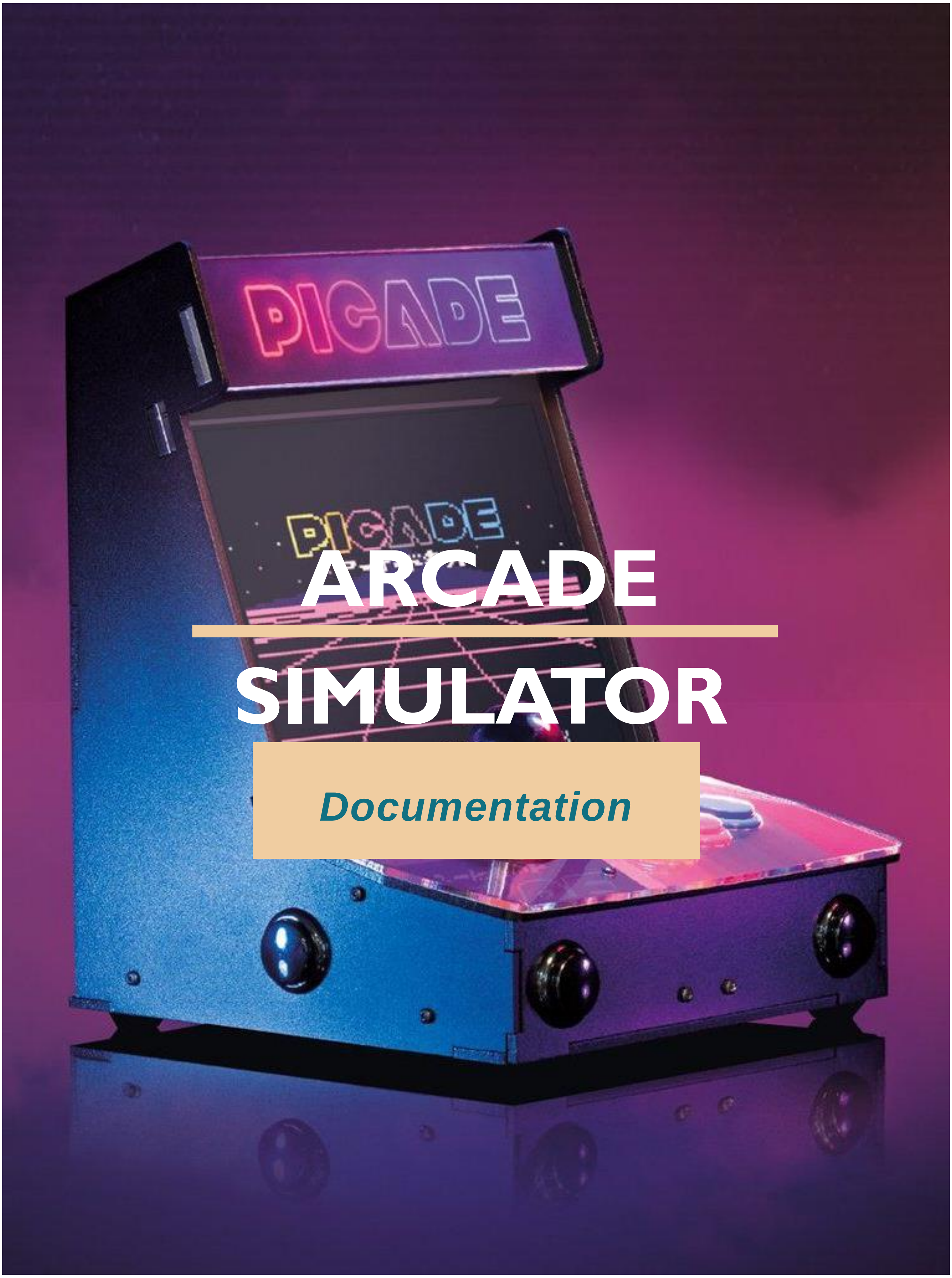

ARCADE SIMULATOR

Documentation

TABLE DES MATIÈRES

INTRODUCTION

The project is a simulator of an arcade. The goal is to be able to load and use dynamicaly both a graphic librarie and a game. Each of those librarie must follow a very presice interface and need to respect a given impletation for some methods. The graphic librarie must implement arcade::displayer::IDisplay interface. The game librarie must implement arcade::games::IGame interface. Both librarie type must have an extern "C" entry point function, which takes no param and return a unique_ptr of the given interface. The name of the fuction can be found under DISPLAYER_ENTRY_POINT and GAMES_ENTRY_POINT.

1. IDISPLAY INTERFACE

```
namespace arcade
{
    namespace displayer
    {
        class IDisplay {
            public:
                enum AvailableOptions {
                    NO_OPTIONS          = 0,
                    SET_CHARACTER_SIZE  = 1 << 0,
                    MOUSE_MOVE          = 1 << 1,
                    SETTING_FONTS       = 1 << 2,
                };
                #define isOptions(disp, opt)    ((disp)-
>availableOptions() & arcade::displayer::IDisplay::AvailableOptions::opt)

                virtual int availableOptions() const = 0;
                virtual void init(const std::string &winName, unsigned int framesLim
it = 60) = 0;
                virtual void stop() = 0;
                virtual bool isOpen() = 0;
                virtual void clearWindow() = 0;
                virtual void display() = 0;
                virtual void restartClock() = 0;
                virtual double getDeltaTime() = 0;
                virtual arcade::data::Vector2u getWindowSize() = 0;
                virtual std::vector<arcade::data::Event> getEvents() = 0;
                virtual void draw(std::unique_ptr<IText> &text) = 0;
                virtual void draw(std::unique_ptr<ISprite> &sprite) = 0;
                virtual std::unique_ptr<IText> createText() = 0;
                virtual std::unique_ptr<IText> createText(const std::string &text) =
 0;
                virtual std::unique_ptr<ISprite> createSprite() = 0;
                virtual std::unique_ptr<ISprite> createSprite(const std::string &spr
itePath, const std::vector<std::string> &asciiSprite, arcade::data::Vector2f scale =
 arcade::data::Vector2f{1, 1}) = 0;
                virtual double scaleMoveX(double time) = 0;
                virtual double scaleMoveY(double time) = 0;
        };
    } // namespace Displayer
} // namespace arcade
```

```cpp
virtual int availableOptions() const = 0;
```

Must return a combination of the `enum AvailableOptions`. Giving information on what is supported by the librarie.

```cpp
virtual void init(const std::string &winName, unsigned int framesLimit = 60) = 0;
```

This is allways called in first, the librarie can set here any necessary ressources and must initialize the window at this point.

```cpp
virtual void stop() = 0;
```

This is allways called before destroying the object. The librarie must close the window, and release all ressources.

```cpp
virtual bool isOpen() = 0;
```

Is the window still open.

```cpp
virtual void clearWindow() = 0;
```

Clear the buffer to be displayed on the window.

```cpp
virtual void display() = 0;
```

Display the buffer onto the window. It must also stop the process to respect as much as possible the frame limit that was given at initialisation. It must also restart the internal clock used to calculate a frame duration.

```cpp
virtual void restartClock() = 0;
```

Force to restart the internal clock used to calculate a frame duration. It is only used at the start of the program, after the init and just before launching the loop.

```
virtual double getDeltaTime() = 0;
```

Returns the duration of the previous frame duration. This meens that the returned value does not change until the display method is called. It must use the same clock as the restartClock method and the one being restarted in the display method.

```
virtual arcade::data::Vector2u getWindowSize() = 0;
```

Return a vector with the width and height of the actual window. The width is stored in the x property and the height in the y.

```
virtual std::vector<arcade::data::Event> getEvents() = 0;
```

Return a vector of normalise events. This method will be called multiple times per frame, it must then return always the same data until a call to display is made.

```
virtual void draw(std::unique_ptr<IText> &text) = 0;
virtual void draw(std::unique_ptr<ISprite> &sprite) = 0;
```

Draw into the buffer the given object. It will be display onto the screen on the next call to the display method.

```
virtual std::unique_ptr<IText> createText() = 0;
virtual std::unique_ptr<IText> createText(const std::string &text) = 0;
```

Those methods are the only way to build an IText object.

```
virtual std::unique_ptr<ISprite> createSprite() = 0;
virtual std::unique_ptr<ISprite> createSprite(const std::string &spritePath, const s
td::vector<std::string> &asciiSprite, arcade::data::Vector2f scale = arcade::data::V
ector2f{1, 1}) = 0;
```

Those methods are the only way to build an ISprite object. A graphic librarie can be either a terminal based one, such as ncurses, or a gui such as sfml. The param spritePath is used to load a sprite from a file in a gui lib. The asciiSprite is used by the terminal based lib.

```
virtual double scaleMoveX(double time) = 0;
virtual double scaleMoveY(double time) = 0;
```

Those methods gives a simple way to move an object at a consistent speed. Time, gives the wiched time to pass from one side of the screen to the other one. The traveled time does not depend on the width or the height of the window, nor the number of frames per second. The X variation depend on the width of the screen, where the Y variation depend on the height.

A possible implementation is :

```
double LibSFML::scaleMoveX(double time)
{
    if (!time) {
        return 0;
    }
    return (getWindowSize().x / time) / (1.0f / getDeltaTime());
}

double LibSFML::scaleMoveY(double time)
{
    if (!time) {
        return 0;
    }
    return (getWindowSize().y / time) / (1.0f / getDeltaTime());
}
```

2. ITEXT & ISPRITE

```
namespace arcade
{
    namespace displayer
    {
        class IText {
            public:
                virtual void setText(const std::string &text) = 0;
                virtual std::string getText() const = 0;
                virtual void setPosition(arcade::data::Vector2f pos) = 0;
                virtual arcade::data::Vector2f getPosition() const = 0;
                virtual void setFont(const std::string &font) = 0;
                virtual void setColor(arcade::data::Color color) = 0;
                virtual arcade::data::Color getColor() const = 0;
                virtual void setCharacterSize(unsigned int size) = 0;
                virtual arcade::data::FloatRect getLocalBounds() = 0;
                virtual arcade::data::FloatRect getGlobalBounds() = 0;
                virtual void setOrigin(arcade::data::Vector2f origin) = 0;
                virtual arcade::data::Vector2f getOrigin() = 0;
        };
        class ISprite {
            public:
                virtual void setSprite(const std::string &spritePath, const std::
vector<std::string> &asciiSprite) = 0;
                virtual void setPosition(arcade::data::Vector2f pos) = 0;
                virtual arcade::data::Vector2f getPosition() const = 0;
                virtual void move(arcade::data::Vector2f pos) = 0;
                virtual void move(float x, float y) = 0;
                virtual void setOrigin(arcade::data::Vector2f origin) = 0;
                virtual arcade::data::Vector2f getOrigin() = 0;
                virtual arcade::data::FloatRect getLocalBounds() = 0;
                virtual arcade::data::FloatRect getGlobalBounds() = 0;
                virtual void setScale(arcade::data::Vector2f scale) = 0;
                virtual arcade::data::Vector2f getScale() = 0;
                virtual void rotate(float anle) = 0;
                virtual void setRotation(float angle) = 0;
                virtual float getRotation() = 0;
                virtual void setTextureRect(const arcade::data::IntRect &rect) =
0;
                virtual arcade::data::IntRect getTextureRect() const = 0;
                virtual void setColor(arcade::data::Color color, const std::vecto
r<std::vector<arcade::data::Color>> &asciiColors) = 0;
        };
    } // namespace Displayer
} // namespace arcade
```

a. Common methods

```
virtual void setPosition(arcade::data::Vector2f pos) = 0;
virtual arcade::data::Vector2f getPosition() const = 0;
```

The position of the object. This position do not take the origin into consideration in contrary of the global bounds. Do not use this for any collision.

```
virtual arcade::data::FloatRect getLocalBounds() = 0;
virtual arcade::data::FloatRect getGlobalBounds() = 0;
```

getLocalBounds describe the object in his original state before any scale. The left and top will then be always set to 0.

getGlobalBounds describe the object as he is displayed. The left and top property correspond to the object position representing the top left coner of the object. The width and height correspond to the real dimension of the object. This is the property to be used to detect any collision.

```
virtual void setOrigin(arcade::data::Vector2f origin) = 0;
virtual arcade::data::Vector2f getOrigin() = 0;
```

By default, the position represent the top left corner of the object. The origin allow to change this relation. The origin depends on the original sprit, but will be changed with the sprite when scaling. If the position is set to {x=2, y=1} and the origin {x=1, y= 0} the object will be drawn from {x=1, y=1}. To set the origin at the center:

auto localBounds = myObject->getLocalBounds();

myObject->setOrigin(arcade::data::Vector2f{localBounds.width / 2, localBounds.height / 2});

b. IText

```
virtual void setText(const std::string &text) = 0;
virtual std::string getText() const = 0;
```

Set and get the string to be displayed.

```
virtual void setFont(const std::string &font) = 0;
```

Load a font found at the given path by the font param. Some libs cannot set any font, the method then will have no effect.

```
virtual void setCharacterSize(unsigned int size) = 0;
```

Set the size of a character. Some libs cannot set the size of a char, the method then will have no effect.

c. ISprite

```
virtual void setSprite(const std::string &spritePath, const std::vector<std::string
&asciiSprite) = 0;
```

Can change the sprite that was previously loaded. The param spritePath is used to load a sprite from a file in a gui lib. The asciiSprite is used by the terminal based lib.

```
virtual void move(arcade::data::Vector2f pos) = 0;
virtual void move(float x, float y) = 0;
```

Move the position of the sprite.

mySprite->move(pos); is equivalent to: mySprite->setPosition(mySprite->getPosition() + pos);

```cpp
virtual void setScale(arcade::data::Vector2f scale) = 0;
virtual arcade::data::Vector2f getScale() = 0;
```

Set and get the scale of the sprite. It scales the image, changing its size. The scaled size can be found with getGlobalBound().

```cpp
virtual void rotate(float anle) = 0;
virtual void setRotation(float angle) = 0;
virtual float getRotation() = 0;
```

Set and get the rotation of the sprite. setRotation override the angle, where rotate change it relative to it actual rotation:

mySprite->rotate(90); is equivalent to mySprite->setRotation(mySprite->getRotation() + 90);

```cpp
virtual void setTextureRect(const arcade::data::IntRect &rect) = 0;
virtual arcade::data::IntRect getTextureRect() const = 0;
```

By default, a sprite is fully drawn, setting a different texture rectangle allow to partially draw the sprite. Top and left are relative to the top left corner.

```cpp
virtual void setColor(arcade::data::Color color,
          const std::vector<std::vector<arcade::data::Color>> &asciiColors) = 0;
```

The param color is used only in a gui lib. asciiColors allow to set a different color for each char composing the sprite.

3. IGAME INTERFACE

```
namespace arcade
{    namespace games
    {
        #define GAMES_ENTRY_POINT       entry_point

        enum GameStatus {
            PLAYING,
            GAME_ENDED,
        };
        class IGame {
            public:
                virtual void init(std::shared_ptr<arcade::displayer::IDisplay> &disp
) = 0;
                virtual GameStatus update() = 0;
                virtual void stop() = 0;
                // virtual void restart() = 0;
                virtual unsigned int getScore() = 0;
        };
    } // namespace games
} // namespace arcade
```

This interface is at the time not finiched.

4. NORMALISE DATA

The exchange of data between the arcade, the libs and the libs them selfs is simplified with a set of standardize data. Each graphic librarie having their own event system will need to normilse it to fit with the arcade::data::Event type.

a. Events

```
namespace arcade
{
    namespace data
    {
        enum EventType
        {
            WINDOW_CLOSED,
            KEY_PRESSED,
            MOUSE_MOVED,
            MOUSE_PRESSED,
            MOUSE_RELEASED,
        };
        enum        KeyCode
        {
            ENTER              = 10,
            ESCAPE             = 27,
            SPACE              = 32,
            SPECIAL_KEYS_START  = 257,
            DOWN               = 258,
            UP                 = 259,
            LEFT               = 260,
            RIGHT              = 261,
            BACKSPACE          = 263,
        };
        enum MouseBtn
        {
            BTN_1,
            BTN_2,
            BTN_3,
            BTN_4,
        };
        struct Event
        {
            Event();
            Event(EventType type);
            Event(EventType type, MouseBtn btn, int x, int y);
            Event(EventType type, int x, int y);
            Event(EventType type, KeyCode keyCode);
            Event(EventType type, char key);

            EventType type;
            union
            {
                KeyCode keyCode = static_cast<KeyCode>(0);
                char key;
                MouseBtn btn;
            };
            struct
            {
                int x;
                int y;
            };
        };
    } // namespace data
} // namespace arcade
```

This data type is used by the getEvents API. It allows for the arcade or game to use the events without knowing which graphic lib is being used. Apart of the event type, each property is field or not depending on its use for the type.

b. Vectors

```cpp
namespace arcade
{
    namespace data
    {
        template<typename T>
        struct Vector2
        {
            Vector2()
            Vector2(T x)
            Vector2(T x, T y)
            template<typename U>
            Vector2(const Vector2<U> &vect);

            template<typename U>
            Vector2<T> &operator=(const Vector2<U> &other);

            template<typename U>
            Vector2<T> &operator+=(const Vector2<U> &other);

            Vector2<T> operator+(const Vector2<T> &other) const;

            template<typename U>
            bool operator==(const Vector2<U> &other) const;

            Vector2<T> &move(T x);
            Vector2<T> &move(T x, T y);
            template<typename U>
            Vector2<T> &move(const Vector2<U> &other);

            T x;
            T y;
        };

        typedef Vector2<int>          Vector2i;
        typedef Vector2<unsigned int> Vector2u;
        typedef Vector2<float>        Vector2f;
    } // namespace data
} // namespace arcade
```

c. Rectangles

```
namespace arcade
{
    namespace data
    {
        template<typename T>
        struct Rect
        {
            Rect();
            Rect(T width, T height);
            Rect(T top, T left, T width, T height);
            template<typename U>
            explicit Rect(U top, U left, U width, U height);
            template<typename U>
            Rect(const Rect<U> &rect);

            T top;
            T left;
            T width;
            T height;
        };

        typedef Rect<int>       IntRect;
        typedef Rect<float>     FloatRect;
    } // namespace data
} // namespace arcade
```

d. Colors

```
namespace arcade
{
    namespace data
    {
        struct Color
        {
            Color();
            Color(uint8_t red, uint8_t green, uint8_t blue, uint8_t alpha = 255);

            bool operator==(const Color &other) const;

            uint8_t r;
            uint8_t g;
            uint8_t b;
            uint8_t a;

            static const Color Black;
            static const Color White;
            static const Color Red;
            static const Color Green;
            static const Color Blue;
            static const Color Yellow;
            static const Color Magenta;
            static const Color Cyan;
            static const Color Transparent;
        };
    } // namespace data
} // namespace arcade
```

All the static const variable are shortcuts for the basic colors. They are the pure colors, red is {r=255, g=0, b=0, a=255}. All the colors have the alpha set to the maximum. The transparent is a white color whith an alpha set to 0.

5. ERRORS

```
namespace arcade
{
    namespace errors
    {
        class Error : public std::exception {
            public:
                Error(const std::string &msg);
                ~Error();

                const char *what() const noexcept override;

            protected:
            private:
                const std::string _msg;
        };
    } // namespace Errors
} // namespace arcade
```

This is the only type of errors that are expect to be thrown by either the graphic lib or the game lib.