Victor DUBRET
Vincent ROYE
Thibaut TROUVE

# Arcade

## I How to create libraries

To create new games or new graphicals interface, you need to use 2 interfaces.

IGame is the interface for the game, this interface handle the game algorithm.

IDisplay is the interface for the graphical libraries.

Also we have a namespace « ar ». In this namespace we have an enum for the Event. This allow for exemple in the core we don’t really know wich graphical library is load, but with this enum we know if we do a key up and we don’t care about the library.

When you will create game your binary must match with the regular expresion lib_arcade_$GAMENAME.so, the library must be put in the ./games file, and for graphical library lib_arcade_$GAMENAME.so the library must be put in the ./liblib_arcade_$GAMENAME.so

## II Graphic Library

To launch a graphical library with the core, the implementation of some functions is required :
(You MUST override every function)

**void initMenu(const std::vector<std::string> &games, const std::string &menuName, const std::vector<std::string> &libs)**

→ Initialize the menu. The two vectors are the vectors who will be displayed in the menu.
→ One for the game content in the « ./games » and an other for graphical lib content in the « ./lib ».
→ initMenu will always be called before **refreshMenu**

***int refreshMenu(const ar::Event &key, const std::vector<userInterface> &dataArray)***

→ This function will display the menu.
→ This function take Event as parameter, because the function handle the events *AR_UP*, *AR_DOWN*, *AR_VALIDATE. They are used to select the game.*
*→ The std::vector is the best scores save in the current session*
*→ The refreshMenu return the index of the game selected*

***void destroyMenu()***

*→ This function destroy the resource needed by the menuCh*

***void getEvent(int &realEvent)***

*→ Get the event, and return an ar::Event corresponding to the Event bind.*
*→ The realEvent is used to set an username in the menu, with the function* ***refreshUserName***

***void refreshUsername(std::string &name, int realKey)***

→ Update the name of the current user, depending on the key value, only call in the menu.
**bool canHandleSprites()**

→ Check if the library handle sprites.

***void loadResources(const std::map<unsigned char, colorVector> &colors)***

***void loadResources(const std::string &filePath, const std::map<unsigned char, spriteCoords> &sprites)***

→ loadResources functions is call before loading a game, if the graphic library load handle sprite, the load Resources with two arguments will be called.

***void displayGame(const userInterface &UI, Map &map)***

→ This function is called when all the game ressources is set. It displays the map modified by the game.

→ UI parameter content the score, the name and the timer of the user.

***typedef ar::IDisplay *createDisplay()***

→ The typedef for the extern « C » function needed, the function must match with the symbole « createDisplay »
→ this function return a pointer to the graphical library instance

***typedef void destroyDisplay(ar::IDisplay *)***

→ The typedef for the extern « C » function who destroy the display. The function must match with the symbole « destroyDisplay »

## III Game Library

***void manageKey(const Event &key)***

*→ function takes an event in parameter and handle the event, depending on the game.*

***const std::map<unsigned char, spriteCoords> &getSprites() const***

*→ function to get the sprites depending on the number of the Map :*
*→ exemple, number 3 in the map will load the Mario sprite.*

***const std::string getSpritesPath() const***

→ function to call if the graphic library handle sprites, return the path where load the sprites

***const std::map<unsigned char, colorVector> &getColors()***

→ This function set the color depending on the number on the map

***int refreshScore()***

→ This function refresh the score since the game started

***int refreshTimer()***

→ This function refresh the timer since the game started

***bool isGameOver()***

→ this function return true if the GameOver condition is validate

***Map &getMap()***

→ getter to the map of the game

***void loop()***

→ loop is the main algorithme of the game, is modified the map, which is display in the graphic

***const std::string getGameName() const***

→ getter to the Game name

***void setPause()***

→ Set or unset the pause

***typedef ar::IGame *createGame()***

→ The typedef for the extern « C » function needed, the function must match with the symbole « createGame »
→ this function return a pointer to the Game instance

***typedef void destroyGame(ar::IGame *)***

→ The typedef for the extern « C » function who destroy the display. The function must match with the symbole « destroyGame »