ARCADE

A- Implémentation d'un jeu

---

1 - Création d'un jeu

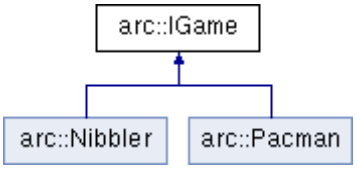
Pour implémenter une nouveau jeu entièrement compatible avec notre programme ARCADE, vous allez devoir suivre certaines instructions.

Tout d'abord, le jeu que vous devez implémenter doit être écrit en C++. Il doit par la suite être compilé en une librairie dynamique (avec une extension .so). Le nom de ce jeu devra être de ce format : **lib_arcade_$NAME.so**, en remplaçant $NAME par le nom de votre jeu. Pour que notre programme ARCADE puisse le trouver, vous devez la mettre dans le **dossier games** situé à la racine du projet.

2 - Description de l'interface IGame

Entrons maintenant un peu plus en détails sur la création d'un jeu. Celui-ci doit contenir une classe héritant de l'interface **IGame** décrite juste après ce paragraphe, que vous pouvez retrouver dans le dossier include situé à la racine du projet. Vous devez implémenter toutes les fonctions, sans cela votre librairie ne pourra donc pas marcher avec notre Arcade.

Exemple:



size_t getMapHeight() const

Cette fonction retourne la hauteur de la map de votre jeu. Sa taille maximale ne doit pas dépasser 51 pour être compatible avec notre librairie ncurses.

size_t getMapWidth() const

Cette fonction retourne la largeur de la map de votre jeu. Sa taille maximale ne doit pas dépasser 68 pour être compatible avec notre librairie ncurses.

const std::string &getMusic() const

Cette fonction retourne le filepath de la musique de votre jeu.

const std::string &getSound() const

Cette fonctionne retourne le filepath du sound design de votre jeu.

const std::string &getScore() const

Cette fonction retourne le score actuel de votre jeu.

const std::map<std::pair<Event::Type, Event::Key>, std::function<void()>> &getControls() const

Cette fonction retourne une map contenant une paire d'événement (défini dans le fichier Utils.hpp situé dans le dossier include à la racine du projet) et une fonction associée à cette paire.

const std::vector<std::shared_ptr<Entity>> &getEntities() const

Cette fonction retourne la liste d'entités qui sera par la suite affiché par une librairie graphique. Cette fonction sera appelé à chaque tour de boucle lorsque le jeu est lancé afin de récupérer des informations mises à jour. La structure Entity est défini dans le fichier Utils.hpp situé dans le dossier include à la racine.

const std::vector<std::pair<std::string, std::string>> &getGameControls() const

Cette fonction retourne une liste de paire de string. L'élément premier de la paire correspond à une touche et le deuxième élément de la paire correspond à une description de l'action que permet de réaliser la touche en question.

const std::vector<std::pair<std::string, std::string>> &getGameStats() const

Cette fonction retourne une liste de paire de string. Cette fonction a pour but de retourner une liste de string correspondant à des statistiques présentes dans votre jeu. Cela peut être un nombre de consommables mangé ou bien le nombre d'ennemis sur la map. A vous de choisir les statistiques que vous souhaitez voir s'afficher.

void restart()

Cette fonction permet de recommencer votre jeu dès le début. Il est donc important de réinitialiser vos valeurs de départ.
Elle est appelée lorsque le joueur clique sur un bouton restart.

void updateGame()

Cette fonction met à jour les informations du jeu. Elle est en effet appelé à chaque tour de boucle lorsque le jeu est lancé.
Il est important de mettre votre score à jour, mais également vos entitées (comme ses positions par exemple).

bool isGameOver() const

Cette fonction renvoie un booléen. Elle permet d'indiquer au Core si le joueur a perdu ou non. Il faut donc renvoyer "TRUE" lorsque les règles de votre jeu correspondent à une défaite.

const std::string &getTitle() const

Cette fonction renvoie le nom de votre jeu.

3 - Les touches

Vous pouvez retrouver les touches implémentables dans le fichier Utils.hpp situé dans le dossier include à la racine du projet. Cela vous permettra de créer les contrôles nécessaires pour votre jeu.

4 - Description de la structure entitée

La structure Entity défini dans le fichier Utils.hpp dans le dossier include situé à la racine du projet, contient plusieurs membres que la jeu devra initialiser pour son utilisation avec une librairie graphique.

spritePath: C'est une chaine de caractère qui indique le chemin du sprite que vous souhaitez pour votre entité. Vous pouvez également ne pas en mettre en initialisant avec une chaine de caractère vide.

backgroundColor: C'est une structure définie dans le fichier Utils.hpp qui indique la couleur de l'entité à afficher.

orientation: C'est une énumération définie dans le fichier Utils qui permet de connaître l'orientation de l'entité.

type: C'est une énumération définie dans le fichier Utils qui permet de savoir le type de l'entité.

x: Sa position sur l'axe des abscisses.

y: Sa position sur l'axe de ordonnées.

5 - Finalisation

Pour finir et pour que notre Arcade puisse créer votre jeu, vous devez créer une fonction C compilé avec votre librairie pour retourner votre classe nouvellement créer. Cette fonction devra s'appeler **instance_ctor** et devra retourner un pointeur d'IGame comme ci-dessous.

```
extern "C" arc::IGame *instance_ctor()
{
    return (new arc::Nibbler());
}
```

B- Implémentation d'une librairie graphique

---

1 - Création de la librairie graphique

Pour implémenter une nouvelle librairie graphique entièrement compatible avec notre programme ARCADE, vous allez devoir suivre certaines instructions.

Tout d'abord, la librairie que vous voulez implémenter doit être écrite en C++. Elle doit par la suite être compilé en une librairie dynamique (avec une extension .so). Le nom de cette librairie devra être de ce format : **lib_arcade_$NAME.so**, en remplaçant $NAME par le nom de votre librairie. Pour que notre programme ARCADE puisse la trouver, vous devez la mettre dans le **dossier lib** situé à la racine du projet.

2 - Description de la librairie graphique

Entrons maintenant un peu plus en détails sur la création de la librairie. Celle-ci doit contenir une classe héritant de l'interface **IGraphical** décrite juste après ce paragraphe, que vous pouvez retrouver dans le dossier include situé à la racine du projet. Vous devez implémenter toutes les fonctions, sans cela votre librairie ne pourra donc pas marcher avec notre Arcade.

void display()

Cette fonction est appelé à chaque tour de boucle dans notre Arcade. Le but est de mettre à jour le jeu et de gérer les différents évènements de la librairie.

Event::Type getEventType() const

Cette fonction retourne le type de l'événement de l'utilisateur. Nos événements sont définis dans le fichier Utils.hpp situé dans le dossier include à la racine du projet.

Event::Key getKeyPressed() const

Cette fonction retourne l'événement que l'utilisateur a émis.

void setListGames(const std::vector<std::string> &games, const std::function<void (const std::string &)> &fct, int chosen = -1)

Cette fonction envoie à la librairie une liste des jeux disponibles, une fonction à exécuter avec le nom du jeu en paramètre quand l'utilisateur clique sur un jeu. Et enfin, l'entier chosen (défini à -1 par défaut, ce qui correspond à aucun) permets de savoir quelle jeu a été choisi par l'utilisateur précédemment.

void setListLibraries(const std::vector<std::string> &libraries, const std::function<void(const std::string &)> &fct, int chosen = -1)

Même instruction que pour setListGames mais avec les librairies graphiques.

void setScores(const std::vector<std::pair<std::string, std::string>> &scores)

Cette fonction envoie la liste des scores du jeux. Le première chaîne de caractère correspond au nom de l'utilisateur et la deuxième au score. Elle est appelé lorsque l'utilisateur clique sur un jeu et à la fin d'un jeu. Libre à vous de décidez quand afficher cette liste.

void setControls(const std::map<std::pair<Event::Type, Event::Key>, std::function<void ()>> &controls)

Cette fonction envoie une map contenant une paire d'événement (défini dans le fichier Utils.hpp situé dans le dossier include à la racine du projet) et une fonction à exécuter si l'utilisateur a émis cette paire d'évènement.

void setFunctionPlay(const std::function<void()> &function)

Cette fonction envoie la fonction à exécuter si l'utilisateur clique sur bouton Play.

void setFunctionRestart(const std::function<void()> &function)

Cette fonction envoie la fonction à exécuter si l'utilisateur clique sur bouton Restart.

void setFunctionMenu(const std::function<void()> &function)

Cette fonction envoie la fonction à exécuter si l'utilisateur clique sur le bouton Menu.

void setFunctionTogglePause(const std::function<void()> &function)

Cette fonction envoie la fonction à exécuter si l'utilisateur clique sur le bouton Pause.

const std::string &getUsername()

Cette fonction renvoie le nom d'utilisateur.

void setUsername(const std::string &username)

Cette fonction envoie le nom d'utilisateur.

Scene getScene() const

Cette fonction renvoie la scène dans lequel se trouve l'utilisateur. Scene est une énumération que l'on peut retrouver dans le fichier IGraphical. En tout, il y a trois principales scènes, le menu, le jeu et la fin du jeu.

void setScene(Scene scene)

Cette fonction envoie la scène que la librairie doit afficher.

void setHowToPlay(const std::vector<std::pair<std::string, std::string>> &info)

Cette fonction envoie une liste de paire de chaînes de caractères pour indiquer à l'utilisateur les événements. La première chaîne correspond à la description et la deuxième à l'événement.

void setGameStats(const std::vector<std::pair<std::string, std::string>> &info)

Cette fonction envoie à l'utilisateur les stats du jeu. La première chaîne de caractère correspond à la description et la deuxième à la valeur. Cette fonction est appelé à chaque tour de boucle lorsque le jeu est lancé.

void updateGameInfo(const std::vector<std::shared_ptr<Entity>> &gameMap)

Cette fonction envoie la liste d'entités qu'il faudra afficher sur la map du jeu. Cette fonction est appelé à chaque tour de boucle lorsque le jeu est lancé. La structure Entity est définie dans le fichier Utils.hpp situé dans le dossier include à la racine.

void setMapSize(size_t height, size_t width)

Cette fonction envoie la taille de la map que le jeu veut afficher.

void setGameTitle(const std::string &game)

Cette fonction envoie le titre du jeu qui est lancé.

void setGamePause(bool pause)

Cette fonction vous indique si le jeu doit être mis en pause ou pas.

3 - Les événements

Pour que votre librairie soit entièrement compatible avec le coeur de notre programme, il y a des événement que vous devez renvoyer pour son bon fonctionnement.

Type: KEY_PRESSED
Event: NUM0
Description: Charge la librairie suivant.

Type: KEY_PRESSED
Event: NUM1
Description: Charge la librairie précédent.

Type: KEY_PRESSED
Event: NUM9
Description: Charge le jeu suivant.

Type: KEY_PRESSED
Event: NUM2
Description: Charge le jeu précédent.

Type: KEY_PRESSED
Event: PAUSE
Description: Met en pause le jeu.

Type: KEY_PRESSED
Event: R
Description: Redémarre le jeu.

Type: KEY_PRESSED
Event: M
Description: Retour au menu.

4 - Description de la structure entitée

La structure Entity défini dans le fichier Utils.hpp dans le dossier include situé à la racine du projet, contient plusieurs membres que la librairie graphique devra traduire en fonction des ses propres configurations.

spritePath: C'est une chaine de caractère qui indique le chemin du sprite à charger. Si votre librairie ne peut charger ce sprite, il faut se référencer à la couleur.

backgroundColor: C'est une structure définie dans le même fichier qui indique la couleur de l'entité à afficher.

orientation: C'est une énumération définie dans le même fichier qui permet de connaître l'orientation de l'entité.

type: C'est une énumération défini dans le même fichier qui permet de savoir le type de l'entité.

x: Sa position sur l'axe des abscisses.

y: Sa position sur l'axe de ordonnées.

5 - Finalisation

Pour finir et pour que notre Arcade puisse créer votre librairie, vous devez créer une fonction C compilée avec votre librairie pour retourner votre classe nouvellement créer. Cette fonction devra s'appeler **instance_ctor** et devra retourner un pointeur d'IGraphical comme ci-dessous.

```
extern "C" arc::IGraphical *instance_ctor()
{
    return (new arc::Graphical());
}
```

Enfin, si vous avez des problèmes pour implémenter votre librairie graphique et/ou votre jeu, n'hésitez surtout pas à nous contacter à nos adresses ci-dessous.

Bon courage !

amaury.lecomte@epitech.eu
oriane.aumoitte@epitech.eu
celeste.bousseau@epitech.eu