Documentation Arcade

Présentation du projet

Arcade est une plateforme de jeu, ce programme permet à un joueur de choisir le jeu auquel il souhaite jouer ainsi que la librairie graphique qu'il souhaite utiliser.
Nous avons décider d'implémenter 3 librairies graphiques :

- NCurses
- SDL2
- SFML

Nous avons également implémenté 2 librairies de jeux :

- PacMan
- Nibbler

Compilation du programme

La compilation du programme se fait grâce à un Makefile :

- La commande *« make »* à la racine du repo permet de compiler l'entièreté du projet.
- La commande *« make core »* à la racine du repo permet de compiler seulement le cœur du projet.
- La commande *« make games »* à la racine du repo permet de compiler seulement les librairies des jeux implémentés.
- La commande *« make graphicals »* à la racine du repo permet de compiler seulement les librairies graphiques implémentés.

Tout ce qui est compilé par le Makefile se retrouve dans le dossier ./lib à la racine du repo.

Lancer le programme

Le programme doit être lancé de la manière suivante :

« ./arcade ./lib/[NOM_DE_LA_LIBRAIRIE_GRAPHIQUE].so »

Cela lancera le programme avec comme librairie graphique initiale, celle de votre choix.

Comment jouer

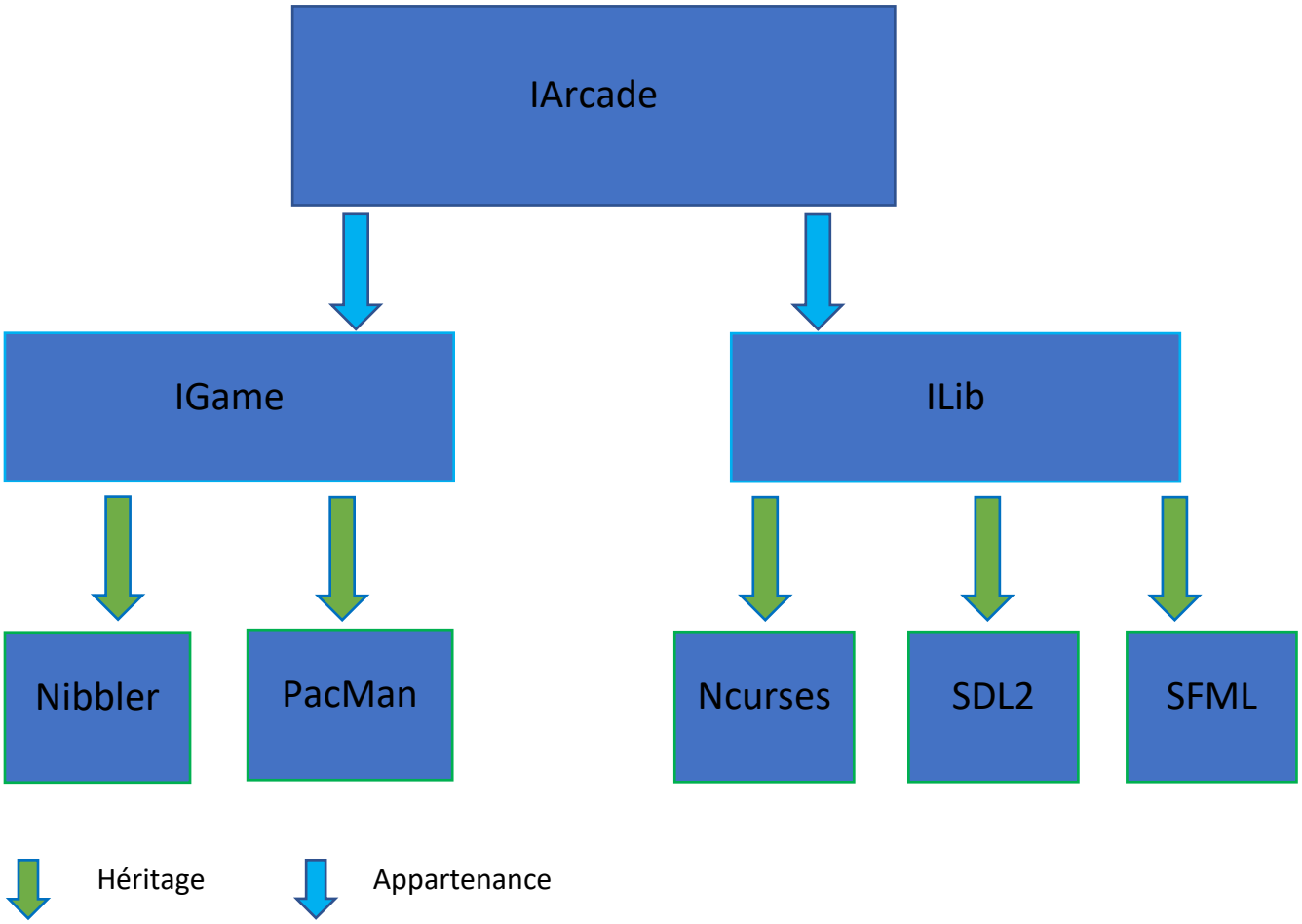
1. Sur le menu
    - La flèche du haut et du bas vous permettent de choisir le jeu auquel vous voulez jouer.
    - Vous pouvez entrer votre pseudo en utilisant les différentes touches du clavier.
    - La touche F1 vous permet de charger la librairie graphique précédente.
    - La touche F2 vous permet de charger la librairie graphique suivante.
    - Le touche Escape vous permet de quitter le programme.

Une fois que vous êtes prêt vous pouvez appuyer sur la touche Entrée afin de lancer votre jeu.

2. Dans un jeu
    - Les flèches vous permettent de vous déplacer.
    - La touche F1 vous permet de charger la librairie graphique précédente.
    - La touche F2 vous permet de charger la librairie graphique suivante
    - Le bouton F3 vous permet de passer au jeu précédent
    - Le bouton F4 vous permet de passer au jeu suivant.
    - La touche F5 vous permet de recommencer le jeu auquel vous être en train de jouer.
    - La touche F6 vous permet de retourner au menu.
    - La touche Escape vous permet de quitter le programme.

Partie développement

Décomposition du projet :



IArcade est une interface qui contient une variable IGame et ILib. Les interfaces Nibbler et Pacman héritent de l'interface modèle IGame afin d'utiliser les mêmes fonctions pour tous les jeux. Cela fonctionne de la même manière pour les librairies graphiques et l'interface ILib. Grace à cette similarité entre les interfaces, nous pouvons lancer n'importe quel jeu avec n'importe quelle librairie.
L'interface IArcade permet d'appeler en boucle :

- La librairie qui récupère l'input rentré par le joueur.
- Le jeu qui réagit en fonction de l'input récupéré auparavant.
- Le jeu qui renvoie la liste de tous les éléments à dessiner sur la fenêtre ainsi que leur position.
- La librairie qui dessine un à un les éléments récupérés auparavant.

Décomposition de la classe modèle de librairie jeu :

Si vous voulez créer un jeu, vous devez recoder toutes les fonctions communes aux jeux qui sont appelées par IArcade.

Les fonctions en rouges, sont les fonctions utilisées par l'interface IArcade.

- ## IGame :

Public :

```
virtual ~IGames() = default;
virtual void update(Key &key) = 0;
virtual std::vector<Object> getObjectList() const = 0;
virtual int move(Key &key) = 0;
virtual int moveAlone() = 0;
virtual int moveUp() = 0;
virtual int moveDown() = 0;
virtual int moveLeft() = 0;
virtual int moveRight() = 0;
virtual int getScore() const = 0;
virtual int getLives() const = 0;
virtual int getWave() const = 0;
virtual void restart() = 0;
virtual bool victory() = 0;
virtual bool defeat() = 0;
virtual int looseLife() = 0;
protected :
clock_t _clock;
std::vector<Object> _objectList;
std::vector<std::string> _map;
int _input;
int _score;
int _direction;
Vect2i _pos;
bool _infinLoop;
int _wave;
int _lives;
float _speed;
```

- ➔ **update(Key &key)** prend une Key en paramètre qui contient l'input rentrée par le joueur et fait réagir le jeu d'un tour de boucle en fonction de cette input.
- ➔ **getObjectList()** cette fonction renvoie le vector d'*Object* créer par le jeu qui contient tous ce qu'il y a à dessiner sur la fenêtre avec leur positions.
- ➔ **getScore()** renvoie le score que le joueur est en train de réaliser au cours de sa partie.
- ➔ **getLives()** renvoie le nombre que le joueur a actuellement au cours de sa partie.
- ➔ **getWave()** renvoie le numéro de la vague dans laquelle se trouve le joueur.
- ➔ **restart()** recommence le jeu à son état initiale.

Afin d'illustrer les jeux par n'importe quelle librairie graphique, les jeux créent un vector composé de la structure *Object_t*.

```
typedef struct Object_t
{
   int objectID;
   std::string texturePath;
   Vect2f fPos;
   Vect2i iPos;
   Rectangle rect;
   const char *character;
   int colorID;
   Vect2f fOffSet;
   char mapChar;
} Object;

typedef struct Rectangle_t
{
   int top;
   int left;
   int width;
   int height;
} Rectangle;

typedef struct Vect2i_t
{
   int x;
   int y;
} Vect2i;

typedef struct Vect2f_t
{
   float x;
   float y;
} Vect2f;
```

Ce vector contient tous ce dont une librairie graphique a besoin afin de d'illustrer les actions du joueur sur le jeu.

Si vous souhaiter créer un jeu compatible avec notre projet, il est nécessaire que votre jeu créer un vector d'*Object_t* et qu'il soit renvoyé dans la fonction : std::vector<Object> getObjectList() const. De plus votre devra avoir les fonctions communes de nos jeux comme : *update, getScore, getLives…* Enfin votre jeu devra être compilé en librairie dynamique (.so) et ajouter dans le dossier ./lib à la racine.

Décomposition de l'interface modèle de librairie graphique

Si vous voulez créer une librairie graphique vous devez recoder toutes ces fonctions.

- ILib :

```
public:
  virtual ~ILib() = default;
  virtual void init() = 0;
  virtual void displayObject(Object &object) = 0;
  virtual void displayBackground() = 0;
  virtual void displayWindow() = 0;
  virtual void destroyWindow() = 0;
  virtual void putText(Text &text) = 0;
  virtual void createObjects(int &gameID) = 0;
  virtual void getInput(Key &input) = 0;
  virtual std::string getLibName() = 0;
  virtual void clearWindow() = 0;
  virtual void destroyObjects() = 0;
```

Les fonctions en rouges, sont les fonctions utilisées par l'interface IArcade.

- ➔ **init()** initialise la librairie graphique.
- ➔ **displayObject(Object &object)** récupère un *Object* (voir plus haut) et qui le dessine sur la fenêtre.
- ➔ **displayBackground()** dessine le background à chaque fois qu'elle est appelée.
- ➔ **displayWindow()** fait apparaitre la fenêtre de la libraire.
- ➔ **destroyWindow()** détruit la fenêtre ainsi que les éléments initialisés par la fonction **init()**.
- ➔ **putText(Text &text)** récupère un *Text* (voir plus bas) et le dessine sur la fenêtre.
- ➔ **createObjectf(int &gameID)** récupère un ID en paramètre et en fonction de cela va charger une texture.
- ➔ **getInput(Key &input)** récupère un *Key* (voir plus bas) en paramètre et le remplis en fonction de la touche rentrée par le joueur.
- ➔ **getLibName()** renvoie le nom de la Lib.
- ➔ **clearWindow()** va clear la Window.
- ➔ **destroyObject()** va détruire la texture qui a été créée par *createObject(int &gameID)*.

```
typedef struct Text_t
{
  std::string fontPath;
  std::string text;
  Vect2i intPos;
  Vect2f floatPos;
  int textID;
  int color;
  int size;
} Text;
```

```
typedef struct Key_t {
   KeyState state;
   KeyCode code;
} Key;

typedef enum KeyState_t {
   StateNone,
   KeyPressed,
   KeyReleased
} KeyState;
```

Keycode est une enum qui comprend toutes les touches qui sont utilisables par le joueur.

Afin de créer une nouvelle librairie graphique utilisable par notre projet, il est nécessaire que votre librairie ai les fonctions communes à toutes les librairies.
Votre librairie graphique peut bien sur utiliser d'autres fonctions.
Attention : ces fonctions ne doivent en rien changer le lien entre IArcade et votre ILib.
De plus elle doit utiliser les informations disponibles dans le vector de *Object_t*. Enfin votre librairie graphique devra être compilé en .so et ajouter dans le dossier ./lib à la racine.

Votre Librairie doit bien sûre être équipé d'un Makefile, qui doit être appelé par le Makefile à la racine.

Ce projet a été réalisé par :

- Titouan Loeb : titouan.loeb@epitech.eu
- Ilan Benarroche : ilan.benarroche@epitech.eu
- Julien Fumey : julien.fumey@epitech.eu