MyArcade

Overview

MyArcade is an arcade emulator designed to use differents graphicals and games libraries. The librairies are loaded from /games and /lib directories by the core program at run time. Currently, there are only 2 games (Pacman and Nibbler) and 3 graphicals modules (ncurse, SFML and SDL 2.0) but this documentation will help you integrate some more.

Implementation

To start implementing more modules, you must compile your code in a different Makefile and link it into a shared object (.so) otherwise the core program will not load it properly. Your library must have a correct constructor and destructor which will be call by dlopen, dlsym and dlclose to instantiate and destroy your class.

```
void __attribute__((constructor)) calledFirst();
void __attribute__((destructor)) calledLast();
```

You must also provide an “entryPoint” function which will be used to retrieve a pointer to your class.
For a graphical module :

```
IDisplayModule *LibEntryPoint();
```

or for a game module :

```
IGameModule *GameEntryPoint();
```

Alternatively, you can implement your own `ICoreProgram` module.

ICoreProgram implementation

The core program is used to load libraries from directories (defined in `Paths.hpp` file) and

switch between graphicals libraries or games.
It must respect the following interface :

```
class ICoreProgram {
public:
 virtual ~ICoreProgram() = default;
 virtual void setGraphicLib(const std::string &) = 0;
 virtual void setGameLib(const std::string &) = 0;
 virtual void pushGraphicName() = 0;
 virtual void pushGameName() = 0;
 virtual void pushScores() = 0;
 virtual void nextGraphicLib() = 0;
 virtual void nextGameLib() = 0;
 virtual void prevGraphicLib() = 0;
 virtual void prevGameLib() = 0;
 virtual void run() = 0;
};`
```

The `pushGraphicName`, `pushGameName`, `pushScores` functions must be linked to the `IDisplayModule`'s `setGraphicLibName`, `setGameLibName`, `setScore` functions.
While loading shared object with dlopen, you must also save a pointer to the handler with `IDisplayModule` and `IGameModule` function `setHandler` in order to unload the libraries at the end of execution.

> `prevGraphicLib`, `prevGameLib`, `nextGraphicLib`, `nextGameLib` are directly called by `IDisplayModule` and must clean and initiate properly `IGameModule` and `IDisplayModule` otherwise this will result in dead windows and crashes.
> In our implementation, this is handled by `setGraphicLib` and `setGameLib` the following way :
>
> ```
> void MyArcade::setGraphicLib(const std::string &libName)
> {
>  if (this->_currentGraphicLib)
>   this->_currentGraphicLib->destroy();
>  this->_currentGraphicLib = this->_graphicLibraries[libName];
>  if (!this->_currentGraphicLib) {
>   //throw error
>  }
>  this->_currentGraphicLib->init(this);
>  if (_currentGameLib)
>   this->_currentGameLib->setGraphicalLib(this->_currentGraphicLib);
> }
> ```

```
void MyArcade::setGameLib(const std::string &libName)
{
 this->_currentGameLib = this->_gameLibraries[libName];
 if (!this->_currentGameLib) {
  //throw error
 }
 this->_currentGameLib->init();
 this->_currentGameLib->setGraphicalLib(this->_currentGraphicLib);
}
```

`void MyArcade::run()` is called by the main function and run a loop with `IDisplayModule::displayMenu` and `IGameModule::runGame`.

> Our implementation :
>
> ```
> void MyArcade::run()
> {
>  int menuValidate = 0;
>  int runGame = 0;
>
>  while (1) {
>   while (!menuValidate)
>    menuValidate = this->_currentGraphicLib->displayMenu();
>   runGame = 1;
>   while (runGame != 0) {
>    this->_currentGameLib->init();
>    runGame = this->_currentGameLib->runGame();
>   }
>   menuValidate = 0;
>  }
> }
> ```

IDisplayModule implementation

To implement a new graphical module, you must use the following interface :

```
class IDisplayModule {
public:
 virtual ~IDisplayModule() = default;
 virtual void init(ICoreProgram *) = 0;
```

```
 virtual void destroy() = 0;
 virtual void setHandler(void *) = 0;
 virtual void setGraphicLibName(std::vector<std::string>) = 0;
 virtual void setGameLibName(std::vector<std::string>) = 0;
 virtual void setScore(std::vector<std::string>) = 0;
 virtual void setUserName(std::string userName) = 0;
 virtual void *getHandler() const = 0;
 virtual const std::string &getName() const = 0;
 virtual const std::string &getUserName() const = 0;
 virtual void updateDisplay(IGameModule *) = 0;
 virtual int displayMenu() = 0;
 virtual void printText(std::string, long = 2000000) = 0;
 virtual std::string getKey() = 0;
 virtual void keyAction(std::string) = 0;
};
```

void init(ICoreProgram *)

This function is called to initiate the graphics (load window, load sprites ...). It takes a pointer to the `ICoreProgram` as argument in order to modify it later.

void destroy()

This function is used to destroy everything about your graphics. Your class must still be able to be reloaded with another `init` call.

void updateDisplay(IGameModule *)

This function is called by the game module to print the game map. It takes a pointer to `IGameModule` as argument to be able to retrieve all the informations you need.

std::string getKey()

This function is called by the game module to retrieve user input. It must respect a standard format defined in `DefaultKey.hpp`. It's simply the SDL key names in lowercase.
[SDL Keycodes]

> To convert the "normals" keys or any others undefined keys, you can use a function like the following :
>
> ```
> std::string NcurseDisplayModule::translateKey(int key)
> {
>  if (this->_keyDictionary[key] == "") {
>   this->_keyDictionary[key].push_back(key);
> ```

```
  std::transform(this->_keyDictionary[key].begin(),
  this->_keyDictionary[key].end(),
  this->_keyDictionary[key].begin(),
  ::tolower);
 }
 return (std::string(this->_keyDictionary[key]));
}
```

void keyAction(std::string)

This function is used to map internals actions from the graphical library (next lib, game ...) by calling the appropriate `ICoreProgram` functions. The defaults keys to use are defined in `DefaultKeys.hpp`. To make theses actions available from game, your must call `KeyAction` in `getKey` function.

> The expected behavior is the following :
>
> ```
> void SFMLDisplayModule::keyAction(std::string key)
> {
>  if (key == EXIT_KEY) {
>   this->destroy();
>   exit(0);
>  }
>  if (key == NEXT_LIB_KEY) {
>   this->_coreProgram->nextGraphicLib();
>  }
>  if (key == NEXT_GAME_KEY) {
>   this->_coreProgram->nextGameLib();
>  }
>  if (key == PREV_LIB_KEY) {
>   this->_coreProgram->prevGraphicLib();
>  }
>  if (key == PREV_GAME_KEY) {
>   this->_coreProgram->prevGameLib();
>  }
> }
> ```

void printText(std::string, long = 2000000)

This is the function used to draw text from game (like game overs ...). It clear the screen and print the text in the middle of it. The second argument is the timeout.

int displayMenu()

This is the function called by `ICoreProgram` to show the library menu. It must let the user select his game, graphical library and enter his name.
It must return 0 if the program quit without selecting properly and 1 when everything is fine.

IGameModule implementation

To implement a new game, you must use the following interface :

```
class IGameModule {
public:
 virtual ~IGameModule() = default;
 virtual void init() = 0;
 virtual void setHandler(void *) = 0;
 virtual void *getHandler() const = 0;
 virtual void setGraphicalLib(IDisplayModule *) = 0;
 virtual const std::string &getName() const = 0;
 virtual void keyAction(std::string) = 0;
 virtual const std::vector<std::string> getGameMap() const = 0;
 virtual const std::vector<objPos_t> getObjPos() const = 0;
 virtual int getScore() const = 0;
 virtual std::string getUserName() const = 0;
 virtual void saveScore() const = 0;
 virtual ITilemap *getTilemap() const = 0;
 virtual int runGame() = 0;
 virtual std::pair<double, double> getMapSize() const = 0;
};
```

void init()

This function must (re)set your game. It's called when loading a new game or simply restart.

void setGraphicalLib(IDisplayModule *)

This function is used to attach the graphical display to the current game. It must be call before printing anything or at a graphical library change.

void saveScore() const

This function save the current score in a file, called the same as the return of the `getName` function, in the directory specified in `Paths.hpp`.

void keyAction(std::string)

This function is similar to `IDisplayModule::keyAction`, it must be used with the return of `IDisplayModule::getKey`.
It's important that the key mapped to library builtins stop the game for the change to be effective.
(e.g)

```
void PacMan::keyAction(std::string key)
{
 if (key == NEXT_GAME_KEY) {
  saveScore();
  this->_state = 2;
 }
 if (key == PREV_GAME_KEY) {
  saveScore();
  this->_state = 2;
 }
 if (key == MENU_KEY) {
  saveScore();
  this->_state = 0;
 }
}
```

Jointly with :

```
int PacMan::runGame()
{
 this->_state = 1;
 while (this->_state == 1) {
  this->_graphicModule->updateDisplay(this);
  this->keyAction(this->_graphicModule->getKey());
  if (this->_state != 1)
      return (this->_state);
 }
 return (this->_state);
}
```

const std::vector< std::string > getGameMap() const

This function return the game map in a array of strings.

const std::vector<objPos_t> getObjPos() const

This function return all the movable objects to print on the map in a array.

> The definition of objPos_t is :
>
> ```
> typedef struct objPos_s {
>     double x;
>     double y;
>     char value;
>     std::string name;
>    } objPos_t;
> ```

ITilemap *getTilemap()

This function return the tilemap to use with the game.
(see `ITilemap` implementation)

int runGame()

This is the function called by ICoreProgram to launch a game. It must call `IDisplayModule::updateDisplay` and `IDisplayModule::getKey` to properly interract with graphical module.
It must return 0 to return to `displayMenu` loop or any other value restart the `runGame` loop.
(see `ICoreProgram::run`)

ITilemap implementation

In order to fully implement a game, you must implement a `ITilemap` interface inside the `IGameModule` interface.
The interface is the following :

```
class ITilemap{
public:
 virtual ~ITilemap() = default;
 virtual int getScale() const = 0;
 virtual std::map<char, std::pair<int, int>> getTilemap() const = 0;
 virtual std::string getTilemapPath() const = 0;
};
```

int getScale() const

This function return the size of the side of one tile.

std::map<char, std::pair<int, int>> getTilemap() const

This function return the position (upper left corner pixel) of a tile in the file, associated with a char.
(e.g)

```
_tilemap['C'] = std::make_pair(64, 85);
_tilemap['c'] = std::make_pair(1, 64);
_tilemap['v'] = std::make_pair(1, 127);
_tilemap['V'] = std::make_pair(1, 107);
```

std::string getTilemapPath() const

This function return the path of the tilemap.