Arcade Documentation

This is a documentation for the EPITECH project called "Arcade" done by Matthieu Veiga, Yoan Saint-Juste and Jérôme Eap in C++. Our Arcade has 2 games (Nibbler and SolarFox) and 3 graphic libraries (SFML, ncurses and libcaca) implemented. This documentation will explain how to implement new graphic libraries or game libraries compatible with our Core Program.

Game Interface

IGames.hpp :

```
#ifndef  IGAMES_HPP_
#define  IGAMES_HPP_

#include  <iostream>
#include  <vector>
#include  "Input.hpp"

class  IGames  {
    public:
        virtual  ~IGames()  =  default;

        virtual  void  run()  =  0;

        virtual  void  setEvent(std::vector<Arcade::Arcade_key>  eventArray)  =  0;

        virtual  std::vector<std::vector<int>>  getMap()  const  =  0;
        virtual  std::map<Arcade::Block_tile,  std::string>  getSprite()  const  =  0;
        virtual  std::map<Arcade::Block_tile,  Arcade::Text>  getColor()  const  =  0;
        virtual  std::vector<std::vector<int>>  getPlayer()  const  =  0;
        virtual  std::vector<std::vector<int>>  getShoot()  const  =  0;
        virtual  int  getScore()  const  =  0;
        virtual  bool  getState()  const  =  0;
};

#endif /* !IGAMES_HPP_ */
```

Must have :

If you want to implement a new game in our Core Program, your game class must fit with **all of the following statements** :

- Your class must inherit from the `IGames` interface, and must implement all the methods in which are in the interface

- Some classes and enum have been defined in a `namespace` called `Arcade`. Your implementation of this `namespace` must fit with this exemple :

```
namespace Arcade {
    // Define all mandatory binding
    enum  Arcade_key  {
        NONE = 0,
        UP, DOWN, LEFT, RIGHT,
        QUIT,
        MENU,
        USE,
        NEXT_LIB, PREVIOUS_LIB, PREVIOUS_GAME, NEXT_GAME,
        RESTART,
        PAUSE,
    };

    // Define all different block
    enum  Block_tile  {
        VOID  =  0,
        WALL,
        ITEM,
        ENEMY,
        PLAYER,
        PLAYER_HEAD,
        SHOOT
    };

    // Abstraction to fit with all Graphical Libraries
    class  Text  {
        public:
            Text(std::string  text,  std::vector<int>  fc,  std::vector<int>  bc)  :
                _text(text),  _fcolor(fc),  _bcolor(bc)  {  };
            ~Text()  =  default;

            std::string  getText()  const  {  return _text;  };
            std::vector<int>  getFColor()  const  {  return _fcolor;  };
            std::vector<int>  getBColor()  const  {  return _bcolor;  };

        private:
            std::string _text;        // Text
            std::vector<int> _fcolor; // Foreground Color
            std::vector<int> _bcolor; // Background Color
    };
}
```

Methods Explanation :

```
virtual void run() = 0;
```

This function is called each time the game is refreshed on the Core, this value is different of the time between two frames. It updates all the game, including the player, enemies, map, item, score, etc..

```
virtual  void  setEvent(std::vector<Arcade::Arcade_key>  eventArray)  =  0;
```

This function is called every frame. It takes an array of event which fits with `Arcade::Block_tile` enum in parameter and compute it.

```
virtual  std::vector<std::vector<int>>  getMap()  const  =  0;
```

This function is called every frame. It gets a `std::vector<std::vector<int>>` which is the Map of the game.

```
virtual  std::map<Arcade::Block_tile,  std::string>  getSprite()  const  =  0;
```

This function is called every frame. It gets a `std::map<Arcade::Block_tile, std::string>` which links a path to a sprite and the corresponding `Arcade::Block_tile`.

```
virtual  std::map<Arcade::Block_tile,  Arcade::Text>  getColor()  const  =  0;
```

This function is called every frame. It gets a `std::map<Arcade::Block_tile, Arcade::Text>` which links a `Arcade::Text` and the corresponding `Arcade::Block_tile`.

```
virtual  std::vector<std::vector<int>>  getPlayer()  const  =  0;
```

This function is called every frame. It gets a `std::vector<std::vector<int>>` which contains all the positions of the player. Indeed, if the player would have more than one position (like a Snake for exemple), we have to get all of his position.

```
virtual  std::vector<std::vector<int>>  getShoot()  const  =  0;
```

This function is called every frame. It gets a `std::vector<std::vector<int>>` which is a Map, such as the one got by `getMap()` method, of all projectiles.

```
virtual  int  getScore()  const  =  0;
```

This function is called every frame. It gets the score.

```
virtual  bool  getState()  const  =  0;
```

This function is called every frame. It returns a Boolean. `true` if the game is actually running, `false` if not

(when you're dead for example).

Graphical Interface

IGraphics.hpp :

```
#ifndef  IGRAPHICS_HPP_
#define  IGRAPHICS_HPP_

#include  <iostream>
#include  <vector>
#include  <map>
#include  "Input.hpp"

class  IGraphics  {
    public:
        virtual  ~IGraphics()  =  default;

        virtual  void  printMenu(std::vector<std::string>  libGraph,  std::vector<std::s
        virtual  void  printLose(int  score,  std::string  name)  =  0;
        virtual  void  printMap(std::vector<std::vector<int>>  map)  =  0;
        virtual  void  printShoot(std::vector<std::vector<int>>)  =  0;
        virtual  void  printUI(int  score,  std::string  name)  =  0;
        virtual  void  printPlayer(std::vector<std::vector<int>>)  =  0;

        virtual  void  clearScreen()  =  0;
        virtual  void  refreshScreen()  =  0;

        virtual  void  setSprite(std::map<Arcade::Block_tile,  std::string>)  =  0;
        virtual  void  setColor(std::map<Arcade::Block_tile,  Arcade::Text>)  =  0;

        virtual  std::vector<Arcade::Arcade_key>  getEvent()  const  =  0;
};

#endif /* !IGRAPHICS_HPP_ */
```

Must have :

If you want to implement a new graphical library in our Core Program, your class must fit with **all of the following statements** :

- Your class must inherit from the `IGraphics` interface, and must implement all the methods in which are in the interface
- Some classes and enum have been defined in a `namespace` called `Arcade`. Your implementation of

this `namespace` must fit with this exemple :

```
namespace Arcade {
    // Define all mandatory binding
    enum  Arcade_key  {
        NONE = 0,
        UP, DOWN, LEFT, RIGHT,
        QUIT,
        MENU,
        USE,
        NEXT_LIB, PREVIOUS_LIB, PREVIOUS_GAME, NEXT_GAME,
        RESTART,
        PAUSE,
    };

    // Define all different block
    enum  Block_tile  {
        VOID  =  0,
        WALL,
        ITEM,
        ENEMY,
        PLAYER,
        PLAYER_HEAD,
        SHOOT
    };

    // Abstraction to fit with all Graphical Libraries
    class  Text  {
        public:
            Text(std::string  text,  std::vector<int>  fc,  std::vector<int>  bc)  :
                _text(text),  _fcolor(fc),  _bcolor(bc)  {  };
            ~Text()  =  default;

            std::string  getText()  const  {  return _text;  };
            std::vector<int>  getFColor()  const  {  return _fcolor;  };
            std::vector<int>  getBColor()  const  {  return _bcolor;  };

        private:
            std::string _text;        // Text
            std::vector<int> _fcolor; // Foreground Color
            std::vector<int> _bcolor; // Background Color
    };
}
```

Methods Explanation :

```
virtual void printMenu(std::vector<std::string> libGraph, std::vector<std::string>
```

this function is called every frame. It prints a menu that has to display all available game libraries and graphical libraries. It only displays, and does not manage event !

```
virtual void printLose(int score, std::string name) = 0;
```

This function is called every frame. It displays a transition menu when the player dies. It only displays, and does not manage event !

```
virtual void printMap(std::vector<std::vector<int>> map) = 0;
```

This function is called every frame. It prints the Map got by `getMap()`. The function choose itself where and how to print the map.

```
virtual void printShoot(std::vector<std::vector<int>>) = 0;
```

This function is called every frame. It prints the Map got by `getShoot()`. The function choose itself where and how to print the map.

```
virtual void printUI(int score, std::string name) = 0;
```

This function is called every frame. It prints the score and the name of the player.

```
virtual void printPlayer(std::vector<std::vector<int>>) = 0;
```

This function is called every frame. It displays the player. The function choose itself where and how to print the player.

```
virtual void clearScreen() = 0;
```

This function is called every frame. This is the first function to be called each frame. It clears the screen.

```
virtual void refreshScreen() = 0;
```

This function is called every frame. This is the last function to be called each frame. It refresh the screen.

```
virtual void setSprite(std::map<Arcade::Block_tile, std::string>) = 0;
```

This function is called when loading a library. It sets the sprite given by `getSprite()` method

```
virtual void setColor(std::map<Arcade::Block_tile, Arcade::Text>) = 0;
```

This function is called when loading a library. It sets the `Arcade::Text` given by `getColor()` method

```
virtual  std::vector<Arcade::Arcade_key>  getEvent()  const  =  0;
```

This function is called every frame. It returns an array of all event which was catched between two frames.