronomètre pour qu'il s'arrête à zéro (0).

#### 2.3 addTimeValue

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

#### Code Source

Ajouter les minutes, les secondes et les millisecondes au temps actuel.

#### 2.4 setTimeValue

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

#### Code Source

Définie une nouvelle minute, seconde et milliseconde pour le temps actuelle.

## 2.5 setMSecond

```
void setMSecond(int ms)
```

### Code Source

Définie des millisecondes qui seront distribuées en minutes et secondes.

## 2.6 getTimeString

```
std::string getTimeString() const
```

### Code Source

**Retourne** le temps actuel sous la forme d'une chaîne de caractère (exemple **00: 00.00**).

## 2.7 getTimeValue

```
unsigned int getTimeValue() const
```

### Code Source

**Retourne** le temps en milliseconde.

## 2.8 getMinute

```
unsigned int getMinute() const
```

### Code Source

**Retourne** la minute.

## 2.9 getSecond

```
unsigned int getSecond() const
```

### Code Source

**Retourne** la seconde.

## 2.10 getMSecond

```
unsigned int getMSecond() const
```

### Code Source

**Retourne** la milliseconde.

## 2.11 operator=

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

### Code Source

Opérateur d'égalité pour comparer deux objets de type temps.

## 2.12 operator<<

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

### Code Source

Opérateur pour afficher le temps avec le flux **std::cout**.

### 3. Autres Fonctions de 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)
```

#### Code Source

Ces Opérateurs permettent de faire des comparaisons avec les objets de type temps.

## Commande du jeu

---

### 1. class GameKeyData

```
class GameKeyData
```

Entête : **isEngine/system/function/GameKeyData.h**

#### Code Source

Classe qui permet de gérer les commandes du jeu. Elle prend en compte le clavier et la souris sur PC et devient un Game Pad Virtuel sur Android.

### 2. Les éléments de GameKeyData

#### 2.1 enum VirtualKeyIndex

```
enum VirtualKeyIndex;
```

Enumérateur	
V_KEY_LEFT	Représente la touche directionnelle GAUCHE
V_KEY_RIGHT	Représente la touche directionnelle DROITE
V_KEY_UP	Représente la touche directionnelle HAUT
V_KEY_DOWN	Représente la touche directionnelle BAS
V_KEY_A	Représente la touche A
V_KEY_B	Représente la touche B
V_KEY_NONE	Aucune touche

#### Code Source

Représente les commandes du jeu.

#### 2.2 GameKeyData

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

#### Code Source

Constructeur qui prend en paramètre la scène.

#### 2.3 loadResources

```
void loadResources(sf::Texture &tex)
```

#### Code Source

Permet de charger la texture qui servira à créer les touches du Game Pad Virtuel.

#### 2.4 step

```
void step(float const &DELTA_TIME)
```

#### Code Source

Met à jour la position du Game Pad Virtuel sur l'écran et permet aussi de détecter l'utilisation des commandes.

#### 2.5 draw

```
void draw(sf::RenderTexture &surface)
```

#### Code Source

Affiche le Game Pad Virtuel.

#### 2.6 m\_keyPausePressed

```
bool m_keyPausePressed
```

#### Code Source

Détermine si la touche de pause est appuyée.

#### 2.7 m\_keyLeftPressed

```
bool m_keyLeftPressed
```

#### Code Source

Stock l'état de la touche GAUCHE.

#### 2.8 m\_keyRightPressed

```
bool m_keyRightPressed
```

#### Code Source

Stock l'état de la touche DROITE.

#### 2.9 m\_keyUpPressed

```
bool m_keyUpPressed
```

#### Code Source

Stock l'état de la touche HAUT.

#### 2.10 m\_keyDownPressed

```
bool m_keyDownPressed
```

#### Code Source

Stock l'état de la touche BAS.

#### 2.11 m\_keyAPressed

```
bool m_keyAPressed
```

#### Code Source

Stock l'état de la touche A.

### 2.12 `m_keyBPressed`

`bool m_keyBPressed`

#### Code Source

Stock l'état de la touche B.

### 2.13 `m_keyAUsed`

`bool m_keyAUsed`

#### Code Source

Stock l'état de la touche A quand elle est utilisée.

### 2.14 `m_keyBUsed`

`bool m_keyBUsed`

#### Code Source

Stock l'état de la touche B quand elle est utilisée.

### 2.15 `m_disableAllKey`

`bool m_disableAllKey`

#### Code Source

Permet de désactiver tous les commandes du jeu.

### 2.16 `m_hideGamePad`

`bool m_hideGamePad`

#### Code Source

Permet de cacher le Game Pad Virtuel sur Android.

### 2.17 `m_keyboardA`

`sf::Keyboard::Key m_keyboardA`

#### Code Source

Représente la touche du clavier qui sert de touche A.

### 2.18 `m_keyboardB`

`sf::Keyboard::Key m_keyboardB`

#### Code Source

Représente la touche du clavier qui sert de touche B.

### 2.19 `m_keyboardLeft`

`sf::Keyboard::Key m_keyboardLeft`

#### Code Source

Représente la touche du clavier qui sert de touche GAUCHE.

## 2.20 `m_keyboardRight`

`sf::Keyboard::Key m_keyboardRight`

### Code Source

Représente la touche du clavier qui sert de touche DROITE.

## 2.21 `m_keyboardUp`

`sf::Keyboard::Key m_keyboardUp`

### Code Source

Représente la touche du clavier qui sert de touche HAUT.

## 2.22 `m_keyboardDown`

`sf::Keyboard::Key m_keyboardDown`

### Code Source

Représente la touche du clavier qui sert de touche BAS.

## 2.23 `m_moveKeyPressed`

`VirtualKeyIndex m_moveKeyPressed`

### Code Source

Permet de savoir si les touches directionnelles virtuelles sont appuyées.

## 2.24 `m_actionKeyPressed`

`VirtualKeyIndex m_actionKeyPressed`

### Code Source

Permet de savoir si les touches virtuelles A, B sont appuyées.

## 2.25 `keyLeftPressed`

`bool keyLeftPressed()`

### Code Source

**Retourne** **vrai** si la touche directionnelle GAUCHE est appuyée, **faux** si non.

## 2.26 `keyRightPressed`

`bool keyRightPressed()`

### Code Source

**Retourne** **vrai** si la touche directionnelle DROITE est appuyée, **faux** si non.

## 2.27 `keyUpPressed`

`bool keyUpPressed()`

### Code Source

**Retourne** **vrai** si la touche directionnelle HAUT est appuyée, **faux** si non.

## 2.28      **keyDownPressed**

**bool** keyDownPressed()

### Code Source

**Retourne** **vrai** si la touche directionnelle BAS est appuyée, **faux** si non.

## 2.29      **keyAPressed**

**bool** keyAPressed()

### Code Source

**Retourne** **vrai** si la touche A est appuyée, **faux** si non.

## 2.30      **keyBPressed**

**bool** keyBPressed()

### Code Source

**Retourne** **vrai** si la touche B est appuyée, **faux** si non.

## 2.31      **virtualKeyPressed**

**bool** virtualKeyPressed(VirtualKeyIndex virtualKeyIndex)

### Code Source

**Retourne** **vrai** si la touche virtuelle correspondante est appuyée, **faux** si non.

## 3.    **Autres fonctions de GameKeyData**

Ces fonctions se trouvent dans **GameKeyName.h**.

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

- **Première forme**

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

### Code Source

**Retourne** le nom de la touche du clavier sous forme de chaîne de caractère.

- **Deuxième forme**

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

### Code Source

**Retourne** le nom de la touche du clavier sous forme de chaîne **std::wstring**.

## *Game Système*

---

### 1. **class GameSystem**

**class** GameSystem;

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

## Code Source

Classe de base qui permet d'assurer le partage des informations du jeu entre les différents composants du moteur de jeu. Elle contient les variables globales et fonctions qui assurent le bon fonctionnement du moteur.

## 2. Les éléments de GameSystem

### 2.1 enum ValidationButton

**enum** ValidationButton;

Enémerateur	
MOUSE	Représente le bouton de validation de la souris (s'il est utilisé, il devient une action tactile sur Android)
KEYBOARD	Représente le bouton de validation du clavier
ALL_BUTTONS	Représente le bouton de validation de la souris et du clavier (s'il est utilisé, il devient une action tactile sur Android)

## Code Source

Représenter la touche de validation sur PC, Il permet de connaitre le bouton qui sera utilisé lors d'un test de validation.

### 2.2 GameSystem

GameSystem()

## Code Source

Constructeur par défaut.

### 2.3 isPressed

```
bool isPressed(  
  
    #if defined(__ANDROID_)  
  
    int finger = 0  
  
    #else  
  
    ValidationButton validationButton = ALL_BUTTONS  
  
    #endif  
  
) const
```

## Code Source

- Windows, Linux:

Vérifie si la touche de validation est appuyée.

La touche de validation est définie dans **GameConfig.h** (*voir ici : 2.1*).

- Android:

Vérifie si l'écran est touché par l'utilisateur.

**Paramètre :**



**finger** index du doigt (sur Android).

**validationButton** Représente le bouton de validation à utiliser pour passer le test.

**Exemple :**

- Vérifier si la touche de validation du clavier est appuyée, Par défaut cette touche est **ENTER**.

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

- Vérifier si le bouton de validation de la souris est appuyé, Par défaut c'est le bouton **GAUCHE**.

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

## 2.4 keyIsPressed

- **Première forme**

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

### Code Source

Vérifiez si la touche du clavier est enfoncée.

**Retourne vrai** si la touche est enfoncée, **faux** sinon.

- **Deuxième forme**

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

### Code Source

Vérifiez si le bouton de la souris est enfoncé.

**Retourne vrai** si le bouton est enfoncé, **faux** sinon.

## 2.5 fileExist

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

### Code Source

**Retourne vrai** si le fichier existe, **faux** si non.

## 2.6 playSound

**void** playSound(sf::Sound &snd)

### Code Source

Permet de jouer un son si l'option est activée.

## 2.7 stopSound

**void** stopSound(sf::Sound &snd)

### Code Source

Permet de stopper un son.

## 2.8 useVibrate

**void** useVibrate(**short** ms)

### Code Source

Permet d'utiliser le vibreur si l'option est activée (uniquement pour Android).

**Paramètre** **ms** représente la durée du vibreur en milliseconde.

## 2.9 saveConfig

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

### Code Source

Sauvegarde les données de configurations du jeu.

## 2.10 loadConfig

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

### Code Source

Charge les données de configurations du jeu.

## 2.11 savePadConfig

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

### Code Source

Enregistre les données de configuration du Game Pad Virtuel.

## 2.12 loadPadConfig

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

### Code Source

Charge les données de configuration du Game Pad Virtuel.

## 2.13 m\_disableKey

**bool** m\_disableKey

### Code Source

S'il est **vrai** toutes les fonctions du moteur qui gèrent les entrées (clavier, souris, tactile) sont désactivées.

## 2.14 m\_enableSound

**bool** m\_enableSound

### Code Source

Permet de savoir si le son est activé.

## 2.15 m\_enableMusic

**bool** m\_enableMusic

### Code Source

Permet de savoir si la musique est activée.

#### 2.16 m\_enableVibrate

**bool** m\_enableVibrate

### Code Source

Permet de savoir si le vibreur est activé.

#### 2.17 m\_keyIsPressed

**bool** m\_keyIsPressed

### Code Source

Permet de savoir si une touche / un bouton a été enfoncée.

#### 2.18 m\_firstLaunch

**bool** m\_firstLaunch

### Code Source

Permet de vérifier si le jeu a été lancé au moins une fois.

#### 2.19 m\_validationMouseKey

**sf::Mouse::Button** m\_validationMouseKey

### Code Source

Représente la variable qui stocke le bouton de validation de la souris.

#### 2.20 m\_validationKeyboardKey

**sf::Keyboard::Key** m\_validationKeyboardKey

### Code Source

Représente la variable qui stocke la touche de validation du clavier.

#### 2.21 m\_gameLanguage

**int** m\_gameLanguage

### Code Source

Représente l'index de la langue choisie.

#### 2.22 m\_padAlpha

**int** m\_padAlpha

### Code Source

Permet de modifier la transparence du Game Pad Virtuel.

## 1. class GameSystemExtended

```
class GameSystemExtended;
```

Entête : [app\\_src/gamesystem\\_ext/GameSystemExtended.h](#)

### Code Source

Classe dérivée de **GameSystem** ([cliquer ici 1 pour plus d'info](#)), elle assure le même rôle que son parent. Sa particularité est qu'elle contient de nouveaux éléments qui serviront à gérer le game play et à manipuler les différentes scènes de jeu.

## 2. Les éléments de GameSystemExtended

### 2.1 GameSystemExtended

```
GameSystemExtended()
```

### Code Source

Constructeur par défaut.

### 2.2 enum DisplayOption

```
enum DisplayOption;
```

Enumérateur	
INTRO	Accède à la scène d'introduction
RESTART_LEVEL	Recommence la scène quand on perd
GAME_OPTION_RESTART	Recommence la scène avec l'option restart
NEXT_LEVEL	Aller au niveau suivant
MAIN_MENU	Accède à la scène Menu Principal
GAME_LEVEL	Accède à la scène Game Level
GAME_OVER	Accède à la scène Game Over
GAME_END_SCREEN	Accède à la scène Fin du jeu

### Code Source

Permet de manipuler les différentes scènes.

### 2.3 initSystemData

```
void initSystemData()
```

### Code Source

Initialise les données liées au moteur de jeu.

### 2.4 initProgress

```
void initProgress()
```

### Code Source

Initialise les données de progression du jeu.

### 2.5 initData

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

## Code Source

Initialise les données de game play (score, vie, ...).

### 2.6 saveData

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

## Code Source

Sauvagarde les données du jeu.

### 2.7 loadData

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

## Code Source

Charge les données du jeu.

### 2.8 m\_launchOption

DisplayOption m\_launchOption

## Code Source

Détermine l'action qui sera effectuée sur les différentes scènes du jeu.

### 2.9 game play variables

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

## Code Source

Les variables globales du jeu.

## *Game Function*

---

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

Ces fonctions du moteur permettent de faire des conversions sur les chaines de caractères, manipuler le temps, manipuler les objets SFML, afficher des texts spéciaux, utiliser certaines fonctionnalités d'Android, faire des calculs géométriques, faire des tests sur des variables, utiliser des fonctions pour manipuler des valeurs aléatoires, ...

### 1. Fonction Générale

#### 1.1 VALUE\_CONVERSION

```
static float const VALUE_CONVERSION(65.f);
```

## Code Source

Agit sur le timing des compteurs.

Exemple:

- Ceci créer un compteur en milliseconde quand on le met dans la boucle de mise à jour

```
// msCpt est une variable de type entier
```

```
msCpt += (is::VALUE_CONVERSION * 1.538f) * DELTA_TIME; // DELTA_TIME est le temps d'exécution renvoyé par la machine
```

## 1.2 WITH

```
#define WITH(_SIZE)
```

### Code Source

Permet de parcourir un tableau de vector. **\_I** est le compteur.

Exemple:

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

## 1.3 w\_chart\_tToStr

```
std::string w_chart_tToStr(wchar_t const *str)
```

### Code Source

Convertie un **w\_chart\_t** en **std::string**.

## 1.4 strToWStr

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

### Code Source

Convertie un **std::string** en **std::wstring**.

## 1.5 numToStr

```
template <class T>
```

```
std::string numToStr(T val)
```

### Code Source

Convertie un numérique en un **std::string**.

## 1.6 strToNum

```
template <typename T>
```

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

### Code Source

Convertie un **std::string** en un numérique.

## 1.7 numToWStr

```
template <class T>
```

**std::wstring** numToWStr(T val)

#### Code Source

Convertie un numérique en un **std::wstring**.

#### 1.8 writeZero

**template** <class T>

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

#### Code Source

Dessine des zeros devant un nombre.

**Paramètre zeroNumber** représente le nombre de zéro à afficher.

#### Exemple :

```
int var(7);  
std::cout << is::writeZero(var, 2) << std::endl; // sa affiche dans la console "007"
```

#### 1.9 getMSecond

**int** getMSecond(**float** **const** &DELTA\_TIME)

#### Code Source

**Retourne** le temps d'exécution en milliseconde.

#### 1.10 showLog

**void** showLog(**std::string** str)

#### Code Source

Affiche des messages dans la console.

#### 1.11 arraySize

**template** <**size\_t** SIZE, **class** T>

**inline size\_t** arraySize(T (&arr)[SIZE])

#### Code Source

**Retourne** la taille d'un tableau.

#### 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)

#### Code Source

Permet de faire le choix d'une valeur de façon aléatoire.

**Paramètre valNumber** le nombre valeur à tester.

### Exemple :

```
std::cout << is::choose(3, 7, 12, 4) << std::endl; // sa affiche dans la console de façon aléatoire 7 ou 12 ou 4
```

#### 1.13 setVarLimit

```
template <typename T>
```

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

##### Code Source

Permet d'encadrer une valeur.

#### 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)
```

##### Code Source

Vérifie si la valeur d'une variable se trouve dans un ensemble de valeur.

### Exemple :

```
int year(2020);
if (is::isIn(3, year, 2020, 2005, 2000)) // cette condition sera vraie car la valeur de year se trouve dans la fonction
{
    // do something
}
```

#### 1.15 isBetween

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

##### Code Source

Vérifies si une valeur se trouve dans un intervalle.

#### 1.16 isCrossing

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

##### Code Source

Vérifie si un point intersecte un autre.

#### 1.17 side

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

##### Code Source

Retourne -1 à gauche, 1 à droite, 0 si a b c sont alignés.

#### 1.18 sign

```
int sign(float x)
```

##### Code Source



**Retourne** le signe de la valeur.

### 1.19 pointDirection

**template** <typename T>

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

#### Code Source

Determine l'angle entre deux points.

### 1.20 pointDistance

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

#### Code Source

Determine la distance entre deux points.

### 1.21 radToDeg

float radToDeg(float x)

#### Code Source

Convertie le radient en degré.

### 1.22 degToRad

float degToRad(float x)

#### Code Source

Convertie le degré en radiant.

### 1.23 lengthDirX

float lengthDirX(float dir, float angle)

#### Code Source

**Retourne** la composante de x.

### 1.24 lengthDirY

float lengthDirY(float dir, float angle)

#### Code Source

**Retourne** la composante de y.

### 1.25 increaseVar

**template** <typename T>

void increaseVar(const float &DELTA\_TIME, T &var, T increaseValue, T varFinal, T varMax)

#### Code Source

Incrémente une variable avec le temps d'exécution.

**Exemple :**

```
int var(0);
is::increaseVar(DELTA_TIME, var, 1, 15, 10)); // la variable "var" va s'incrémenter avec la valeur 1. Si elle est
// supérieur à 10 elle devient 15 et l'incrémentatation s'arrête
```

## 1.26 decreaseVar

```
template <typename T>
```

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

### Code Source

Décrémente une variable avec le temps d'exécution.

#### Exemple :

```
int var(40);
is::decreaseVar(DELTA_TIME, var, 1, 20, 25)); // la variable "var" va se décrémenter avec la valeur 1. Si elle est
// inférieur à 25 elle devient 20 et la décrémentatation s'arrête
```

## 1.27 collisionTest

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

### Code Source

Test la collision entre deux (2) rectangles.

## 2. Fonction sur les objets SFML

### 2.1 getSFMLObjAngle

```
template <class T>
```

```
float getSFMLObjAngle(T obj)
```

### Code Source

**Retourne** l'angle de l'objet.

### 2.2 getSFMLObjXScale

```
template <class T>
```

```
float getSFMLObjXScale(T obj)
```

### Code Source

**Retourne** l'échelle x de l'objet.

### 2.3 getSFMLObjYScale

```
template <class T>
```

```
float getSFMLObjYScale(T obj)
```

### Code Source

**Retourne** l'échelle y de l'objet.

### 2.4 getSFMLObjWidth

**template** <class T>

**float** getSFMLObjWidth(T obj)

#### Code Source

**Retourne** la largeur de l'objet.

### 2.5 getSFMLObjHeight

**template** <class T>

**float** getSFMLObjHeight(T obj)

#### Code Source

**Retourne** la hauteur de l'objet.

### 2.6 getSFMLObjOriginX

**template** <class T>

**float** getSFMLObjOriginX(T obj)

#### Code Source

**Retourne** l'origine en x.

### 2.7 getSFMLObjOriginY

**template** <class T>

**float** getSFMLObjOriginY(T obj)

#### Code Source

**Retourne** l'origine en y.

### 2.8 getSFMLObjX

- Première forme

**template** <class T>

**float** getSFMLObjX(T obj)

- Deuxième forme

**template** <class T>

**float** getSFMLObjX(T \*obj)

#### Code Source

**Retourne** la position x.

### 2.9 getSFMLObjY

- Première forme

**template** <class T>

**float** getSFMLObjY(T obj)

- Deuxième forme

**template** <class T>

**float** getSFMLObjY(T \*obj)

#### Code Source

**Retourne** la position y.

#### 2.10 setSFMLObjAngle

**template** <class T>

**void** setSFMLObjAngle(T &obj, **float** angle)

#### Code Source

Définit l'angle.

#### 2.11 setSFMLObjRotate

**template** <class T>

**void** setSFMLObjRotate(T &obj, **float** rotationSpeed)

#### Code Source

Définit la rotation de l'objet.

#### 2.12 setSFMLObjScaleX\_Y

**template** <class T>

**void** setSFMLObjScaleX\_Y(T &obj, **float** x, **float** y)

#### Code Source

Définit l'échelle x et y.

#### 2.13 setSFMLObjScale

**template** <class T>

**void** setSFMLObjScale(T &obj, **float** scale)

#### Code Source

Définit l'échelle x et y avec la même valeur.

#### 2.14 setSFMLObjOrigin

**template** <class T>

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

#### Code Source

Définit l'origine x et y.

#### 2.15 setSFMLObjX

**template** <class T>

`void setSFMLObjX(T &obj, float x)`

#### Code Source

Définie la position x.

#### 2.16 setSFMLObjY

`template <class T>`

`void setSFMLObjY(T &obj, float y)`

#### Code Source

Définie la position y.

#### 2.17 centerSFMLObj

`template <class T>`

`void centerSFMLObj(T &obj)`

#### Code Source

Centrer l'objet en x et y.

#### 2.18 centerSFMLObjX

`template <class T>`

`void centerSFMLObjX(T &obj)`

#### Code Source

Centrer l'objet en x.

#### 2.19 centerSFMLObjY

`template <class T>`

`void centerSFMLObjY(T &obj)`

#### Code Source

Centrer l'objet en y.

#### 2.20 setSFMLObjX\_Y

- Première forme

`template <class T>`

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

- Deuxième forme

`template <class T>`

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

#### Code Source

Définie la position x et y.

## 2.21 moveSFMLObjX

**template** <class T>

**void** moveSFMLObjY(T &obj, **float** speed)

### Code Source

Déplace l'objet SFML sur l'axe des x.

## 2.22 moveSFMLObjY

**template** <class T>

**void** moveSFMLObjY(T &obj, **float** speed)

### Code Source

Déplace l'objet SFML sur l'axe des y.

## 2.23 setSFMLObjSize

- Première forme

**template** <class T>

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

- Deuxième forme

**template** <class T>

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

### Code Source

Définit la taille de l'objet.

## 2.24 setSFMLObjAlpha

- Première forme

**template** <class T>

**void** setSFMLObjAlpha(T &obj, **unsigned int** alpha)

- Deuxième forme

**template** <class T>

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

- Troisième forme

**template** <class T>

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

### Code Source

Définit la transparence. *Peut générer des WARNING si on l'utilise sur des textes et des formes géométriques !*

## 2.25 setSFMLObjAlpha2

**template** <class T>

**void** setSFMLObjAlpha2(T &obj, **unsigned int** alpha)

#### Code Source

Définit la transparence pour les objets de type texte, rectangles, ...***Ne marche pas sur les sprite !***

#### 2.26 setSFMLObjColor

**template** <class T>

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

#### Code Source

Définit la couleur de l'objet (Sprite).

#### 2.27 setSFMLObjFillColor

**template** <class T>

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

#### Code Source

Définit la couleur de l'objet (Texte, Rectangle, ..).

#### 2.28 scaleAnimation

**template** <class T>

**void** scaleAnimation(**float const** &DELTA\_TIME, **float** &var, T &obj, **short** varSign = 1, **float** scaleSize = 1.f)

#### Code Source

Permet de faire une animation d'étirement sur un objet SFML.

#### 2.29 setFrame

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

#### Code Source

Définit l'animation d'un sprite (***cliquer ici Figure 1 pour voir comment on l'utilise***).

#### 2.30 setSFMLObjOutlineColor

- Première forme

**template** <class T>

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

#### Code Source

Définit la couleur de contour.

- Deuxième forme

**template** <class T>

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

### Code Source

Définie la couleur de contour et sa taille.

#### 2.31 setSFMLObjTexRec

**template** <class T>

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

### Code Source

Définie le **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)

### Code Source

Définie les diverses propriétés d'un objet SFML.

#### 2.33 getSFMLSndState

**template** <class T>

**bool** getSFMLSndState(T &obj, **int** stateIndex)

### Code Source

**Retourne** l'état du son.

**Paramètre** valeur de test de **stateIndex** : 0 = Playing, 1 = Stopped, 2 = Pause

#### 2.34 collisionTestSFML

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

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

### Code Source

Test la collision entre deux (2) objets SFML.

#### 2.35 createRectangle

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

### Code Source

Créer un rectangle avec divers paramètres.

#### 2.36 textStyleConfig

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

### Code Source

Définie le style d'un texte.



## 2.37 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)
```

### Code Source

Créer un texte avec un chaîne `std::wstring`.

## 2.38 createText

- Première forme

```
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)
```

- Deuxième forme

```
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)
```

- Troisième forme

```
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)
```

- Quatrième forme

```
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)
```

- Cinquième forme

```
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)
```

### Code Source

Ces fonctions permettent de créer un texte avec divers paramètres.

## 2.39 createSprite

- Première forme

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

- Deuxième forme

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

- Troisième forme

```
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)
```

### Code Source

Ces fonctions permettent de créer un sprite avec divers paramètres.

## 2.40 mouseCollision

- Première forme

**template** <class T>

```
bool mouseCollision(sf::RenderWindow &window, T const &obj  
  
    #if defined(__ANDROID_)  
  
    , unsigned int finger = 0  
  
    #endif  
  
)
```

#### Code Source

**Windows, Linux :** Détecte si le curseur de la souris est en collision avec un objet de la fenêtre.

**Android :** Détecte si l'utilisateur touche un objet de la fenêtre.

#### Paramètre :

**obj** l'objet avec lequel on veut faire le test.

**finger** représente l'index du doigt.

**Retourne** **vrais** si il y a collision et **faux** si non.

#### Exemple :

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

- **Deuxième forme**

**template** <class T>

```
bool mouseCollision(sf::RenderWindow &window, T const &obj, sf::RectangleShape &cursor  
  
    #if defined(__ANDROID_)  
  
    , unsigned int finger = 0  
  
    #endif  
  
)
```

#### Code Source

**Windows, Linux :** Détecte si le curseur de la souris est en collision avec un objet de la fenêtre.

**Android :** Détecte si l'utilisateur touche un objet de la fenêtre.

#### Paramètre :

**obj** l'objet avec lequel on veut faire le test.

**cursor** permet de récupérer la position du point de collision.

**finger** représente l'index du doigt.

**Retourne** **vrais** si il y a collision et **faux** si non.

### **Exemple :**

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

## **3. Autres fonctions**

### **3.1 vibrate**

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

#### **Code Source**

Lance le vibreur Android.

### **3.2 openURL**

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

#### **Code Source**

Ouvre un url dans le navigateur (e.g www.website.com).

### **3.3 setScreenLock**

**void** setScreenLock(**bool** disableLock)

#### **Code Source**

Définie le verrouillage de l'écran Android.

### **3.4 jstring2string**

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

#### **Code Source**

Convertie **jstring** en **std::string**.

### **3.5 getDeviceId**

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

#### **Code Source**

**Retourne** l'id du périphérique Android.

## ***Bibliothèque externe***

---

### **1. Swoosh**

Elle est intégrée par défaut au moteur. C'est grâce à elle que le moteur arrive à faire des effets transisitions.

Pour plus d'information cliquer [ici](#).

## 2. Tiny File Dialogs (uniquement pour Windows et Linux)

### 2.1 class TinyDialogBox

```
class TinyDialogBox;
```

Entête : *isEngine/ext\_lib/TinyFileDialogs/TinyDialogsBox.h*

#### Code Source

Classe qui permet d'utiliser la bibliothèque TinyFileDialogs de la façon la plus simple. Elle vous permet d'utiliser les boîtes de dialogue du système d'exploitation (Windows et Linux).

### 2.2 tinyString

```
#if !defined(SFML_SYSTEM_LINUX)
```

```
typedef wchar_t const* tinyString;
```

```
#else
```

```
typedef char const* tinyString;
```

```
#endif
```

#### Code Source

Type personnalisé qui permet de manipuler les données de tinyFileDialogs. Lorsqu'on utilise tinyFileDialogs les données diffèrent en fonction du système d'exploitation. Sur windows les chaînes de caractères deviennent des `wchar_t const*` et sur Linux des `char const*`, ce qui implique l'utilisation de deux (2) types différents ayant le même but pour un même programme. Le type `tinyString` permet de palier à ce problème en déterminant automatiquement le type qui correspond au système d'exploitation cible.

### 2.3 TINY\_FILE\_DIALOGBOX\_PATH

```
static tinyString TINY_FILE_DIALOGBOX_PATH;
```

#### Code Source

Stocke le chemin du fichier de la boîte de dialogue.

### 2.4 enum FileDialogType

```
enum FileDialogType;
```

Enumérateur	
SAVE_FILE	Sauvegarde fichier
LOAD_FILE	Charger fichier

#### Code Source

Représente le type de boîte de dialogue à afficher.

### 2.5 enum DialogType

```
enum DialogType;
```

Enumérateur	
OK	Message avec bouton OK
OKCANCEL	Message avec bouton OK et CANCEL
YESNO	Message avec bouton YES et NO

## Code Source

Représente les boutons qui seront affichés sur la boite de dialogue.

### 2.6 enum IconType

**enum** IconType;

Enumérateur	
INFO	Boite de dialogue avec une icône INFO
WARNING	Boite de dialogue avec une icône WARNING
ERROR_ICO	Boite de dialogue avec une icône ERROR
QUESTION	Boite de dialogue avec une icône QUESTION

## Code Source

Représente l'icône qui sera affichée sur la boite de dialogue.

### 2.7 enumDialogTypeToStr / enumIconTypeToStr

**static tinyString const** enumDialogTypeToStr(DialogType val)

**static tinyString const** enumIconTypeToStr(IconType val)

## Code Source

Ces fonctions convertissent les **enum** en chaîne de caractère pour les passer plus tard dans les fonctions de la bibliothèque.

### 2.8 showDialogBox

```
static int showDialogBox(tinyString title,  
                        tinyString msgError,  
                        DialogType dialogType,  
                        IconType iconType  
                        )
```

## Code Source

Affiche une boite de dialogue de type message.

**Retourne 1** lorsque l'utilisateur clique sur le bouton OK et **0** lorsqu'il clique sur CANCEL ou 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

)

```

### Code Source

Affiche une boîte de dialogue de type fichier.

**Retourne** le chemin du fichier si la fonction a réussi et "" (**chaîne vide**) si non.

#### 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

)

```

### Code Source

Affiche une boîte de dialogue de type sélection dossier.

**Retourne** le chemin du dossier si la fonction a réussi et "" (**chaîne vide**) si non.

### 3. Box 2D

Box 2D est un moteur physique intégré au moteur de jeu. Pour l'utiliser dans une scène vous devez l'inclure de cette façon : **#include "../..../isEngine/ext\_lib/Box2D/Box2D.h"**

## Game Engine

---

### 1. class GameEngine

```
class GameEngine;
```

Entête : *isEngine/core/GameEngine.h*

## Code Source

Cette Classe assure l'interconnexion des différents composants du moteur et lance la boucle de rendu dans laquelle le jeu va se dérouler.

### 2. Les méthodes de GameEngine

#### 2.1 GameEngine

GameEngine()

## Code Source

Constructeur par défaut.

#### 2.2 initEngine

bool initEngine()

## Code Source

Initialise le moteur de jeu.

#### 2.3 play

bool play()

## Code Source

Boucle de rendu principal du moteur.

#### 2.4 basicSFMLmain

bool basicSFMLmain()

## Code Source

Boucle de rendu d'une fenêtre SFML classique.

#### 2.5 getRenderWindow

sf::RenderWindow& getRenderWindow()

## Code Source

**Retourne** la fenêtre SFML.

## *Configuration du jeu*

---

namespace GameConfig;

Entête : *app\_src/config/GameConfig.h*

Permet de définir des paramètres pour préconfigurer ces parties du jeu : La taille de la fenêtre et de la vue, Les touches du clavier et de la souris à utiliser pour commander le jeu, les informations du jeu (nom, auteur, version), chemin des fichiers ressources (son, image, sauvegarde, ...) et les informations d'Admob.

### 1. Paramètre de la fenêtre

## 1.1 WINDOW\_WIDTH

```
static const unsigned int WINDOW_WIDTH
```

### Code Source

Définit la largeur de la fenêtre.

## 1.2 WINDOW\_HEIGHT

```
static const unsigned int WINDOW_HEIGHT
```

### Code Source

Définit la hauteur de la fenêtre.

## 1.3 VIEW\_WIDTH

```
static const float VIEW_WIDTH
```

### Code Source

Définit la largeur de la vue.

## 1.4 VIEW\_HEIGHT

```
static const float VIEW_HEIGHT
```

### Code Source

Définit la hauteur de la vue.

## 2. Paramètre des boutons de validation

### 2.1 KEY\_VALIDATION\_MOUSE

```
static const sf::Mouse::Button KEY_VALIDATION_MOUSE
```

### Code Source

Représente le bouton qui valide les options avec la souris.

### 2.2 KEY\_VALIDATION\_KEYBOARD

```
static const sf::Keyboard::Key KEY_VALIDATION_KEYBOARD
```

### Code Source

Représente la touche qui valide les options avec le clavier.

### 2.3 KEY\_CANCEL

```
static const sf::Keyboard::Key KEY_CANCEL
```

### Code Source

Représente la touche qui annule les options avec le clavier.

## 3. Paramètre des touches du clavier

### 3.1 KEY\_A

```
static const sf::Keyboard::Key KEY_A
```

### Code Source



Représente la touche du bouton A.

### 3.2 KEY\_B

```
static const sf::Keyboard::Key KEY_B
```

#### Code Source

Représente la touche du bouton B.

### 3.3 KEY\_LEFT

```
static const sf::Keyboard::Key KEY_LEFT
```

#### Code Source

Représente la touche directionnelle GAUCHE.

### 3.4 KEY\_RIGHT

```
static const sf::Keyboard::Key KEY_RIGHT
```

#### Code Source

Représente la touche directionnelle DROITE.

### 3.5 KEY\_UP

```
static const sf::Keyboard::Key KEY_UP
```

#### Code Source

Représente la touche directionnelle HAUT.

### 3.6 KEY\_DOWN

```
static const sf::Keyboard::Key KEY_DOWN
```

#### Code Source

Représente la touche directionnelle BAS.

## 4. Les informations du jeu

### 4.1 MAJOR

```
static const std::wstring MAJOR
```

#### Code Source

Définit la version majeur.

### 4.2 MINOR

```
static const std::wstring MINOR
```

#### Code Source

Définit la version mineur.

### 4.3 getGameVersion

```
inline std::wstring getGameVersion()
```

## Code Source

**Retourne** la version du jeu.

### 4.4 GAME\_NAME

```
static std::wstring const GAME_NAME
```

## Code Source

Définie le nom du jeu.

### 4.5 GAME\_AUTHOR

```
static std::wstring const GAME_AUTHOR
```

## Code Source

Définie le nom de l'auteur.

## 5. Paramètre Admob

```
namespace AdmobConfig;
```

Permet de définir les informations d'Admob pour pouvoir afficher les annonces dans le jeu. ***Ces informations sont fournies sur la plateforme de Google Admob !***

### 5.1 Id de la PUB

#### 5.1.1 kAdMobAppID

```
static const char* kAdMobAppID
```

## Code Source

Code de l'application Admob.

#### 5.1.2 kBannerAdUnit

```
static const char* kBannerAdUnit
```

## Code Source

Code de la bannière.

#### 5.1.3 kRewardedVideoAdUnit

```
static const char* kRewardedVideoAdUnit
```

## Code Source

Code de la vidéo récompense.

### 5.2 Taille de la bannière de PUB

#### 5.2.1 kBannerWidth

```
static const int kBannerWidth
```

## Code Source

Définie la largeur de la bannière d'annonce.

#### 5.2.2 kBannerHeight

**static const int** kBannerHeight

#### Code Source

Définie la hauteur de la bannière d'annonce.

### 5.3 Publique cible de la PUB

#### 5.3.1 kBirthdayDay

**static const int** kBirthdayDay

#### Code Source

Définie le jour de naissance des utilisateurs.

#### 5.3.2 kBirthdayMonth

**static const int** kBirthdayMonth

#### Code Source

Définie le mois de naissance des utilisateurs.

#### 5.3.3 kBirthdayYear

**static const int** kBirthdayYear

#### Code Source

Définie l'année de naissance des utilisateurs.

#### 5.3.4 kKeywords

**static const char\*** kKeywords[]

#### Code Source

Mots-clés à utiliser pour faire la demande d'une annonce.

## 6. Chemin des fichiers ressources du jeu

### 6.1 GUI\_DIR

**static std::string const** GUI\_DIR

#### Code Source

Chemin des fichiers ressources qui servent d'interface graphique.

### 6.2 FONT\_DIR

**static std::string const** FONT\_DIR

#### Code Source

Chemin des fichiers ressources qui servent de police.

### 6.3 SPRITES\_DIR

**static std::string const** SPRITES\_DIR

#### Code Source

Chemin des fichiers ressources qui servent de Sprite.

## 6.4 TILES\_DIR

```
static std::string const TILES_DIR
```

### Code Source

Chemin des fichiers ressources qui servent de tuiles et d'arrière-plan.

## 6.5 SFX\_DIR

```
static std::string const SFX_DIR
```

### Code Source

Chemin des fichiers ressources qui servent de SFX.

## 6.6 MUSIC\_DIR

```
static std::string const MUSIC_DIR
```

### Code Source

Chemin des fichiers ressources qui servent de musique.

## 7. Nom du package du jeu (Android)

```
static std::string const PACKAGE_NAME
```

### Code Source

Nom du package du jeu. Représente l'endroit où vos données seront enregistrées sur Android.

Vous devez appliquer ce nom pour l'**applicationId** dans le fichier **build.gradle**

## 8. Chemin des fichiers de sauvegarde

### 8.1 GAME\_DATA\_FILE

```
static std::string const GAME_DATA_FILE
```

### Code Source

Chemin pour enregistrer le fichier de sauvegarde du jeu.

### 8.2 CONFIG\_FILE

```
static std::string const CONFIG_FILE
```

### Code Source

Chemin pour enregistrer le fichier de configuration du jeu.

### 8.3 GAME\_PAD\_FILE

```
static std::string const GAME_PAD_FILE
```

### Code Source

Chemin pour enregistrer le fichier de configuration du Game Pad Virtuel sur Android.

## 1. class GameActivity

**class** GameActivity;

Entête: *app\_src/activity/GameActivity.h*

### Code Source

Permet de lancer les différentes scènes de jeu. Une autre particularité de cette classe est qu'elle relie les scènes du moteur et la bibliothèque **SWOOSH** afin de pouvoir utiliser les effets transitions de cette dernière.

## 2. Les éléments de GameActivity;

### 2.1 GameActivity

GameActivity(ActivityController& controller, GameSystemExtended &gameSysExt)

### Code Source

Constructeur de la classe, elle prend en paramètre le contrôleur d'activité (de la bibliothèque SWOOSH) et gestionnaire du système de jeu (*[cliquer ici 1 pour plus d'info](#)*).

### 2.2 m\_gameScene

**std::shared\_ptr**<**is::GameDisplay**> m\_gameScene;

### Code Source

Instance de la scène qui sera utilisée.

### 2.3 onStart

**virtual void** onStart()

### Code Source

Lorsque la scène est lancée.

### 2.4 onUpdate

**virtual void** onUpdate(**double** elapsed)

### Code Source

Sert à mettre à jour les informations de la scène.

### 2.5 onLeave

**virtual void** onLeave()

### Code Source

Lorsque la scène n'est plus utilisée (interruption).

### 2.6 onExit

**virtual void** onExit()

### Code Source

Lorsqu'on quitte la scène pour une autre.

### 2.7 onEnter

**virtual void** onEnter()

#### Code Source

Lorsque la séquence de la scène commence.

### 2.8 onResume

**virtual void** onResume()

#### Code Source

Lorsqu'on reprend la scène après une interruption.

### 2.9 onDraw

**virtual void** onDraw(sf::RenderTexture& surface)

#### Code Source

Permet d'afficher la scène.

### 2.10 onEnd

**virtual void** onEnd()

#### Code Source

Lorsqu'on quitte la scène (destruction).

## Niveau

---

### 1. Les niveaux

Dans is::Engine les niveaux du jeu sont des tableaux d'entier contenus dans des fichiers entête (fichier.h). Ces niveaux sont créés grâce à l'éditeur de niveau **is::Level Editor** ([lien](#) du projet) qui est livré avec le moteur.

Entête : *app\_src/levels/Level.h*

### 2. Intégration d'un niveau

Pour intégrer un niveau on inclut son entête dans le fichier **Level.h** de cette façon :

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

### 3. Les éléments pour gérer les niveaux

#### 3.1 namespace level

```
namespace level;
```

#### Code Source

Permet d'utiliser les contenus qui permettent de gérer les niveaux.

#### 3.2 enum LevelId

```
enum LevelId
```

```
{
```

```

LEVEL_1,
LEVEL_2,
/* ... */
, LEVEL_MAX // Permet de connaitre le nombre total de niveau intégré
}

```

### Code Source

Représente l'index de chaque niveau. A chaque fois qu'un nouveau niveau est intégré au moteur on doit déclarer son index.

### 3.3 getLevelMap

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

### Code Source

**Retourne** le tableau du niveau renseigné dans le paramètre.

A chaque fois qu'un nouveau niveau est intégré, on doit renseigner l'instruction qui renverra ce niveau dans la fonction.

#### Exemple :

- **Intégration dans la fonction :**

```

// Renvoie le tableau du niveau qui se trouve dans level_1.h
inline short const* getLevelMap(int CURRENT_LEVEL)
{
    switch (CURRENT_LEVEL)
    {
        case LEVEL_1 : return LEVEL_1_MAP; break; // LEVEL_1_MAP est le nom du tableau qui se trouve dans level_1.h
        // ...
    }
}

```

- **Utilisation dans un fichier source externe :** (Ceci est un exemple simple juste pour vous expliquer le principe. Pour aller plus loin, veuillez-vous référer à la *Démo du moteur*)

```
short*currentLevelArray = getLevelMap(LEVEL_1); // Retourne le tableau qui se trouve dans level_1.h
```

## Langage du jeu

---

### 1. Les langues

Les langues sont représentées dans is::Engine par des tableaux de chaîne.

Entête : **app\_src/language/GameLanguage.h**

### 2. Les éléments pour gérer les langues

#### 2.1 namespace Lang

```
namespace lang;
```

## Code Source

Sert à gérer les langues du jeu.

### 2.2 enum GameLanguage

```
enum GameLanguage
{
    ENGLISH, ///< Représente la langue Anglaise
    FRANCAIS, ///< Représente la langue le française

    /* ... */
}
```

## Code Source

Cette énumération permet d'implémenter l'index de chaque langue afin de pouvoir les utiliser plus facilement lors du développement.

### Exemple:

- Créer une phrase:

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

- Utilisation : (Ceci est un exemple simple juste pour vous expliquer le principe. Pour aller plus loin, veuillez-vous référer à la Démo du moteur)

```
gameSystemExt.m_gameLanguage = is::lang::GameLanguage::ENGLISH; // Choix de la langue Anglaise
is::showLog(is::lang::hello_world[gameSystemExt.m_gameLanguage]); // on aura dans la console : Hello World !
gameSystemExt.m_gameLanguage = is::lang::GameLanguage::FRANCAIS; // Choix de la langue Française
is::showLog(is::lang::hello_world[gameSystemExt.m_gameLanguage]); // on aura dans la console : Salut le monde !"
```

## Boite de Dialogue du jeu

---

### 1. class GameDialog

```
class GameDialog;
```

Entête : `app_src/objects/widgets/GameDialog.h`

## Code Source

Classe qui permet d'afficher des boîtes de dialogue comme dans les jeux RPG. Elle est étroitement liée à la partie langue du jeu ([cliquer ici 1 pour plus d'info](#)). Pour pouvoir afficher une dialogue vous devez définir un tableau de chaîne représentant ce dialogue dans `GameLanguage.h`.

### 2. Les éléments de GameDialog

#### 2.1 GameDialog

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

## Code Source



Constructeur de la classe, elle prend en paramètre la scène dans laquelle elle est utilisée.

## 2.2 enum DialogIndex

```
enum DialogIndex
{
    DIALOG_NONE = -1,
    DIALOG_PLAYER_MOVE, // Représente le dialogue qui parle de comment déplacer le joueur
    /* ... */
};
```

### Code Source

Représente les différents dialogues qui seront affichés dans le jeu. Les informations qui sont définies à l'intérieur sont liées à la partie langue du jeu.

A chaque fois qu'un index est ajouté on doit déclarer son tableau de chaîne dans **GameLanguage.h**.

#### Exemple :

- **Déclaration du dialogue DIALOG\_PLAYER\_MOVE dans 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()
```

### Code Source

Relie le tableau de chaîne qui se trouve dans **GameLanguage.h** et l'index du dialogue.

#### Exemple :

- **Lier un Index et son tableau de chaîne :** (Ceci est un exemple simple juste pour vous expliquer le principe. Pour aller plus loin, veuillez-vous référer à la *Démo du moteur*)

```
void linkArrayToEnum()
{
    // ...
    switch (m_dialogIndex)
    {
        case DIALOG_PLAYER_MOVE: // l'index du dialogue
            m_msgIndexMax = is::arraySize(is::lang::dialog_player_move); // Détermine le nombre de phrase
            checkMsg(is::lang::dialog_player_move); // Définir le dialogue grâce à son tableau de chaîne
    }
```

```
break;
```

```
// ...
```

## 2.4 loadResources

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

### Code Source

Charge les fichiers ressources de la boite de dialogue.

## 2.5 step

```
void step(const float &DELTA_TIME)
```

### Code Source

Met à jour les informations de la boite de dialogue.

## 2.6 setDialog

```
void setDialog(DialogIndex dialogIndex)
```

### Code Source

Définit le dialogue qui sera lancé.

## 2.7 setMouseInCollison

```
void setMouseInCollison(bool val)
```

### Code Source

Force la collision du curseur de la souris ou le doigt de l'utilisateur (sur Android) avec la boite de dialogue.

## 2.8 draw

```
void draw(sf::RenderTarget &surface)
```

### Code Source

Affiche la boite de dialogue.

## 2.9 getDialogIndex

```
DialogIndex getDialogIndex() const
```

### Code Source

**Retourne** l'énumérateur du dialogue qui est affiché.

## 2.10 getMouseInCollison

```
bool getMouseInCollison() const
```

### Code Source

**Retourne** **vrai** quand le curseur de la souris ou le doigt de l'utilisateur (sur Android) touche la boite de dialogue, **faux** si non.

## 2.11 showDialog

`bool showDialog() const`

## Code Source

**Retourne** **vrai** quand la boite de dialogue est ouverte et **faux** si non.

### Exemple de jeu

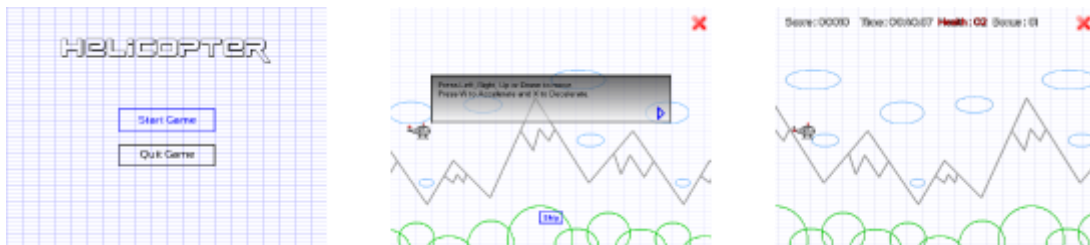
---

#### 1. Introduction

Dans cette partie du document nous allons découvrir comment utiliser les fonctions de `is::Engine` pour créer un mini jeu. Notez que ceci est juste un petit tutoriel pour vous permettre de bien démarrer avec le moteur.

Nous allons créer un jeu d'arcade dans lequel on contrôle un Hélicoptère qui doit éviter des obstacles et collecter des objets bonus qui augmentent le temps du chronomètre et son score. Si le chronomètre du niveau atteint zéro (0), il perd la partie.

Le jeu sera jouable sur Android et PC.



**Vous pouvez accéder au projet [ici](#).**

#### 2. Comment le jeu sera créé ?

##### 2.1 Voici les éléments du moteur que le jeu utilisera

- Classe `GameDisplay` pour créer les scènes
- Classe `MainObject` et ces parents pour créer les objets de game play (Joueur, HUD, Bonus, ...)
- Classe `GameKeyData` pour commander le joueur
- Classe `GameDialog` pour afficher le tutoriel
- `GameLangague.h` pour ajouter des phrases à traduire
- Certaines fonctions du moteur qui se trouvent dans `GameFunction.h`
- Classe `Activity` pour lancer les différents scènes et les faire interagir entre eux

##### 2.2 Les objets qui seront utilisés dans le jeu

- Un Menu principal qui contiendra ces objets :
  - Un Texte pour le titre du jeu
  - Deux (2) sprites qui serviront de Boutons : Un pour lancer le jeu et un autre pour quitter
  - Deux (2) textes qui serviront de titre pour les Boutons
- Une Scène appelée `GameLevel` où se déroule le jeu et aura pour contenu :
  - Un objet joueur qui servira d'Hélicoptère
  - Un objet HUD
  - Un sprite en forme de Croix pour quitter le Niveau
  - Un sprite pour le background
  - Des sons
  - Un texte pour afficher le message de game over
  - Un conteneur d'objet (`std::vector`) pour les Bonus
  - Un conteneur d'objet (`std::vector`) pour les Obstacles

##### 2.3 Les rôles des objets

- Classe Activity
  - Lancer les différents scènes
  - Transition entre Menu Principal et Game Level et vis versa.
- Classe MainMenu
  - Naviguer dans le menu avec la souris (toucher sur Android) et clavier
  - Utiliser les touches de validation pour choisir une option
  - Quitter le menu grâce une boîte de dialogue
- Classe GameLevel
  - Lancer la partie
  - Recommencer le niveau quand le joueur perd
  - Quitter le niveau quand l'utilisateur clic sur la croix (sprite)
- Classe Player sera un Hélicoptère :
  - Les touches HAUT, BAS, GAUCHE, DROITE serviront à déplacer l'objet
  - La touche A pour accélérer
  - La touche B pour la vitesse normale
  - Animé le sprite
- Classe Bonus
  - Disparaît quand le joueur le touche
  - Augmenter le Score et le temps du joueur et joue un son quand il est détruit
- Classe Obstacle
  - Collision avec le joueur (retirer la santé)
- Classe HUD
  - Affiche le chronomètre du niveau
  - Affiche le nombre de Bonus
  - Affiche le score du joueur
  - Affiche la santé du joueur

### 3. Intégration des phrases dans le jeu

#### 3.1 Création des phrases dans 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"};

#ifdef __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 Accélérer 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 Accélérer 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"};
}
}

```

### ➤ Explication

Ce fichier permet de définir les phrases à traduire qui vont être utilisées dans le jeu. Une phrase à traduire est représentée par un tableau de chaîne (**std::string** ou **std::wstring**). La première case du tableau représente la première langue, la case suivante la deuxième et ainsi de suite.

- **static std::wstring** dialog\_player\_move

### Code Source

Phrase qui sera utilisée plus tard dans la Boîte de dialogue pour montrer à l'utilisateur comment commander l'Hélico.

Sur Android on affiche comment bouger le joueur par rapport aux touches du Game Pad Virtuel et sur PC par rapport aux touches du clavier (qui peuvent changer en fonction des paramètres définis dans **GameConfig.h**).

- **is::getKeyWName(is::GameConfig::KEY\_LEFT)**

### Code Source

Permet d'obtenir le nom de la touche du clavier (sous forme de chaîne **std::wstring**) grâce son code qui lui est associé.

Ceci permet de connaître le nom de la touche du clavier associé à chaque action.

## 3.2 Association de la boîte de dialogue avec la phrase du jeu

Le code ci-dessous est une partie de la déclaration de la classe **GameDialog**.

```
// ...
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;
    }
}
// ...
```

#### ➤ Explication

L'**enum** **DialogIndex** et la fonction **void linkArrayToEnum()** sont les deux éléments de la classe **GameDialog** qui nous permet d'afficher des phrases de **GameLanguage.h** avec la boîte de dialogue.

- DIALOG\_PLAYER\_MOVE

#### Code Source

Représente la phrase **dialog\_player\_move** de **GameLanguage.h**. Les éléments de l'**enum** **DialogIndex** permettent de relier les phrases de **GameLanguage.h** et la classe **GameDialog**.

- **switch** (m\_dialogIndex)  
{  
    **case** DIALOG\_PLAYER\_MOVE:  
        m\_msgIndexMax = is::arraySize(is::lang::dialog\_player\_move);  
        checkMsg(is::lang::dialog\_player\_move);  
        **break**;

#### Code Source

Ces instructions permettent d'associer une phrase de **GameLanguage.h** avec la classe **GameDialog**. La procédure est la même pour tout autre type de phrase mais n'oubliez pas que pour chaque phrase (tableau de chaîne) vous devez définir son élément dans **enum DialogIndex**.

#### 4. Création des classes du jeu

##### 4.1 Classe Obstacle

###### 4.1.1 Entête

```
#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);
};
```

###### ➤ Explication

Classe Obstacle est une classe qui hérite de **MainObject** (offre des fonctions pour gérer le déplacement et l'affichage de l'objet) et **ScorePoint** une classe qui permet d'assigner des points bonus aux objets.

**void step(float const& DELTA\_TIME)** permet de mettre à jour les instances de la classe Obstacle.

##### 4.1.2 Implémentation

###### 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();
}
```

###### ➤ Explication

Constructeur de la classe qui prend en paramètre la texture et position de l'objet dans la scène.

**ScorePoint(20)** représente le score qui est attribué à l'objet. A l'intérieur du bloc il y a la définition de la taille du masque de collision, la vitesse de déplacement de l'objet et la fonction qui permet de créer le sprite de l'objet.

###### 4.1.2.2 step

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

###### ➤ Explication

Cette méthode permet de déplacer l'objet vers la gauche en fonctions de sa vitesse, mettre à jour la position du masque de collision et du sprite.

## 4.2 Classe Bonus

### 4.2.1 Entête

```
#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);
};
```

#### ➤ Explication

Classe fille de **MainObject**, elle hérite aussi de **Destructible** qui offre des fonctions pour gérer la destruction de ces instances de façon explicite. **ScorePoint** pour attribuer un point à l'objet qui sera comptabilisé plus tard. **Step** permet de gérer les différentes étapes de l'objet : collision avec le joueur et la destruction.

### 4.2.2 Implémentation

#### 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));
}
```

#### ➤ Explication

Constructeur qui prend la texture du sprite et la position de l'objet dans la scène.

A l'intérieur, la taille du masque de collision a été définie avec la vitesse de déplacement de l'objet, suivis de la fonction qui permet de créer le sprite de l'objet.

#### 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();
}
```

#### ➤ Explication

Cette méthode permet de déplacer l'objet et d'enclencher la destruction de l'objet quand son étape passe à 1. Elle met à jour aussi les propriétés du sprite et celui du masque de collision.



## 4.3 Classe Player

### 4.3.1 Entête

```
#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;
};
```

#### ➤ Explication

Classe fille de **MainObjet**, **Health** offre des méthodes qui permet de gérer la santé du joueur, **HurtEffect** permet de faire un effet invincibilité (faire clignoter l'objet quand il est touché).

- `void loadResources(sf::Texture &tex)`

#### Code Source

Permet d'attribuer des ressources extérieures (utilisé dans la scène) à l'objet.

- `GameKeyData &m_gameKey;`

#### Code Source

Sert à utiliser l'objet qui permet de gérer les commandes du jeu.

## 4.3.2 Implémentation

### 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();
}
```

#### ➤ Explication

Constructeur prend en paramètre l'instance de l'objet qui gère les commandes du jeu. Il permet aussi de définir le nombre de santé du joueur et de choisir le sprite qui sera utilisé pour faire l'effet invincibilité quand le joueur est touché par un obstacle.

A l'intérieur il y a la définition de la taille du masque de collision. La variable **m\_isActive** = **true** permet à l'utilisateur de contrôler l'objet quand il n'a pas perdu.

#### 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));
}
```

##### ➤ Explication

Permet d'utiliser la texture chargée dans la scène pour créer le sprite du joueur.

#### 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);
}
```

##### ➤ Explication

Méthode dans laquelle on gère le comportement de l'objet. Ici quand la variable **m\_isActive** est **vrai** alors l'utilisateur peut faire accélérer l'hélico quand il appuie sur **la touche A** et le faire ralentir quand il appuie sur **la touche B**. Il peut aussi déplacer l'objet avec **les quatres (4) touches directionnelles**. L'animation de l'Hélico (*qui sera détaillé en dessous*) s'effectue aussi dans ce bloc.

**Note :** quand l'utilisateur accélère ou ralentit ça affecte aussi les autres objets de la scène (Obstacles, Bonus, Arrière plan).

- hurtStep(DELTA\_TIME)

### Code Source

Permet de faire l'animation invincibilité (faire clignoter le sprite).

## Voici comment le sprite est animé :

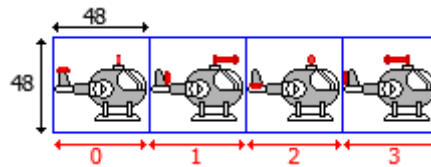


Figure 1

Pour animer le sprite on utilise une texture (**Figure 1**) composée de plusieurs sous image ayant les mêmes tailles. Chaque sous image représente une valeur (**en rouge**) que peut prendre la variable **m\_frame**. Ci-dessous les éléments qui permettent de faire une animation :

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

### Code Source

La fonction qui permet d'animer le sprite. Il prend en paramètre le sprite qui sera utilisé, la sous image qui sera affichée, le nombre de sous image **sur une ligne (ici qui est 4)** et les 3 autres paramètres qui sont la taille des sous images (ils se ressemblent mais ont des buts différents).

**Note :** La fonction découpe automatiquement l'image.

- m\_frame

### Code Source

Permet définir la sous image du sprite qui sera affichée.

- `setFrame(0.f, 3.6f);`

### Code Source

Permet définir l'intervalle de **m\_frame** c'est-à-dire les sous images à choisir.

## 4.4 Classe HUD

### 4.4.1 Entête

```
#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;
};
```

#### ➤ Explication

Classe qui permet d'afficher les informations de game play à l'écran.

- is::GameDisplay &m\_scene

#### Code Source

Permet d'avoir accès à la scène dans laquelle l'objet est utilisé pour pouvoir le positionner et utiliser les variables de game play (score, nombre de bonus collectés).

- is::GameTime &m\_gameTime

#### Code Source

A pour but d'afficher le chronomètre du jeu.

- Player &m\_player

#### Code Source

Permet d'afficher la santé du joueur.

### 4.4.2 Implémentation

#### 4.4.2.1 HUD

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

```

#### ➤ Explication

Constructeur qui prend en paramètre la scène, l'objet qui gère le chronomètre du jeu et l'instance du joueur.

#### 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);
}

```

#### ➤ Explication

Permet d'utiliser la police de caractère chargée dans la scène pour créer les textes.

#### 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);
}

```

#### ➤ Explication

Cette méthode sert à positionner les textes sur l'écran et à mettre à jour leurs informations.

#### 4.4.2.4 draw

```

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

```

#### ➤ Explication

Affiche les différents textes à l'écran. *Cette méthode est une surcharge !*

### 4.5 Classe MainMenu

#### 4.5.1 Entête

```

#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::RenderTarget &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;
};

```

#### ➤ Explication

Déclaration de la classe qui permet de créer la scène du Menu Principal.

```
void componentsController()
```

#### Code Source

Méthode où seront gérés les boutons du menu principal.

## 4.5.2 Implémentation

### 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)
{}
```

#### ➤ Explication

Constructeur de la classe, prend en paramètre la fenêtre, la vue, la surface et le gestionnaire du système de jeu. Il permet aussi de définir la couleur de fond de la scène (ici qui est Blanc).

### 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;
}
```

#### ➤ Explication

Cette méthode charge les ressources qui seront utilisés dans le menu et positionne les objets.

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

### Code Source

Permet de définir la langue par défaut du jeu. Si vous changez la valeur en `is::lang::GameLanguage::FRENCH` la langue française sera choisie.

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

### Code Source

- `is::lang::pad_new_game` : permet d'utiliser le tableau qui se trouve dans `GameLanguage.h`.
- `[m_gameSysExt.m_gameLanguage]` : permet de choisir la phrase qui correspond à la langue.

#### 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));
}
}

```

#### ➤ Explication

Cette méthode est une sous fonction de **step()**. Elle permet d'utiliser les touches du jeu et la souris (devient la fonction touchée sur Android) pour naviguer dans le menu et choisir une option. Elle permet aussi d'animer les objets du menu principal quand on effectue une action (survole souris, clic, appuie d'une touche).

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

#### Code Source

Anime le texte et le sprite et joue un son quand on change une option.

- m\_gameSysExt.m\_launchOption = is::DisplayOption::GAME\_LEVEL

#### Code Source

Informe le moteur que la prochaine scène à lancer sera celui du Niveau.

#### 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;
}
}

```

### ➤ Explication

Cette méthode gère la partie événement associée à la scène et de la boîte de dialogue du moteur de jeu (*non pas celui du tutoriel mais celle qui affiche un bouton OUI - NON*), ainsi que la fermeture de l'application.

- m\_isRunning = false;

### Code Source

Permet de stopper l'exécution de la scène afin de pouvoir la quitter.

#### 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();
}
```

#### ➤ Explication

Affiche les composants du Menu Principal.

### 4.6 Classe GameLevel

#### 4.6.1 Entête

```
#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;
};

```

### ➤ Explication

Déclaration de la classe qui représente le niveau.

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

### Code Source

Conteneur des objets Bonus et Obstacles.

- `GameKeyData m_gameKey`

### Code Source

Objet qui permet de gérer les commandes du jeu afin de contrôler le joueur : touche du clavier, souris et Game Pad Virtuel.

- `is::GameTime m_gameTime;`

### Code Source

Chronomètre du niveau.

- `sf::Text m_txtGameInfo`

### Code Source

Affiche un message et le score du joueur quand il perd la partie.

- `int m_timeCreateOstacle, m_timeCreateBonus`

### Code Source

Variable de type compteur (en milliseconde) pour générer des objets aléatoire dans la scène.

## 4.6.2 Implémentation

### 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))
}
```

### ➤ Explication

On définit un temps par défaut pour les compteurs qui permettent de générer des objets de façons aléatoire dans le niveau.

#### 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;
}
```

#### ➤ Explication

Méthode pour charger les ressources du jeu (musique, sons, sprite, ...), définir les paramètres de création de certains objets et de positionner les objets dans la scène.

- `m_gameTime.setTimeValue(0, 29, 59)`

#### Code Source

Définie le temps du chronomètre.

- `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)`

#### Code Source

Permet de créer l'arrière plan du niveau en répétant sa taille sur la longueur 2.5 fois. Ceci permet de faire défiler le l'arriere plan de façon infinie sur l'axe des x.

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

#### Code Source

Permet d'afficher la boite de dialogue avec le message qui montre comment commander le joueur.

#### 4.6.2.3 updateObjPlayer

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

#### ➤ Explication

Méthode qui met à jour le joueur.

#### 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);
}
```

#### ➤ Explication

Cette méthode permet de stopper la partie quand le joueur n'a plus de santé. Elle stoppe la musique du jeu, définit le texte de game over avec le score du joueur qui sera affiché et désactive le joueur (ce qui veut dire qu'il a perdu).

#### 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();
            }
        }
    }
}
```

#### ➤ Explication

Méthode qui met à jour les Obstacles. A l'intérieur de la boucle **WITH** on vérifie si le joueur est en collision avec l'objet, si oui on détruit l'obstacle et on retire une santé, mais s'il n'a plus de santé alors la partie est terminée.

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

#### Code Source

Permet de savoir si l'objet est sorti du côté gauche de la fenêtre. Si oui on le détruit pour libérer l'espace mémoire et on ajoute des points au joueur.

#### 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.addValue(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();
}
}
}

```

#### ➤ Explication

Méthode qui met à jour les Bonus. A l'intérieur de la boucle **WITH** on vérifie si le joueur est en collision avec l'objet si oui on ajoute un point et on augmente le temps du niveau.

Après on vérifie si le Bonus est sorti du coté gauche de la fenêtre, si oui on le détruit pour libérer l'espace mémoire.

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

#### Code Source

Permet d'exécuter des actions lors de la collision une seule fois et de pouvoir supprimer le Bonus plus tard.

##### 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);
}

```

#### ➤ Explication

Cette méthode met à jour le background en simulant une animation de défilement infinie.

##### 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();
}

```

#### ➤ Explication

Sous fonction de **step()**, elle gère le chronomètre du niveau, les commandes du jeu, les compteurs qui génèrent les objets Bonus et Obstacle et d'appliquer les fonctions qui mettent à jour les objets du game play.

#### 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_window.isActive())
    {
        // 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);
}
}

```

### ➤ Explication

Cette méthode gère la partie événement associé à la scène, la boîte de dialogue pour le tutoriel et les options qui permettent de recommencer un niveau ou de le quitter pour une autre.

- m\_gameSysExt.m\_launchOption = **is::DisplayOption::MAIN\_MENU**
- m\_gameSysExt.m\_launchOption = **is::DisplayOption::RESTART\_LEVEL**;

### Code Source

L'action qui sera effectué sur une scène.

#### 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);
}

```

➤ **Explication**

Affiche les objets de la scène.

## 5. Intégration et utilisation des scènes dans 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::RenderTarget& surface)
{
    m_gameScene->drawScreen();
}

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

```

#### ➤ Explication

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

#### Code Source

Permet d'inclure les deux scènes afin de les utilisées dans la classe **Activity**.

- std::shared\_ptr<is::GameDisplay> m\_gameScene;

#### Code Source

Représente l'instance qui stockera la scène à exécuter. **Attention c'est une variable qui s'adapte à la scène !**

- case is::DisplayOption::MAIN\_MENU:  
m\_gameScene = std::shared\_ptr<is::GameDisplay>(new GameMenu(controller.getWindow(),  
getView(),  
\*(this->controller->getSurface()),

```
gameSysExt));
```

### Code Source

Permet de lancer la scène du Menu Principal. Si **switch (m\_gameScene->getGameSystem().m\_launchOption)** est équivalent à **case is::DisplayOption::MAIN\_MENU**.

- **if** (m\_gameScene->isRunning()) m\_gameScene->step();

### Code Source

Lance la partie **step()** (mise à jour des contenus) d'une scène.

- **using** transition = segue<VerticalSlice, sec<2>>;  
**using** action = transition::to<GameActivity>;  
getController().**replace**<action>(m\_gameScene->getGameSystem());

### Code Source

Ces instructions permettent de passer d'une scène à une autre en faisant un effet transition (Swoosh).

N'oublier pas qu'on arrive à déterminer la scène qui sera changée par une autre grâce à : **switch (m\_gameScene->getGameSystem().m\_launchOption)** et l'instruction **case is::DisplayOption::nom\_de\_la\_scene:**

Cliquer [ici](#) pour avoir plus d'information sur l'utilisation des fonctions de la bibliothèque SWOOSH.

- m\_gameScene->drawScreen();

### Code Source

Lance la partie **draw()** (affichage des contenus) d'une scène.

## 6. Amélioration

Il y a encore plein de fonctionnalités qu'on peut apporter à ce mini jeu, en voici quelques un :

- Eviter que le joueur sorte de l'écran quand on le déplace
- Une interface dans le Menu Princiapal qui permet de changer la langue du jeu
- Une interface dans le Menu Princiapal qui permet d'activer / désactiver le son du jeu
- Augmenter la vitesse des Obstacles et Bonus au fur et à mesure que le score augmente
- Ajouter un bouton pour mettre le jeu en pause le jeu
- Etc...

***Maintenant c'est à vous de jouer !***