ARCADE

I) Fonctionnement du programme

I.I) Présentation

Le projet arcade est composé de trois grandes parties : le « Core » qui est une station de jeu permettant de les lancer avec des modules graphiques et de sauvegarder les scores. Les jeux qui sont des bibliothèques dynamiques contiennent toutes les données nécessaires au déroulement du jeu. Enfin les modules graphiques permettent d'afficher les jeux et le menu du « Core » dans une fenêtre.

I.II) Assignation des touches

Touches	Action
F1	Module graphique précédant
F2	Module graphique suivant
F3	Jeu précédant
F4	Jeu suivant
Enter	Valider
Suppr	Retour
Les flèches directionnelles	Déplacement dans le menu / Mouvement du joueur
Espace	Action du jeu
R	Reset le jeu
M	Retour au menu
Echap	Quitter l'arcade
PageUp / PageDown Z / S Q / D	Déplacement dans les trois axes pour les modules 3D

II) Ajouter des jeux et des modules graphiques

Les jeux et les modules graphiques doivent être compilés sous forme de bibliothèques dynamiques (.so) avec les flags de compilation -fPIC et - shared. Pour ajouter des jeux et des modules graphiques il faut respecter les règles suivantes :

- Implémenter les interfaces IEntity, IDisplayModule, IGame et IMenu
- Nommer et formatter correctement des fichiers ressources
- Formatter les maps correctement

II.I) Ajouter un module graphique

Pour ajouter un module graphique il faut implémenter l'interface IDisplayModule.

```
namespace arc
{
        class IDisplayModule
        {
        public:
                enum State
                {
                        MENU,
                        GAME
                };
                virtual ~IDisplayModule() = default;
                virtual const std::string &getName() const = 0;
                virtual void init() = 0;
                virtual void stop() = 0;
                virtual void display() = 0;
                virtual State getState() const = 0;
                virtual void setState(State) = 0;
                virtual void setMenu(IMenu *) = 0;
                virtual void setGame(IGame *) = 0;
                virtual const std::vector<arc::IGame::Event> &getEvents() = 0;
        };
}
```

Tout d'abord l'énumération State correspond à l'état de l'affichage, c'est à dire si le module doit afficher le jeu ou le menu du programme. Cet état n'est pas à modifier directement dans le module, c'est le « Core » qui doit s'en occuper. Le State doit être stocké dans le module graphique, il possède un getter et un setter pour y accéder.

Ensuite nous avons la méthode « getName » qui permet d'avoir le nom du module, cette fonction est intéressante par exemple pour afficher le nom du module dans le menu.

Les méthodes « init » et « stop » servent à initialiser et détruire tous les éléments graphiques nécessaires à afficher (fenêtres, textures...). La fonction principale est la fonction « display », celle-ci va être appelée dans la boucle de jeu par le « Core » afin de rafraîchir le contenu de la fenêtre.

Les méthodes « setMenu » et « setGame », comme leurs noms l'indique permette au « Core » d'attribuer un jeu et un menu à afficher au module. Pour finir, la fonction « getEvents » permet de renvoyer au « Core » tous les évènements capturés par le module. Les interfaces IMenu et IGame sont détaillés ci-après

Les ressources doivent avoir pour extension ascii pour les modules ascii, png pour les modules 2D et obj/jpg pour les modules 3D. Le chemin de chaque ressource doit impérativement suivre la règle suivante : res/[nom du jeu]/[nom de l'entité ou valeur de la case de la map][.obj/.png/.ascii].

```cpp
namespace arc
{
    class IMenu
    {
    public:
        enum State
        {
            GAME,
            LIBRARY,
            PLAYER
        };
        virtual ~IMenu() = default;
        virtual const std::vector<std::string> &getLibs() const = 0;
        virtual const std::vector<std::string> &getGames() const = 0;
        virtual int getLibIdx() const = 0;
        virtual int getGamesIdx() const = 0;
        virtual void setLibs(const std::vector<std::string> &) = 0;
        virtual void setGames(const std::vector<std::string> &) = 0;
        virtual void setLibIdx(int) = 0;
        virtual void setGamesIdx(int) = 0;
        virtual State getState() const = 0;
        virtual void setState(State state) = 0;
        virtual const std::string &getName() const = 0;
        virtual void setName(const std::string &name) = 0;
        virtual void setScores(
            const std::map<std::string, std::vector<std::string>> &) = 0;
        virtual const std::map<std::string, std::vector<std::string>> &
        getScores() const = 0;
    };
}
```

L'énumération State correspond aux différents états que peut prendre le menu. C'est à dire si nous sommes en train d'entré notre pseudo, de choisir notre jeu ou notre module graphique. Nous avons « getLibs » et « getGames » qui permettent d'accéder aux noms des jeux et des bibliothèques à afficher dans le menu. Nous avons ensuite, LibIdx, GameIdx, scores, name qui possèdent un getter et un setter. Les deux index servent à se déplacer dans le menu. Le nom correspond au nom de l'utilisateur qui doit être entré dans le menu. Et scores est un tableau de clé/valeur, la clé étant le nom du jeu et la valeur un tableau des cinq meilleurs scores du jeu.

II.II) Ajouter un jeu

Pour ajouter un module graphique il faut implémenter l'interface IGame.

```cpp
namespace arc
{
    class IGame
    {
    public:
        enum Event
        {
            UP,
            DOWN,
            LEFT,
            RIGHT,
            JUMP,
            ENTER,
            RETURN,
            NEXTGAME,
            PREVGAME,
            NEXTLIB,
            PREVLIB,
            PAUSE,
            EXIT,
            RESET,
            MENU
        };
        virtual ~IGame() = default;
        virtual void init() = 0;
        virtual const std::string &getName() const = 0;
        virtual void setName(const std::string &name) = 0;
        virtual const std::vector<std::string> &getMap() const = 0;
        virtual void setMap(const std::vector<std::string> &map) = 0;
        virtual long getScore() const = 0;
        virtual void setScore(long score) = 0;
        virtual std::vector<arc::IEntity *> &getEntities() = 0;
        virtual void
        setEntities(const std::vector<IEntity *> &entities) = 0;
        virtual bool update(const std::vector<Event> &events) = 0;
    };
}
```

Tout d'abord, l'énumération Event définie tous les évènements qui peuvent être interprété par le jeu et le « Core ». Ensuite nous avons la méthode « init » qui permet d'instancier tous les objets nécessaires au jeu. Nous avons aussi nos entités du jeu stockés sous forme d'un tableau d'entités, le nom du jeu stocké dans une chaîne name puis le score dans un entier ; tous ces attributs possèdent un getter et un setter dans IGame. Enfin la fonction « update » prend en entrée les évènements envoyés par IDiplayModule afin de gérer les différents évènements du jeu. Cette fonction est appelée dans la boucle de jeu. Afin que les jeux s'affichent correctement par les modules graphiques, les ressources doivent être fournies pour chaque type de module comme expliqué plus haut. Pour finir, les maps sont stockés dans des tableaux de chaînes, elles possèdent aussi un getter et un setter. Chaque ligne de la map doit avoir la même longueur. Les maps doivent se trouver dans le dossier res/[nom du jeu]/maps/[nom de la map].map

```cpp
namespace arc
{
	typedef struct s_posf
	{
		float x;
		float y;
	} posf_t;

	typedef struct s_pos
	{
		int x;
		int y;
	} pos_t;

	enum Direction
	{
		DIR_UNDEFINED,
		DIR_UP,
		DIR_RIGHT,
		DIR_DOWN,
		DIR_LEFT
	};

	class IEntity
	{
	public:
		virtual ~IEntity() = default;
		virtual const std::string &getName() const = 0;
		virtual void setName(const std::string &name) = 0;
		virtual int getHealth() const = 0;
		virtual void setHealth(const int &health) = 0;
		virtual int getAnimationStatus() const = 0;
		virtual void setAnimationStatus(const int &animation) = 0;
		virtual posf_t getPos() const = 0;
		virtual void setPos(const posf_t &pos) = 0;
		virtual Direction getDirection() const = 0;
		virtual void setDirection(const Direction &direction) = 0;
		virtual Direction getNextDirection() const = 0;
		virtual void setNextDirection(const Direction &direction) = 0;
	};
}
```

Enfin l'interface IEntity, correspond à chaque entité dans le jeu (joueurs, bonus ...). Cette interface est surtout utile pour l'affichage graphique elle permet de récupérer la position dans la map avec « getPos ». Il y a aussi un getter et un setter de « Health », permettant d'accéder aux vies de l'entité. Les entités ont aussi besoin de la direction avec « getDirection » et de « getAnimationStatus » qui permettent de connaître le sens de la texture et l'animation de la texture à afficher. La direction est utilisée dans les trois types de module tandis que l'état d'animation est utilisé seulement pour les modules 2D. Les fichiers png des entités doivent respecter un modèle définit, il est composé de quatre lignes avec une direction chacune (haut, bas, droite, gauche) et sur chaque ligne trois sprites permettant d'animer l'entité.