Interfaces Doc

IComponent :

Enum that contain every type we will display, used to differentiate a sprite (OBJECT) than a music (SOUND) for exemple

```
enum Type {
    OBJECT,
    SOUND,
    TEXT
};
```

Component classe inherit from IComponent to define every things needed to transfer information from Games to core and finaly to Libs

```cpp
class IComponent {
    public:

        enum Type {
            OBJECT,
            SOUND,
            TEXT
        };
        virtual ~IComponent() = default;

        [[nodiscard]] virtual std::size_t getId() const = 0;

        [[nodiscard]] virtual Type getType() const = 0;

        [[nodiscard]] virtual std::string getFile() const = 0;

        [[nodiscard]] virtual int getX() const = 0;

        [[nodiscard]] virtual int getY() const = 0;

        [[nodiscard]] virtual int getWidth() const = 0;

        [[nodiscard]] virtual int getHeight() const = 0;

        [[nodiscard]] virtual Rect getRect() const = 0;

        [[nodiscard]] virtual std::string getText() const = 0;

        virtual void setX(std::size_t const x) = 0;

        virtual void setY(std::size_t const y) = 0;

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

        virtual void setRect(Rect const rect) = 0;
};
```

Basically this Interface is composed of differents basics setter used to set information from the game.so and getter to take back those information for the lib.so

IGame :

Every game inherit from IGame to create a new game

```cpp
namespace Arcade {

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

            virtual void init() = 0;

            virtual void stop() = 0;

            [[nodiscard]] virtual std::string getGameName() const = 0;

            virtual std::vector<std::unique_ptr<IComponent>> &getComponents() = 0;

            virtual void sendEvents(std::vector<std::unique_ptr<IEvent>> &events) = 0;

            virtual IEvent *getEvent() = 0;

            virtual void sendDisplayLibs(std::vector<std::string> libs) = 0;

            virtual void sendGameLibs(std::vector<std::string> libs) = 0;

            virtual void setPlayerName(std::string name) = 0;

            [[nodiscard]] virtual std::string getPlayerName() const = 0;
    };
}
```

Init > Initialise all needed for the game like sprite, map or position

Stop > Free all thing initialise in Init

GetIComponent > Return a vector of all the component

SendEvent > Send event from game to core

IDisplay :

Every lib inherit from IDisplay to create a new lib

```cpp
namespace Arcade {
    class IDisplay {
        public:
            virtual ~IDisplay() = default;

            virtual void init() = 0;

            virtual void stop() = 0;

            virtual std::string getLibName() const = 0;

            virtual void display(std::vector<std::unique_ptr<IComponent>> &components) = 0;

            virtual std::vector<std::unique_ptr<IEvent>> &getEvents() = 0;

            virtual void clear() = 0
    };
}
```

Init > Initialise all needed for the lib like open a Window or Font

Stop > Free all things initialise in Init

Display > Display all IComponent get from the core

GetEvent > Handle event from mouse and keyboard and send it to the core

Clear > Clear window and free elements

IEvent :

Event classe inherit from IEvent to handle diferents kind of event

```
namespace Arcade {
    struct Pos {
        int x;
        int y;
    };
    class IEvent {
        public:
            virtual ~IEvent() = default;

            [[nodiscard]] virtual std::size_t getKey() const = 0;

            virtual void setKey(std::size_t key) = 0;

            [[nodiscard]] virtual Pos getMousePos() const = 0;

            [[nodiscard]] virtual std::string getData() const = 0;
    };
}
```

GetKey > Get the key pressed

GetMousePos > Get actual position of the mouse

GetData > Get event Text data