Documentation

Arcade

By:
Adrien Fabre
Laurent Delteil
Arthur Teisseire

1 How to play

Keys:

- F1 - Previous game
- F2 - Next game
- F3 - Previous graphics library
- F4 - Next graphics library
- R - Reload game
- ESCAPE - Go back to menu
- SUPPR - Exit the program
- ENTER - Select
- ARROWS (UP, DOWN, LEFT, RIGHT) - Move

2 Add a game

- Create a directory in games/ named after the game you want to add
- Compile your game using a Makefile to create a .so library file (using -shared) in games/ named following this format:
  lib_arcade_*$GAME_NAME$.so* ($GAME_NAME$ being the name of your game)
- You must implement an entry point function for your game with the following signature:
  *arc::IGame *gameEntryPoint();*
- You must implement a class that inherits from the interface *arc::IGame*

2.1 The Interface IGame

Here is the interface -which syntax has been simplified- you must inherit from, which is located in games/IGame.hpp:

```
1 class IGame {
2   bool isRunning() const;
3   void update(const map<Key, KeyState> &keys, float deltaTime,
4               const pair<unsigned int, unsigned int> &windowSize);
5   vector<reference_wrapper<const IComponent>> getComponents() const;
6   int getScore() const;
7 };
```

2.1.1 isRunning

The method isRunning takes no parameter but must return true or false, regarding whether or not the game is running. If this method returns false, the menu will be displayed. However if it returns true, the game will be displayed.

2.1.2 update

The method update returns nothing but takes 2 parameters:

- keys:
  Keys is a map containing a Key as map key, and a KeyState as map value.
  For more details on these two classes please refer to part 4.1 of this document
  Use this parameter to know what keys the user pressed, held, or released. Therefore this allows your game to be interactive
- windowSize:
  As you will see below, your game must return sprites to draw. Therefore it must choose the size of the sprites to create.
  This parameter allows you to know the current size of the window, and to adapt the sprites' size accordingly.

2.1.3 getComponents

The method getComponents takes no parameter but must return a vector containing references to the components of the game. That is to say the sprites to draw, the

text to write, and the sounds to play.
For more details on these classes, please refer to part 4.2 of this document.
All the components returned by this method will be processed by the graphics library methods (c.f. part 3).

2.1.4 getScore

The method getScore takes no parameter but must return the score of the game. This function is called when the game is over, to keep a record of the highest scores.

3 Add a graphics library

- Create a directory in lib/ named after the graphics library you want to add
- Compile your graphics library using a Makefile to create a .so library file (using -shared) in lib/ named following this format:
  lib_arcade_*LIB_NAME.so* (LIB_NAME being the name of your graphics library)
- You must implement an entry point function for your graphics library with the following signature:
  *arc::IGraphic *graphicEntryPoint();*
- You must implement a class that inherits from the interface *arc::IGraphic*

3.1 The Interface IGraphic

```
1  class IGraphic {
2      bool processSprite(const ISprite &sprite);
3      bool processAudio(const IAudio &audio);
4      bool processText(const IText &text;
5      void processEvents();
6      const std::map<Key, KeyState> &getKeys() const;
7      void draw();
8      bool isOpen() const;
9      std::pair<unsigned, unsigned> getWindowSize() const;
10 };
```

3.1.1 processSprite

The method processSprite takes an ISprite as parameter and returns a bool.
You must implement and use this method in order to add the sprite received as parameter to the buffer of sprites to be drawn.
In order to optimize your library, you may want to load textures only once.
The return value corresponds to whether or not the sprite was successfully prepared.
For more details on this class, please refer to part 4.2 of this document.

3.1.2 processAudio

The method processAudio takes an IAudio as parameter and returns a bool.
You must implement and use this method in order to play the audio received as parameter.
The return value corresponds to whether or not the audio was successfully played.
For more details on this class, please refer to part 4.2 of this document.

3.1.3 processText

The method processText takes an IText as parameter and returns a bool.
You must implement and use this method in order to add the text received as parameter to the buffer of texts to be written.
In order to optimize your library, you may want to load fonts only once.
The return value corresponds to whether or not the text was successfully prepared.
For more details on this class, please refer to part 4.2 of this document.

3.1.4 processEvents

The method processEvents takes no parameter and returns nothing.
You must implement and use this method in order to handle the different keys used (e.g., update, close window).

3.1.5 getKeys

The method getKeys takes no parameter and returns a map of Keys associated with their KeyState (e.g., UP with PRESSED and RIGHT with RELEASED).
For more details on these classes, please refer to part 4.1 of this document.
This map will be given to the game via the method update (c.f. part 2.1.2).

3.1.6 draw

The method draw takes no parameter and returns nothing.
You must implement and use this method in order to clear the window, and draw the sprites and the texts prepared using processSprite and processText.

3.1.7 isOpen

The method isOpen takes no parameter but returns a bool corresponding to whether or not the window is open.

3.1.8 getWindowSize

The method getWindowSize takes no parameter but returns a pair of unsigned corresponding to the window size.

4 Utility interfaces and types

4.1 Key and Keystate

Key and Keystate are two enums that you can find in components/Key.hpp

- KeyState:
  - PRESSED: The state a key is in the first time it is pressed
  - HOLD: The state a key is in the whole time the key is held down
  - RELEASED: The state a key is in when you do nothing

- Key: Currently 12 keys are handled:
  - UP
  - DOWN
  - RIGHT
  - LEFT
  - ENTER
  - ESCAPE
  - SUPPR
  - F1
  - F2
  - F3
  - F4
  - R

  To find which key does which action, please refer to part 1

4.2 IComponents

IComponent is an interface that ISprite, IText and IAudio, that are described below, inherit from.

```
1 enum ComponentType {
2       SPRITE,
3       TEXT,
4       SOUND
5 };
6 class IComponent {
7       ComponentType getType() const;
8 };
```

The only method described by this interface is getType, which returns the type of the component, namely SPRITE, TEXT or SOUND.
You want to use this class to transfer your components from the game (via the method getComponents of IGame, c.f. part 2.1) to the graphics library process methods.

4.2.1 IAudio

```
1 class IAudio : public IComponent {
2     int getVolume() const;
3     const std::string &getSoundPath() const;
4 };
```

This interface is used to implement audio objects and inherits from IComponent.

- getVolume: returns the volume of the audio
- getSoundPath: returns the file where the audio is stored, relative to the project root

4.2.2 ISprite

```
1 class ISprite : public IComponent {
2     const std::pair<float, float> &getPosition() const;
3     const std::string &getTextureName() const;
4     const std::pair<float, float> &getSize() const;
5     unsigned int getColor() const;
6 };
```

This interface is used to implement sprite objects and inherits from IComponent.

- getPosition: returns the position of the sprite, as a percent of the window size. Therefore positions must be between 0 and 1
- getTextureName: returns the location of the texture of the sprite, relative to the project root
- getSize: returns the size of the sprite to be drawn, as a percent of the window size. Therefore sizes must be between 0 and 1.
- getColor: returns a color that corresponds to the sprite in order to replace it in case the texture is missing/invalid.

4.2.3 IText

```
1 class IText : public IComponent {
2     const std::string &getText() const;
3     const std::pair<float, float> &getPosition() const;
4     int getFontSize() const;
5     const std::string &getFontPath() const;
6     unsigned int getColor() const;
7 };
```

This interface is used to implement text objects and inherits from IComponent.

- getText: returns the text to be written
- getPosition: returns the position of the text, as a percent of the window size. Therefore positions must be between 0 and 1
- getFontSize: returns the size of the font
- getFontPath: returns the path to the font, relative to the project root
- getColor: returns the color of the text.