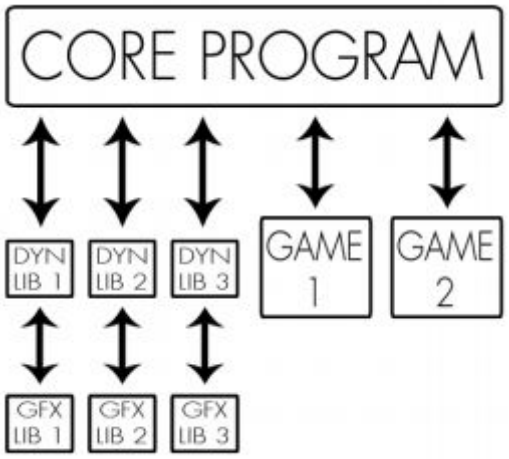
Documentation Arcade :

L’architecture s'articule autour de 3 parties qui ne communiquent jamais entres elles.
Elles sont compilées en librairies partagées (.so), le lien entre ces parties est assuré par le Core.



Chaque librairies graphiques utilisent les mêmes fonctions permettant ainsi à n’importe quel jeu de se lancer dans n’importe quel librairie.

Pour ajouter un jeu, il faut utiliser les prototypes des Interfaces : IGame. Chaque élément affiché à l’écran est une classe, il faut donc recréer une map, des joueurs et des bonus.

De la même façon pour ajouter une méthode d’affichage, il faut recréer une classe qui hérite de l’interface IDisplay.

Exemple : la fonction DrawSquare permet de d’afficher un carré à l’écran. En appelant cette fonction dans le jeu, le carré s’affichera en SFML, Ncurses et en OpenGL.

Pour créer une nouvelle librairie graphique ou librairie de jeu la compiler en .so (librairie dynamique) et la mettre dans le dossier lib (pour une

librairie graphique) ou games (pour une librairie de jeu) à la racine du projet, ainsi le GameCore s'occupera de récupérer la librairie et la lancera.

IGame:

virtual void init_all() -> là ou tout les objets du jeu sont initialisé, type objet, player, ennemies, bonus.

virtual void getevent(Event &evt) -> récupère et stocke les événements de clavier (touches du joueur).

virtual void play() -> commence la boucle de jeu principale.

virtual bool game_over() -> lance le game over.

virtual int getScore() -> récupère la valeur du score en int.

virtual void setScore(int score) -> donne une nouvelle valeur au score.

virtual std::vector ≥tMap() -> récupère la map en brut (chiffres)

virtual std::vector ≥tBonus() -> récupère les bonus du jeu et le nombre de points qu'il rapporte

virtual std::vector ≥tPlayer() -> récupère les positions et l'image du joueur

virtual std::vector ≥tEnemies() -> la même chose que pour le player, mais pour la classe ennemie

virtual void destroy_sprite() -> détruit le sprite après affichage

IDisplay:

virtual void InitWindow() -> initialise la fenêtre d'affichage

virtual void init_map() -> initialise les objets à afficher de la map.

virtual void getevent(Event &evt) -> récupère les évènements clavier.

virtual void create_map(std::vector src_map) -> créé la map du jeu pour l'affichage.

virtual void set_map(std::vector &) -> initialise les objets supplémentaires.

virtual void set_player(std::vector &)-> initialise le player à afficher.

virtual void set_enemy(std::vector &) -> initialise les ennemies à afficher.

virtual void set_bonus(std::vector &) -> initialise les bonus à afficher.

virtual void display_all() -> affiche tout les objets du jeu sur la fenêtre d'affichage.

virtual int game_over() -> affiche l'écran gameover.

virtual int getScore() -> récupère le score à afficher.

Il faut garder à l'esprit que toutes les informations vont transiter par le core (main du jeu).Ce qui veut dire qu'il faut souvent stocker des choses en tampon notamment la map, toutes sortes de positions, et des points.

L'autre point important est le fait de re coder chaque fonction pour chaque nouveau jeu ou librairie graphique, c'est indispensable et contraignant.

Mais c’est également un avantage, car l’architecture du jeu est déjà pensée de façon générique ce qui finalement fait gagner du temps sur la conception de gros projets de jeu.

Cela ajoute également de nouvelles libertés dans l’affichage et dans la façon de gérer toutes la boucle de jeu.