Table des matières

Le Core 2

- *Explication* 2

Ajouter une librairie graphique 3

- *Choisir sa librairie* 3
- *Utiliser le dossier SDK* 3
- *Développer sa classe graphique* 5
- *Ajouter des ressources (optionnel)* 6
- *Compiler sa librairie graphique* 6
- *Runtime* 6

Ajouter un jeu sur Arcadia 7

- *Utiliser le dossier SDK* 7
- *Choisir le nom de votre jeu* 7
- *Comprendre le SDK* 7
- *Touches pour les jeux* 9
- *Explication sur les Entités* 10

Comment exécuter le programme ? 12

- *Compilation des fichiers* 12
- *Lancement du binaire* 12
- *Interaction avec le menu* 12

Le Core

Explication

- Tous les fichiers situés dans le dossier « **/src** » ne sont pas à modifier car ils composent le **Core !**
- Le **Core** est l'interface qui va faire le lien entre les librairies graphiques situées dans le dossier « **/lib** » et les jeux situés dans le dossier « **/games** ».
- Cependant vous pouvez implémenter vous-même les librairies et les jeux de vos rêves ! Pour cela suivez les différentes étapes mises à votre disposition :
  - L'ajout d'une libraire graphique
  - L'ajout d'un jeu

Ajouter une librairie graphique

Choisir sa librairie

- Installer une librairie graphique
- Créer un dossier pour la nouvelle librairie
- Ajouter un "README" qui explique comment installer cette librairie aux futurs utilisateurs
- Pour être utilisable, le nom du binaire de votre librairie doit être sous la forme de : "lib_arcade_" + **le nom de votre lib** + ".so"

Utiliser le dossier SDK

- Copier le dossier lib du SDK

Ce dossier contient toutes les dépendances nécessaires à la création d'une librairie graphique :

- IDisplayModule.hpp *Interface des différentes librairies graphiques*

- **bool isOpen() const**
  - méthode qui doit renvoyer un booléen correspondant au status de la librairie graphique (ouverte / fermée)
- **void drawEntity(const &IEntity _entity)**
  - méthode qui affiche un élément donné en paramètre
  - l'affichage varie en fonction de la librairie choisi
- **KeyBind eventListener()**
  - méthode qui récupère une touche pressée depuis le clavier/souris
  - renvoie un élément de type KeyBind (voir KeyBind.hpp[1])
- **void oneCycledisplay()**
  - méthode regroupant les actions à faire pendant une boucle d'affichage

[1] correspond aux enums des touches.

- **void displayMenu(const MenuInfo &menuinfo)**
  - méthode affichant le menu principal
  - le menu n'a pas besoin d'entités mais des informations utiles sont envoyer en paramètres (voir MenuInfo[2])

- **void OneCycleClear()**
  - méthode qui permet de nettoyer la fenêtre ou le terminal avant de réafficher

- **void initWindow()**
  - méthode initialisant la librairie graphique

- **void destroyWindow()**
  - méthode détruisant la librairie graphique tout en gardant l'objet actif dans le programme

Les classes des différentes librairies graphiques doivent hériter de cette interface.

- Entity.cpp/hpp et IEntity.hpp

Dépendance utile pour récupérer les positions ainsi que le type d'un « entity » que le « Core » enverra à la librairie

- **getPosX()** et **getPosY()**
  - ces deux méthodes permettent de récupérer les positions

- **getType()**
  - renvoie le type de l'entité

- KeyBind.hpp

Fichier contenant un enum "KeyBind" regroupant les touches intéressantes pour le programme

- Les différentes touches en question sont toutes les lettres de l'alphabet, les flèches directionnelles, la touche ENTREE, ESPACE, RETOUR et de F1 à F6 (voir la liste).

- MenuInfo.cpp/hpp

Classe regroupant les informations concernant le menu principal ainsi que des méthodes permettant de les récupérer

- **const std::vector<std::string> &getGraphList() const**
  - méthode qui permet de récupérer la liste des différentes librairies graphiques du dossier « **/lib** »
  - renvoie un vecteur de string

[2] classe regroupant les informations du menu que la librairie graphique doit afficher

- **const std::vector<std::string> &getGamesList() const**
  - méthode qui permet de récupérer la liste des différents jeux du dossier « **/games** »
  - retourne un vecteur de string

- **const std::vector<std::pair<std::string, std::string>> &getGameScores() const**
  - méthode qui permet de récupérer les scores du jeu actuellement sélectionnés dans le menu
  - renvoie un vecteur de paire de deux strings
  - la première string représente le nom du joueur
  - la deuxième représente le score obtenu

- **const int &getGraphIdx() const**
  - méthode qui permet de connaître l'élément actuellement sélectionné dans le menu par l'utilisateur dans la catégorie des **librairies graphiques**
  - renvoie un int représentant le numéro de la lib

- **const int &getGameIdx() const**
  - méthode qui permet de connaître l'élément actuellement sélectionné dans le menu par l'utilisateur dans la catégorie des **jeux**
  - renvoie un int représentant le numéro du jeu

- **const int &getActiveBoxIdx() const**
  - méthode qui permet de reconnaître le bloc d'élément sélectionné : **librairies** / **jeux** / **nom** / bouton « **jouer** »
  - retourne un int représentant le numéro du bloc

- **const std::string &getPlayerName() const**
  - méthode qui renvoie le nom du joueur sous la forme d'une string
  - le nom est mis à jour à chaque pression sur une touche du clavier

Développer sa classe graphique

- Comme énoncer dans le fichier IDisplayModule.hpp, la classe graphique doit hériter de l'interface IDisplayModule pour être reconnue par le "**Core**"

- Lorsque Arcadia va ouvrir votre librairie, le programme va aller chercher un "entryPoint".

- C'est une fonction qui va renvoyer votre classe principale de la lib.

Voici l'entryPoint à implémenter :

```
extern "C" {
        IGameModule *entryPoint() {
        return (new LibSFML());
        }
}
```

Dans ce cas précis, la librairie implémentée est la libSMFL, et la classe principale de cette lib est la classe LibSFML. Remplacez-la par votre propre nom de librairie !

- Il ne vous manque plus qu'à remplir vos fonctions !

Ajouter des ressources (optionnel)

- Les éventuels fichiers d'images, de son ou de polices de caractères seront placés dans le dossier « **assets/**"Nom de la librairie graphique" »
- Cette étape n'est pas obligatoire mais permet une bonne organisation du projet

Compiler sa librairie graphique

- La compilation se fait à partir d'un Makefile fourni dans le SDK
- Dans le makefile :

  - les nouveaux fichiers devront être ajouter dans la partie "**SRCS**"
  - les éventuelles flags supplémentaires dans le groupe "**CPPFLAG**"

Runtime

- Le binaire de la librairie graphique doit être placé dans le dossier « **/lib** » pour être reconnu par le **Core** et être utiliser en runtime

Ajouter un jeu sur Arcadia

Utiliser le dossier SDK

➢ Copier le dossier « **sdk_games »** du SDK

Ce dossier contient toutes les dépendances nécessaires à la création d'un jeu :

- Un dossier **src** : vous allez avoir besoin de compiler avec un fichier source propre à Arcadia : Entity.cpp.

- Un dossier **include** : Il contient des références et des normes que votre jeu va pouvoir utiliser pour être déployé sur Arcadia.

- Un **Makefile** : Il vous donne une base utilisable pour vous permettre de compiler vos sources.

Choisir le nom de votre jeu

➢ Dans le Makefile, à la ligne 8, vous pouvez modifier le nom de votre jeu.

➢ Pour être utilisable, votre nom doit être sous la forme de : "lib_arcade_" + **le nom de votre jeu** + ".so"

Comprendre le SDK

➢ Pour vous permettre de développer votre jeu, Arcadia propose un **S**oftware **D**evelopment **K**it, qui va vous aider à créer votre jeu.

➢ Voici une explication plus détaillée des fichiers qui vous sont fournis :

- IGameModule.hpp *Interface des différents jeux.*

Pour expliquer son utilité, lorsque Arcadia va ouvrir votre jeu, le programme va aller chercher un "**entryPoint**".

C'est une fonction qui va renvoyer la classe principale du jeu, qui héritera de IGameModule.

Voici l'entryPoint à implémenter :

```
extern "C" {
    IGameModule *entryPoint() {
    return (new Centipede());
    }
}
```

Dans ce cas précis, le jeu implémenté est Centipede, et la classe principale de ce jeu est la classe Centipede.

Celle-ci hérite de IGameModule, et implémente toutes ses fonctions. Voici le fonctionnement du IGameModule :

- **const std::vector<IEntity *> &getEntities()**
  - à chaque appel de cette fonction, le jeu doit renvoyer à Arcadia sous forme de vecteur toutes les entités qu'il souhaite afficher.

- **void receiveEvent(KeyBind event)**
  - lorsque le programme va appeler cette fonction, vous aller recevoir un **KeyBind**.
  - vous allez ainsi pouvoir traiter le **KeyBind** reçu comme vous le souhaitez (exemple : UP_KEY = monter le perso d'une case).

- **bool oneCycleLoop()**
  - c'est votre boucle de jeu. Cet appel de fonction sera limité à une vitesse de 20 FPS (pas plus pas moins).
  - si vous souhaitez effectuer qu'une seule action tous les 3 appels de cette fonction, vous pouvez, mais souvenez-vous que vous ne pourrez pas aller plus vite que 20FPS.
- **void initGame()**
  - c'est la fonction principale.
  - votre constructeur est appelé dès le début, mais une seule et unique fois.

Pour ouvrir et fermer votre jeu, le programme va donc faire appel aux fonctions **initGame()** et **closeGame()**. Celles-ci vous permettront de réinitialiser vos valeurs pour recommencer une future partie.

- **int getScore() const**
  - c'est la fonction permettant à votre jeu de renvoyer votre score au programme.
  - ainsi celui-ci sera en mesure de l'enregistrer lors de la fermeture de votre jeu.

- **void closeGame()**
  - fonction qui va fermer votre jeu, très utile pour remettre vos compteurs à zéro.

Votre jeu doit avoir une classe qui héritera de la classe IGameModule. Toutes les fonctions implémentées dans IGameModule doivent donc être implémentées dans une classe de votre jeu !

- Entity.cpp/hpp et IEntity.hpp

Dépendance utile pour récupérer les positions ainsi que le type d'un « entity » que le « Core » enverra à la librairie

- **Voir ici** pour plus d'explications !
- Une classe qui hérite de IEntity.hpp.
- Elle fournit des fonctions pour l'implémentation d'entités qui seront envoyées à la librairie graphique.

- KeyBind.hpp

Fichier contenant un enum "KeyBind" regroupant les touches intéressantes pour le programme

- **Voir ici** pour plus d'explications !
- Une liste de touches et d'events que votre jeu peut recevoir.
- Malgré tout, vous n'avez pas à implémenter certaines touches car elles ne parviendront pas à votre jeu (voir **Touches pour les jeux**).

- TypeEntity.hpp

Des enum qui permettront à la librairie graphique de savoir quoi afficher

- **Voir** ici pour plus d'explications !

Touches pour les jeux

- Pour récuperer les touches des jeux de n'importe quelle librairie graphique, on utilisera l'enum **KeyBind**, il fournit une norme pour les événements.
- Mais pour les jeux, vous n'allez pas pouvoir utiliser tous les events du KeyBind, certains sont réservés à Arcadia.

- Voici les **événements** (KeyBind) que vous allez pouvoir utiliser pour votre jeu :
  - RETURN,
  - A,
  - B,
  - C,
  - D,
  - E,
  - F,
  - G,
  - H,
  - I,
  - J,
  - K,
  - L,
  - M,
  - N,
  - O,
  - P,
  - Q,
  - R,
  - S,
  - T,
  - U,
  - V,
  - W,
  - X,
  - Y,
  - Z,
  - SPACE,
  - UP_KEY,
  - DOWN_KEY,
  - LEFT_KEY,
  - RIGHT_KEY,
  - VALID,
  - NO_EVENT

Explication sur les Entités

- Pour que la librairie puisse afficher ce qu'il faut au bon endroit, il faut lui envoyer des informations.

- Ces informations doivent être formatées pour que chaque librairie comprenne ce qu'il faut afficher.

- C'est à cela que va servir la Classe Entity :

  - Pour créer un **entity**, il vous faut :

    - une **position** : posX et posY (compris dans la taille de la map).

    - un **TypeEntity**

- Pour les **TypeEntity**, vous avez le choix parmi tous les enums proposés :
  - PLAYER_1,
  - ENEMY_1,
  - ENEMY_2,
  - ENEMY_3,
  - WALL_1,
  - FLOOR,
  - BONUS_1,
  - BONUS_2,
  - BONUS_3,
  - PROJECTILE,
  - MAP,
  - WALL_2,
  - WALL_3,
  - WALL_4,
  - PLAYER_2,
  - PLAYER_3,
  - PLAYER_4,
  - BACKGROUND,
  - GAME_WON,
  - GAME_LOST,
  - TITLEMENU,
  - TITLEGAME,
  - ERROR

Chaque librairie affichera votre enum à sa façon, renseignez-vous sur les différents affichages des librairies proposées, ou alors développez la vôtre 😉.

Les positions de l'enum MAP correspondent à la taille voulue de la map du jeu (exemple : 30 par 30 pour notre centipède). L'entité MAP est un point de repère pour les autres entités. Ainsi il faut l'envoyer avant toutes les autres à la librairie graphique !

Comment exécuter le programme ?

Compilation des fichiers

La compilation s'effectue avec le makefile à la base du répertoire. Il appelle le makefile à l'intérieur du dossier « **lib/** » qui lui-même appelle le makefile de chaque librairie pour créer les binaires de celles-ci.

- « **make** », à la base du répertoire
- « **make re** », si des modifications ont été effectuées

Lancement du binaire

- ./arcade **chemin-vers-une-lib-graphique.so**
- Les binaires des librairies sont situées dans le dossier « **lib/** »

Interaction avec le menu

- Les touches pour interagir avec le menu dépendent de la librairie choisie.
- Pour les librairies pré implémentées voici les touches :

- LibSfml, LibNcurses et LibAllegro

- F1 *Lib précédente*
- F2 *Lib suivante*
- F3 *Jeu précédent*
- F4 *Jeu suivant*
- F5 *Redémarrer le jeu*
- ← *Aller à gauche*
- ↑ *Aller en haut*
- → *Aller à droite*
- ↓ *Aller en bas*
- Touche « Entrée » *Valider*
- Toutes les touches alphabétiques