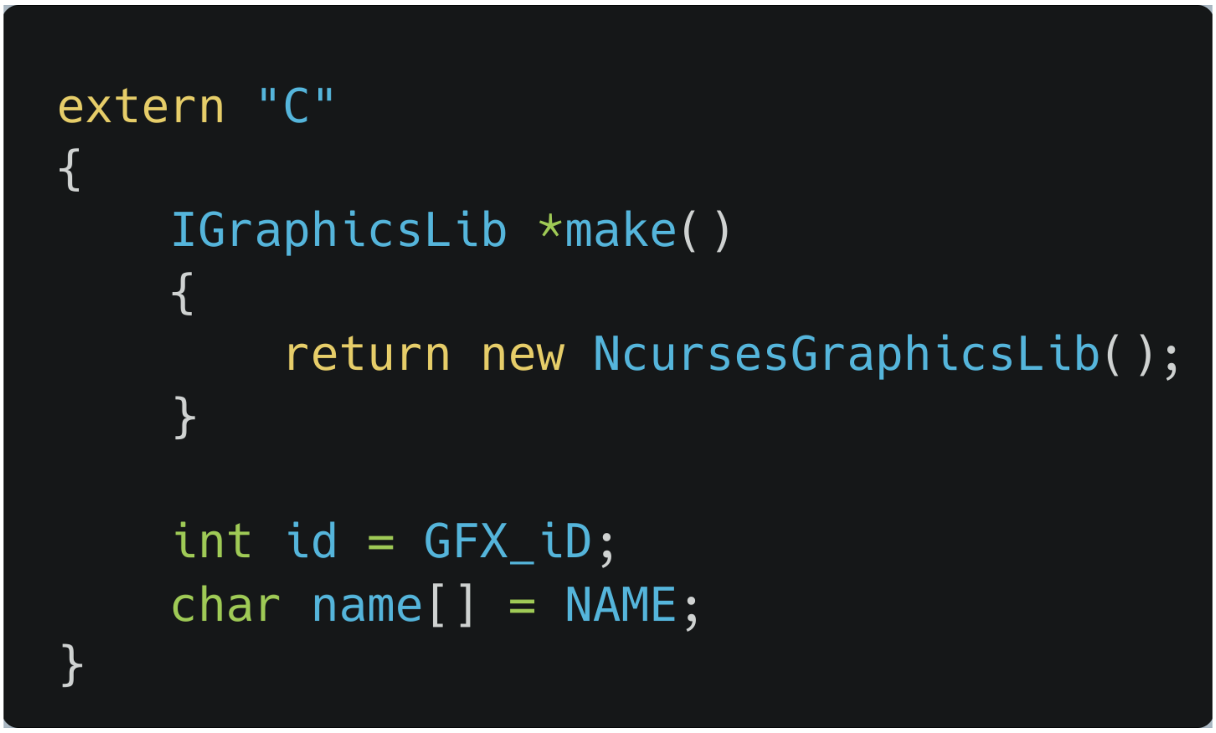
User Manual for the virtual Arcade Machine

By Léon, Mickael and Nicolas

Table of Contents

General Informations

Every library, be it a new game or a graphical library needs one very important file and a couple of values to be recognised by the arcade machine:



```
extern "C"
{
    IGraphicsLib *make()
    {
        return new NcursesGraphicsLib();
    }

    int id = GFX_iD;
    char name[] = NAME;
}
```

The **extern "C"** notation is essential, since C++ is mangling symbol names to support features like parametric polymorphism. All symbol names inside of **extern "C"** will not be mangled and thus we can access them easily to retrieve their values or build a new instance of your game or graphical library.

id can contain two possible values, 48 for games and 93 for graphical libraries. If your binary does not contain this symbol it will be ignored altogether and if the value is not correct it will not be displayed as a game or usable as a graphical engine.

name contains the name of your program, it is much more important if you develop a game since it will be what is displayed on screen.

Developing a Game

```
class IGameLib {
    public:
        virtual ~IGameLib() = default;

        virtual int frame() = 0;

        virtual void setGfx(IGraphicsLib **_gfx) = 0;

        virtual gfx_config_t getConfig() = 0;
    protected:
       IGraphicsLib **_gfx = nullptr;
};
```

Developing a game that is compatible with the arcade machine is rather simple. The arcade machine will access your game only via this interface.

Int frame();

The *frame* function will be called once during each global loop iteration. You will call your entire game logic inside of this function. It has 2 return values, 0 if everything is alright or anything else if you wish to quit the game and return to the arcade game selection menu.

void setGfx(IGraphicsLib **_gfx);

The *setGfx* function is called after your game has been launched. It is a pointer to the current Graphics Library, store it safely since you will need it if you wish to display anything to the screen. Feel free to include this little macro for convenience: `#define GFX (*this->_gfx)`. What this interface has in store for you will be described in the corresponding section.

gfx_config_t getConfig();

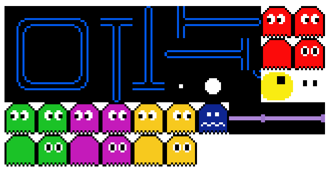
```
typedef struct {
    std::string asciiTilesetPath;
    std::string graphicalTilesetPath;
    std::string fontFolderPath;
    std::string mapPath;
    int tileWidth; // In px
    int tileHeight; // In px
    int windowWidth; // In tiles
    int windowHeight; // In tiles
    int scale;
} gfx_config_t;
```

asciiTilesetPath contains a path leading to a csv file with the following data structure: Character; Colour,Character; Colour,Character; Colour

```
/;BLUE,-;BLUE,\;BLUE,-;BLUE,|;BLUE,|;BLUE,-;BLUE,-;BLUE,M;RED,M;RED
|;BLUE, ,|;BLUE,|;BLUE,|;BLUE,-;BLUE,-;BLUE,|;BLUE,M;RED,M;RED
\;BLUE,-;BLUE,/;BLUE,|;BLUE,-;BLUE,.;WHITE,o;WHITE, ,O;YELLOW,
M;GREEN,M;GREEN,M;MAGENTA,M;MAGENTA,M;YELLOW,M;YELLOW,M;BLUE, , ,
M;GREEN,M;GREEN,M;MAGENTA,M;MAGENTA,M;YELLOW,M;YELLOW, , , ,
```

The currently supported colours are: BLACK, RED, GREEN, YELLOW, BLUE, MAGENTA, CYAN, WHITE. By developing your own terminal based graphics engines you can add more, as existing ones always have a default value.

graphicalTilesetPath contains a path leading to a .png file containing tiles of the dimensions described in **tileWidth** and **tileHeight**. Look at this example:

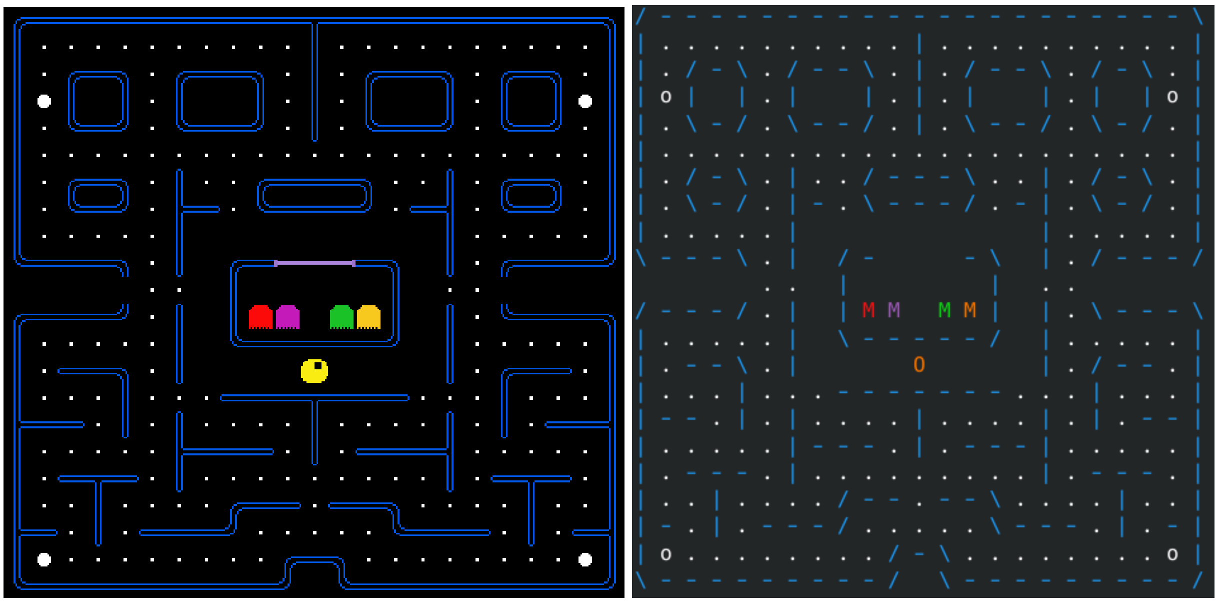


fontFolderPath contains the path to a valid .ttf file.

mapPath contains the path to a csv file that contains numbers corresponding to the position of each tile in the tileset files, the index starts at 0.
The following csv:

```
0,1,1,1,1,1,1,1,1,1,1,3,1,1,1,1,1,1,1,1,1,1,2
10,25,25,25,25,25,25,25,25,25,25,13,25,25,25,25,25,25,25,25,25,25,10
10,25,0,1,2,25,0,1,1,2,25,13,25,0,1,1,2,25,0,1,2,25,10
10,26,10,11,12,25,10,11,11,12,25,13,25,10,11,11,12,25,10,11,12,26,10
10,25,20,21,22,25,20,21,21,22,25,23,25,20,21,21,22,25,20,21,22,25,12
10,25,25,25,25,25,25,25,25,25,25,25,25,25,25,25,25,25,25,25,25,25,10
10,25,0,1,2,25,4,25,25,0,16,16,16,2,25,25,4,25,0,21,2,25,10
10,25,20,21,22,25,5,7,25,20,21,21,21,22,25,15,17,25,20,21,22,25,10
10,25,25,25,25,25,13,11,11,11,11,11,11,11,11,11,13,25,25,25,25,25,10
20,21,21,21,2,25,23,11,0,1,38,37,39,1,2,11,23,25,0,21,21,21,22
11,11,11,11,11,25,25,11,10,11,11,11,11,11,12,11,25,25,11,11,11,11,11
0,21,21,21,22,25,4,11,10,11,11,11,11,11,12,11,4,25,20,21,21,21,2
10,25,25,25,25,25,12,11,20,21,21,21,21,21,22,11,13,25,25,25,25,25,10
10,25,15,16,2,25,23,11,11,11,11,11,11,11,11,11,23,25,0,1,7,25,10
10,25,25,25,13,25,25,25,15,21,21,3,1,21,7,25,25,25,10,25,25,25,10
5,6,7,25,23,25,4,25,25,25,25,13,25,25,25,25,4,25,23,25,15,16,17
10,25,25,25,25,25,5,21,1,7,25,23,25,15,21,21,17,25,25,25,25,25,10
10,25,15,3,7,25,23,25,25,25,25,25,25,25,25,25,23,25,15,3,7,25,10
10,25,25,13,25,25,25,25,0,16,7,25,15,16,2,25,25,25,25,13,25,25,10
5,7,25,23,25,15,16,16,22,25,25,25,25,25,20,6,21,7,25,23,25,15,17
10,26,25,25,25,25,25,25,25,25,0,1,2,25,25,25,25,25,25,25,25,26,10
20,21,21,21,21,21,21,21,21,21,22,11,20,21,21,21,21,21,21,21,21,21,22
```

Results to this (without the pacman and ghosts) in graphical libraries / terminal:



```
/ - - - - - - - - - - - - - - - - - - - - - \
| . . . . . . . . . . | . . . . . . . . . . |
| . / - \ . / - - \ . | . / - - \ . / - \ . |
| o |   | . |     | . | . |     | . |   | o |
| . \ - / . \ - - / . | . \ - - / . \ - / . |
| . . . . . . . . . . . . . . . . . . . . . |
| . / - \ . | . . / - - - \ . . | . / - \ . |
| . \ - / . | - . \ - - - / . - | . \ - / . |
| . . . . . |                   | . . . . . |
\ - - - \ . |   / -       - \   | . / - - - /
          . .   |           |   . .
/ - - - / . |   | M M   M M |   | . \ - - - \
| . . . . . |   \ - - - - - /   | . . . . . |
| . - - \ . |         0         | . / - - . |
| . . . | . . . - - - - - - - . . . | . . . |
| - - . | . | . . . . | . . . . | . | . - - |
| . . . . . | - - - . | . - - - | . . . . . |
| . - - - . | . . . . . . . . . | . - - - . |
| . . | . . . . / - - . - - \ . . . . | . . |
| - . | . - - - / . . . . . \ - - - . | . - |
| o . . . . . . . . / - \ . . . . . . . . o |
\ - - - - - - - - - /   \ - - - - - - - - - /
```

windowWidth and **windowHeight** are the overall window dimensions in tiles, So a 20x20 window will mean 20x20 tiles, meaning the final window size will depend on your tile size.

scale is a multiplier for all dimensions, it is recommended to keep it at 1.

Developing a Graphical Library

Developing a graphical library is a bit stricter as you will have to supply functions that will also be used by the Arcade machines main program

```cpp
class IGraphicsLib
{
    public:
        virtual ~IGraphicsLib() = default;

        // Updates screen with buffer (called at the end of all draw tiles)
        virtual void display() = 0;

        // Clear screen/window
        virtual void flush() = 0;

        // Draw tile of index tile_index, at x tile and y tile
        // (tile is the unit)
        // Note : Could add possibility for rotation of tile in the future
        virtual void drawTile(int tile_index, int x, int y, int orientation = ORIENT_TOP) = 0;

        // Draws text, at x tile and y tile
        virtual void drawText(const std::string &text, int x, int y, rgb_t color = {255,255,255}) = 0;

        /**
         * Saves all inputs to process to a queue
         * Call in Core at the beginning of everything
         */
        virtual void recordInputs() = 0;

        /**
         * Returns all inputs recorded
         * @return std::queue<char>
         */
        virtual std::queue<char> &getInput() = 0;

        virtual void popInput() = 0;

        /**
         * Checks if game config is currently loaded config
         * If not : loads config
         * @param config
         */
        virtual void checkConfig(const gfx_config_t &config) = 0;

        virtual std::string getName() = 0;
};
```

void display();

In this function you will call