How To Create a New Game?

Any game class must be derived from the AGameModule class.

Your class constructor must take a single parameter, a reference to an ArcadeComponent, which must be pass to the constructor of the AGameModule.

Your class **must** implement the following method:

- void update()
  The main function of your game. It is call one per frame.

For more information about the events and the display, refer to the AGameModule section.

What is an AGameModule?

An AGameModule is in abstract class use by Game classes. It manages interactions between users and games.

It is constructed with a reference to an ArcadeContent and a path to a Texture Describer. The Texture Describer file must be the one associated with your game. More information on this file on the Texture Describer section.

An AGameModule possesses the following protected functions:

- **Events managing:**
  - Bool isKeyPressed(), isKeyReleased() const:
    Return True if a key is respectively pressed or released, false otherwise.
  - Bool isKeyUp(), isKeyRight(), isKeyDown(), isKeyLeft(), isKeyEnter(), isKeyBack() const:
    Return True if the corresponding key is triggered, false otherwise.
  - Void backToMenu():
    Exit the game and return to the main menu.

- **Map managing:**
  - Size_t getMapWidth(), getMapHeight() const:
    Return respectively the width and the height of the map use for the display.
  - Char getMapAt(int x, int y) const:
    Return the character at position (x, y). Negative value returns the character since the end. May throw an out_of_range exception.
  - Void setMapAt(int x, int y, char c):
    Set the character at position (x, y) with c. Negative value sets the character since the end. May throw an out_of_range exception.
  - Void cleanMap():
    Clean the map, filling it with empty characters.

How To Create a New Graphical Library

Any graphical library must be derived from the ADisplayModule class.

Your class constructor must take a single parameter, a reference to an ArcadeComponent, which must be pass to the constructor of the ADisplayModule.

Your class **must** implement the following methods:

- Void setTexture(const std::string &filename)
  Use to set the texture to use while displaying. Takes as parameters the path of the file to use as Texture Describer. More details on texture describer files can be found on Texture Describer section.
- Void updateEvent()
  This function should manage the event of the graphical module and call the corresponding ADisplayModule's function. It is call one per frame.
- Void display()
  The main function of display. It is call one per frame

For more information about the events and the display, refer to the ADisplayModule section.

What is an ADisplayModule?

An ADisplayModule is an abstract class use by Graphical classes. It provides some functions for events managing and access to the games map.

It is constructed with a reference to an ArcadeContent.

It possesses the following implemented functions:

- **Events managing**:
  - Bool isKeyPressed(), isKeyReleased() const:
    Return True if a key is respectively pressed or released, false otherwise.
  - Bool isKeyUp(), isKeyRight(), isKeyDown(), isKeyLeft(), isKeyEnter(), isKeyBack() const:
    Return True if the corresponding key is triggered, false otherwise. **These functions are only here for retro compatibility and should never be used.**
  - Void moveUp, moveLeft, moveDown, moveRight, validate, cancel, backToMenu, restart, turnOff, nextGame, prevGame, nextGraph, prevGraph(bool isReleased):
    Managing functions that set the internals input to the corresponding type. Set the kind of input (pressed, released) depending on the parameters. All functions have a single Boolean as parameter. Default values to pressed.
  - Void releasePressedKey():
    Released the pressed key.

- Void noEvent()
  Must be used if none of the other functions are called this turn. **This function is only here for retro compatibility and should never be used.**

- **Map managing**:
  - Size_t getMapWidth(), getMapHeight() const:
    Return the width and the height of the map use for the display.
  - Char getMapAt(int x, int y) const:
    Return the character at position (x, y). Negative value returns the character since the end. May throw an out_of_range exception.

Texture Describer

A Texture Describer is a file that describe how graphical library should understand and translate the characters given by the game. **It must contain all the characters used by the game**. Each character must follow the upcoming format:

X Texture_file x y width height anim YY fg bg

X the character given by the game

Texture_file the path to the texture file, relative to the location of the Texture Describer.

x y width height the bounds of the texture to use relative to the texture file.

anim the number of animations of a sprite.

YY the character to use on a non-graphical library (e.g. nCurses)

fg bg the text color and the background color to use on a non-graphical library

If a sprite use multiples animation, all parts of the animation must have the same size and must be placed side by side. The first one must be the one describe and must be the most left one.

When creating a graphical library, it is possible to use a ParserArcadeFileTexture class.

ParserArcadeFileTexture

A class used to parse a Texture Describer file. It takes the path of the Texture Describer as a constructor. It is composed of the following members functions

- vector<char> getSymbols():
  Return all the characters correctly described in the files as a vector of char.
- **For graphicals library:**
  - const string getFilepath(char symbol) const:
    Return the path of the texture file, relative to the game folder, linked to the symbol passed as parameters
  - const array<size_t, 5> getTextureRect(char symbol):
    Return an array of 5 size_t corresponding to the coordinates of the texture and the number of animations. Coordinates corresponds to: X offset, Y offset, width, height and the number of animations.
- **For non-graphicals library:**
  - const array<char, 2> getChar(char symbol):
    Return an array of two char corresponding to char to use during display, relative to the symbol passed as parameters
  - const array<size_t, 2> getColor(char symbol):
    Return an array of two size_t corresponding respectively to the text color and the background color to use.