{B4 - CPP 2020}

[Lucas Troncy]|[Edouard Puillandre]|[Robin Ustarroz]

Sommaire

Présentation du programme

L'arcade est la réalisation d'une plateforme de jeux modulaire permettant de jouer à deux grands classiques du jeux vidéo que sont le Snake et le Centipede. Le programme supporte aussi le changement de librairie graphique de façon dynamique. (Il est possible en pleine partie de changer la librairie ou le jeu en appuyant sur la touche approprié).

Lexique

Echap : Permet de quitter le programme / partie en cours ;

Espace : Tirer (Centipede) ;

Entrée : Mettre le jeu en pause ;

2 : Charge la librairie graphique précédente ;

3 : Charge la librairie graphique suivante ;

4 : Charge le jeu précédent ;

5 : Charge le jeu suivant ;

8 : Redémarre le jeu

9 : Revenir au menu principal ;

Fleche Haut : Se diriger vers le haut (Jeux)

Fleche Bas : Se diriger vers le Bas (Jeux)

Fleche Gauche : Se diriger vers le Gauche (Jeu/Menu)

Fleche Droite : Se Diriger vers la Droite (Jeu/Menu)

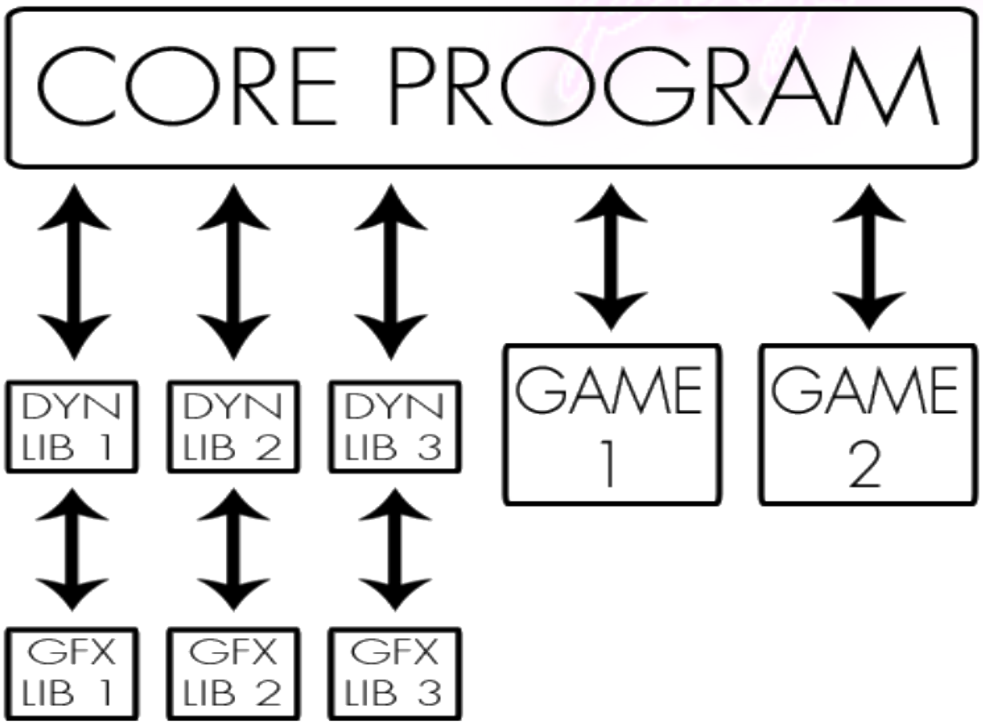
Fonctionnement du programme

Le programme respecte une architecture propre afin de pouvoir fonctionner de façon dynamique.

Le programme est divisé en trois parties bien distinctes :

- Le Core, « Cerveau » du programme fait le lien entre le jeu et la partie graphique.
- Les Jeux, présenté sous forme de librairie ils communiquent uniquement avec le Core.
- Les librairies graphiques, elles permettent l'affichage du jeu dans une fenêtre propre, elles ne communiquent jamais avec le jeu directement.

On peut représenter graphiquement l'architecture de cette façon :



La communication peut être résumé ainsi :

Graphique <-> ***Core*** <-> ***Jeux***

Cette configuration ne peut pas être modifié et doit ABSOLUMENT être respecté afin que les modules puissent être indépendant (cf : *Comment créer un jeu ou une librairie compatible avec l'arcade ?)*.

Le programme s'exécute de cette façon :

$> ./arcade [library_path]

Le programme se lancera donc avec la librairie donnée en argument.

Architecture du programme

Le programme est entièrement développé en C++, il intègre sa propre architecture(un diagramme des classes est présent à la fin de ce manuel). Le programme utilise différentes interfaces, dont deux principales. IGame permet de gérer le jeu de façon façon générique et IGfxLib qui permet de gérer lui les librairies graphiques.

Il existe un programme Core dont l'implémentation est relativement libre, puisque les interfaces suffisent à gérer la totalité des actions graphiques et du joueur.

Le Core DOIT être une exécutable et les librairies doivent être partagés (*.so sur unix *.dll sur Windows).

Comment Créer un jeu ou une librairie compatible avec l'arcade ?

Que la librairie soit un jeu ou une librairie graphique elle doit ne posséder qu'un seul entrypoint.

Chaque jeu doit posséder une fonction qui alloue, construit et retourne le jeu en tant que pointer IGame.

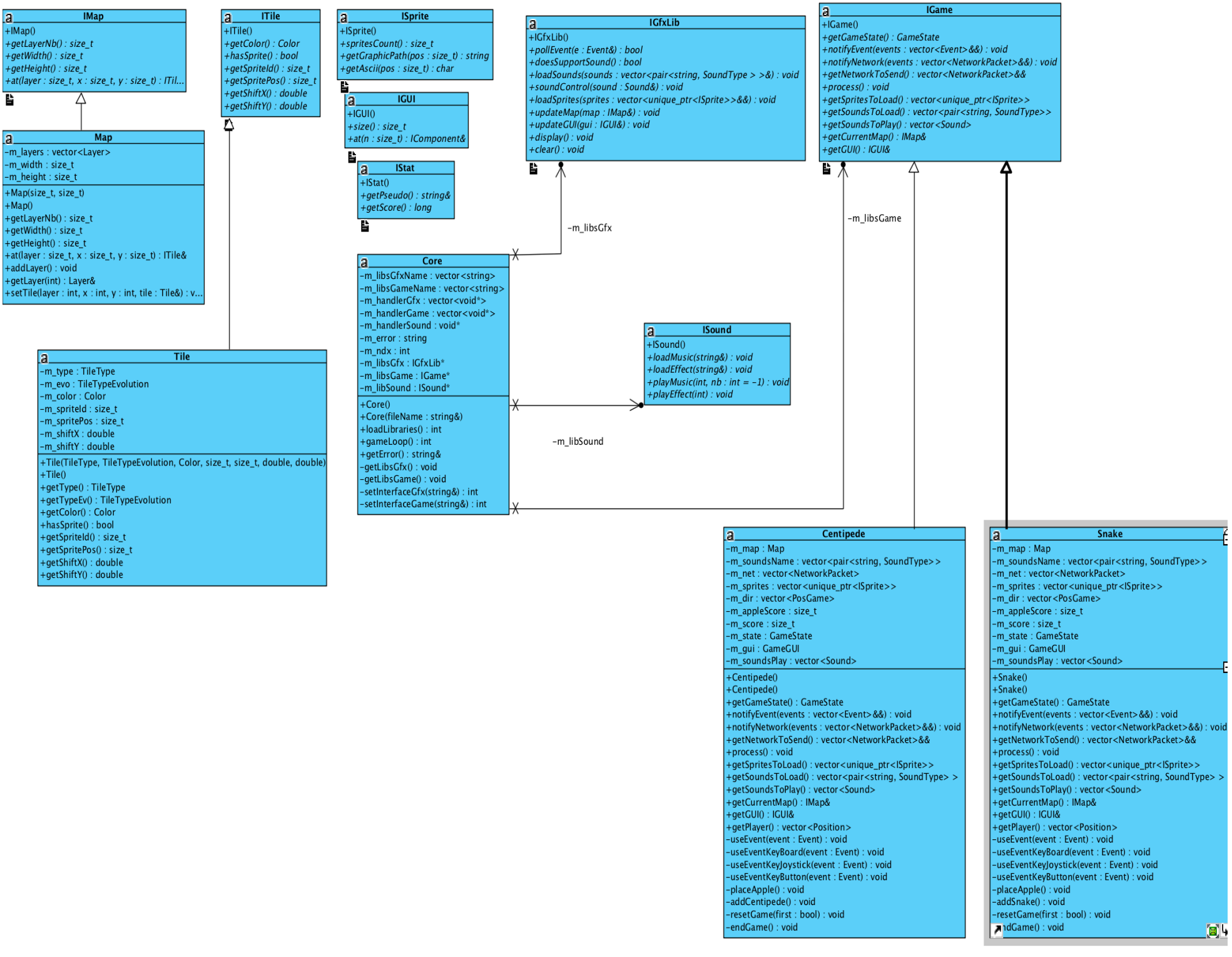
Il est nécessaire d'utiliser le mot clé Extern en C pour éviter les confusions de nom par le compilateur.

*Extern «C » arcade ::Igame *getGame() ;*

Chaque librairie graphique doit posséder une fonction qui alloue, construit et retourne le jeu en tant que pointer IGfxLib.

*Extern «C » arcade ::IGfxLib *getLib() ;*

Le mot clé extern doit être utilisé pour les mêmes raisons.


IMap
+IMap()
+getLayerNb() : size_t
+getWidth() : size_t
+getHeight() : size_t
+at(layer : size_t, x : size_t, y : size_t) : ITil...
Map
-m_layers : vector<Layer>
-m_width : size_t
-m_height : size_t
+Map(size_t, size_t)
+Map()
+getLayerNb() : size_t
+getWidth() : size_t
+getHeight() : size_t
+at(layer : size_t, x : size_t, y : size_t) : ITile&
+addLayer() : void
+getLayer(int) : Layer&
+setTile(layer : int, x : int, y : int, tile : Tile&) : v...
ITile
+ITile()
+getColor() : Color
+hasSprite() : bool
+getSpriteId() : size_t
+getSpritePos() : size_t
+getShiftX() : double
+getShiftY() : double
Tile
-m_type : TileType
-m_evo : TileTypeEvolution
-m_color : Color
-m_spriteId : size_t
-m_spritePos : size_t
-m_shiftX : double
-m_shiftY : double
+Tile(TileType, TileTypeEvolution, Color, size_t, size_t, double, double)
+Tile()
+getType() : TileType
+getTypeEv() : TileTypeEvolution
+getColor() : Color
+hasSprite() : bool
+getSpriteId() : size_t
+getSpritePos() : size_t
+getShiftX() : double
+getShiftY() : double
ISprite
+ISprite()
+spritesCount() : size_t
+getGraphicPath(pos : size_t) : string
+getAscii(pos : size_t) : char
IGUI
+IGUI()
+size() : size_t
+at(n : size_t) : IComponent&
IStat
+IStat()
+getPseudo() : string&
+getScore() : long
IGfxLib
+IGfxLib()
+pollEvent(e : Event&) : bool
+doesSupportSound() : bool
+loadSounds(sounds : vector<pair<string, SoundType > >&) : void
+soundControl(sound : Sound&) : void
+loadSprites(sprites : vector<unique_ptr<ISprite>>&&) : void
+updateMap(map : IMap&) : void
+updateGUI(gui : IGUI&) : void
+display() : void
+clear() : void
IGame
+IGame()
+getGameState() : GameState
+notifyEvent(events : vector<Event>&&) : void
+notifyNetwork(events : vector<NetworkPacket>&&) : void
+getNetworkToSend() : vector<NetworkPacket>&&
+process() : void
+getSpritesToLoad() : vector<unique_ptr<ISprite>>
+getSoundsToLoad() : vector<pair<string, SoundType>>
+getSoundsToPlay() : vector<Sound>
+getCurrentMap() : IMap&
+getGUI() : IGUI&
-m_libsGfx
-m_libsGame
Core
-m_libsGfxName : vector<string>
-m_libsGameName : vector<string>
-m_handlerGfx : vector<void*>
-m_handlerGame : vector<void*>
-m_handlerSound : void*
-m_error : string
-m_ndx : int
-m_libsGfx : IGfxLib*
-m_libsGame : IGame*
-m_libSound : ISound*
+Core()
+Core(fileName : string&)
+loadLibraries() : int
+gameLoop() : int
+getError() : string&
-getLibsGfx() : void
-getLibsGame() : void
-setInterfaceGfx(string&) : int
-setInterfaceGame(string&) : int
ISound
+ISound()
+loadMusic(string&) : void
+loadEffect(string&) : void
+playMusic(int, nb : int = -1) : void
+playEffect(int) : void
-m_libSound
Centipede
-m_map : Map
-m_soundsName : vector<pair<string, SoundType>>
-m_net : vector<NetworkPacket>
-m_sprites : vector<unique_ptr<ISprite>>
-m_dir : vector<PosGame>
-m_appleScore : size_t
-m_score : size_t
-m_state : GameState
-m_gui : GameGUI
-m_soundsPlay : vector<Sound>
+Centipede()
+Centipede()
+getGameState() : GameState
+notifyEvent(events : vector<Event>&&) : void
+notifyNetwork(events : vector<NetworkPacket>&&) : void
+getNetworkToSend() : vector<NetworkPacket>&&
+process() : void
+getSpritesToLoad() : vector<unique_ptr<ISprite>>
+getSoundsToLoad() : vector<pair<string, SoundType> >
+getSoundsToPlay() : vector<Sound>
+getCurrentMap() : IMap&
+getGUI() : IGUI&
+getPlayer() : vector<Position>
-useEvent(event : Event) : void
-useEventKeyBoard(event : Event) : void
-useEventKeyJoystick(event : Event) : void
-useEventKeyButton(event : Event) : void
-placeApple() : void
-addCentipede() : void
-resetGame(first : bool) : void
-endGame() : void
Snake
-m_map : Map
-m_soundsName : vector<pair<string, SoundType>>
-m_net : vector<NetworkPacket>
-m_sprites : vector<unique_ptr<ISprite>>
-m_dir : vector<PosGame>
-m_appleScore : size_t
-m_score : size_t
-m_state : GameState
-m_gui : GameGUI
-m_soundsPlay : vector<Sound>
+Snake()
+Snake()
+getGameState() : GameState
+notifyEvent(events : vector<Event>&&) : void
+notifyNetwork(events : vector<NetworkPacket>&&) : void
+getNetworkToSend() : vector<NetworkPacket>&&
+process() : void
+getSpritesToLoad() : vector<unique_ptr<ISprite>>
+getSoundsToLoad() : vector<pair<string, SoundType> >
+getSoundsToPlay() : vector<Sound>
+getCurrentMap() : IMap&
+getGUI() : IGUI&
+getPlayer() : vector<Position>
-useEvent(event : Event) : void
-useEventKeyBoard(event : Event) : void
-useEventKeyJoystick(event : Event) : void
-useEventKeyButton(event : Event) : void
-placeApple() : void
-addSnake() : void
-resetGame(first : bool) : void
ndGame() : void