Arcade

Projet EPITECH 2025

Réalisé par

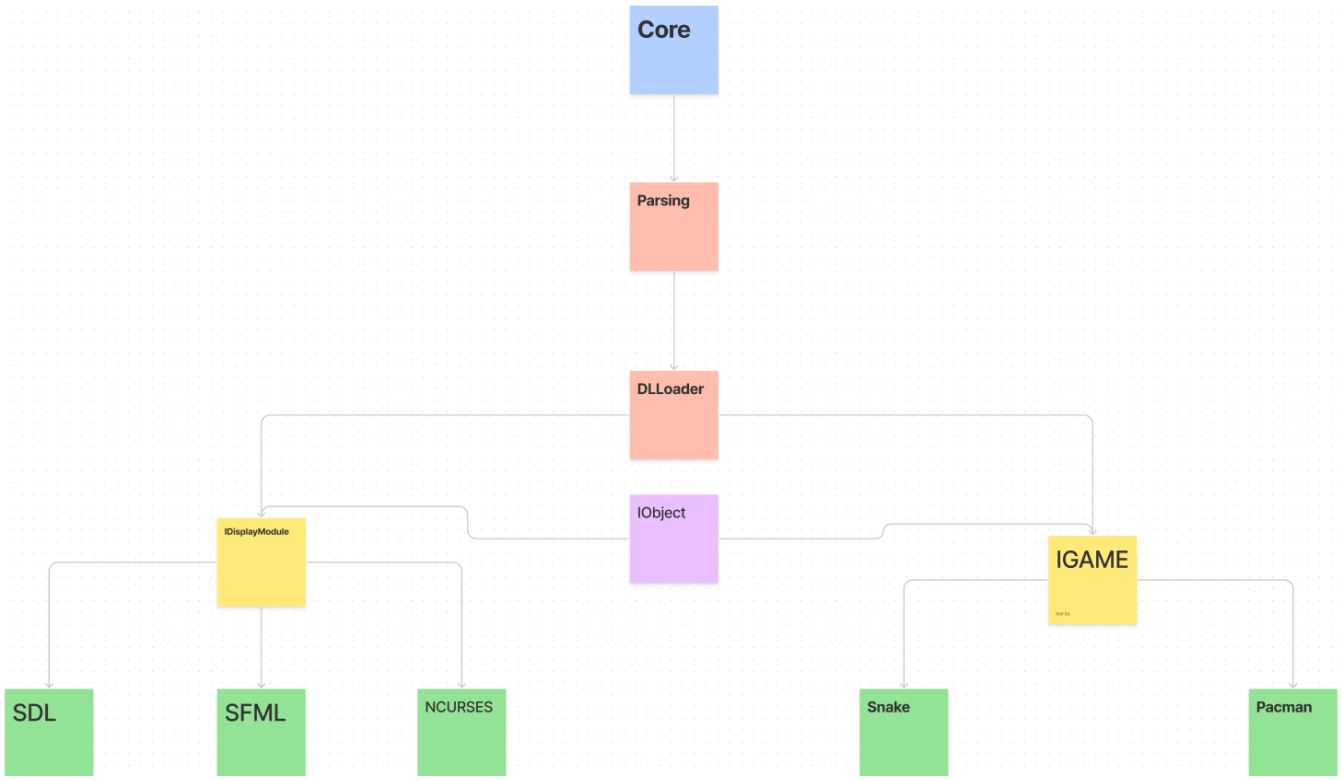
BA Sidi

DELTOUR Géraud

ON Bill

SOMMAIRE :

I) Architecture des Classes :



A) Core :

La partie principale du programme, cette classe permet de charger les librairies graphiques et les jeux.

Liste des fonctions :

- Void load_file()

Vérification du format des librairies

- Void display_lib()

Afficher les informations des librairies utilisées

- Void core_loop()

Gestion des différents éléments de jeu suivant la librairie utilisée

- Void switch_lib(Arcade::Button &input)

Changer de librairie suivant le bouton appuyé

- Void next_game()

Utiliser le jeu suivantdans la liste des jeux disponibles

- Void prev_game()

Utiliser le jeu précédent dans la liste des jeux disponibles

- void next_display()

Utiliser la librairie graphique suivante dans la liste des librairies disponibles

- void prev_display()

Utiliser la librairie graphique précédente dans la liste des librairies disponibles

- void restart_game()

Remettre le jeu actuel dans son état initial

- void init_menu()

Initialiser les composants du menu

- void draw_menu(std::string)

Afficher les composants du menu

- void menu_move(Arcade::Button event)

Gérer les manipulations de librairies suivant la touche utilisée

- void setScore()

Modifier la valeur du score du jeu

- void highScore()

Enregistrer le plus haut score effectué sur un jeu

- void endGame()

Mettre fin à un jeu

- void exit()

Sortir du gestionnaire des librairies et du programme

B) Object :

Entité générique d'un jeu telle que les ennemis ou le joueur

Liste des fonctions :

- std::string getPath() const

Obtenir le chemin vers le sprite de l'entité

- unsigned char getSymbol() const

Obtenir le caractère ASCII associé à l'entité

- Arcade::Color getColor() const

Obtenir la couleur associée à l'entité

- std::pair<float, float> getPos() const

Obtenir les coordonées de l'entité

- void setPath(std::string path)

Définir le chemin vers le sprite de l'entité

- void setSymbol(unsigned char symbol)

Définir le caractère ASCII associé à l'entité

- void setColor(Arcade::Color color)

Définir la couleur associée à l'entité

- void setPos(float x, float y)

Définir les coordonnées de l'entité

C) Sound :

Récupérer et utiliser un fichier audio pour jouer un effet sonore/une musique

Liste des fonctions :

- Std::string getSound() const

Obtenir le chemin vers le fichier audio de la musique

- Void setSound(std::string)

Définir le chemin vers le fichier audio de la musique

D) Text:

Définir un élément textuel à afficher tel que le nom du joueur

Liste des fonctions :

- std::string getText() const

Obtenir le texte du message

- Arcade::Color getColor() const

Obtenir la couleur du message

- std::pair<float, float> getPos() const

Obtenir les coordonnées du message

- void setText(std::string text)

Définir le texte du message

- void setColor(Arcade::Color color)

Définir la couleur du message

- void setPos(float x, float y)

Définir les coordonnées du message

E) DLLoader :

Charger une librairie et vérifier son format

Liste des fonctions :

- Dynamic_close()

Fermer la librairie en libérant la mémoire allouée

- void Dynamic_loader(const std::string &filename)

Charger une nouvelle librairie en libérant la précédente

- void load_sym()

Vérifier que la librairie ait une fonction EntryPoint assurant l'extraction de fonctions

- T *getInstance() const

Obtenir le contenu de la librairie

- void *gethandle() const()

Obtenir le type de la fonction utilisée

F) Menu :

Gérer les différents composants des jeux et de l'interface graphique

Liste des fonctions :

- void addObj(std::shared_ptr<Arcade::IObject> text)

Ajouter un nouvel objet suivant le chemin passé en paramètre

- void addLib(std::shared_ptr<Arcade::Text> text)

Ajouter une nouvelle librairie graphique

- void addGame(std::shared_ptr<Arcade::Text> text)

Ajouter un nouveau jeu

- std::vector<std::shared_ptr<Arcade::IObject>> getMenu()

Obtenir tous les éléments constituant le menu

- std::vector<std::shared_ptr<Arcade::Text>> getLib()

Obtenir tous les composants d'une librairie graphiqe

- std::vector<std::shared_ptr<Arcade::Text>> getGame()

Obtenir tous les composants d'un jeu

II) Comment construire une librairie :

A) Librairie graphique :

Il faut plusieurs fonctions au sein de la librairie nécessaires à l'affichage et gestion d'un jeu.

Il faut premièrement une fonction pour générer une fenêtre de jeu.

Il est ensuite nécessaire d'afficher tous les éléments après avoir effectué les modifications souhaitées par le joueur

Puis il faut pouvoir vider la fenêtre de ses éléments afin d'en dessiner de nouveaux.

Il faut par la suite être capable de dessiner l'arrière-plan sur la fenêtre,

Et de dessiner les différents éléments textuels sans oublier les multiples entités.

Il faut également pouvoir interpréter les différentes commandes du joueur.

Ces différentes fonctions doivent avoir les prototypes suivants :

- void drawObject(Arcade::Object *) override;
- void drawText(Arcade::Text *) override;
- void drawBackground(std::string) override;
- void clear();
- void update();
- void createWindow();
- Arcade::Button getEvent() override;

B) Librairie de Jeu :

Pour assurer le bon comportement du jeu, il faut commencer par initialiser les différentes entités qui le composent. Cela revient à utiliser la classe Arcade::Object pour indiquer :

- Les coordonnées de départ de l'entité
- Sa couleur
- Le caractère ASCII la représentant
- Le chemin vers le Sprite à afficher

Il est par la suite capital d'instaurer une fonction *Play* servant de point central au jeu où le statut des différentes entités sera mis à jour, gérer l'affichage des multiples éléments entre autres.

Il est cependant possible d'incorporer d'autres fonctionnalités telles que le déplacement d'un personnage représentatif du joueur, jouer différents effets sonores ou bien effectuer des actions spéciales en réponse à l'appui d'une touche.

Après l'initialisation et l'utilisation du jeu, il est nécessaire de créer une fonction *endGame* servant à libérer la mémoire allouée lors du jeu ou bien de détecter un événement déclenchant la fin du jeu, tel que la collision entre le joueur et un ennemi.