GAME LIBRARIES MANAGEMENT : Interface IGame

The `IGame` interface is used as a base for all our game libraries.
It is imperative to use it if you want to implement a new game library that is compatible with our program.

Games that you can implement:

The arcade being an old machine, it is only compatible with a few games. Here is a list of compatible games :

- SolarFox: `arcade_solarfox.so`
- Pacman: `arcade_pacman.so`
- Centipede: `arcade_centipede.so`
- Nibbler: `arcade_nibbler.so`
- Qix: `arcade_qix.so`

Be careful when creating the library to use the correct name, only the bellow formats are supported. Moreover, when your dynamic library is created, you must place it in the `./lib` folder for it to be used by the program. If you don't, it won't be used by our arcade.

HOW TO : add a new game library

All the game libraries loaded in the program must be dynamic.
For our project to run properly, your game library must implement this function for its creation:

```
extern "C" std::unique_ptr<IGame> createGame();
```

It will return a pointer to the newly created class that inherits from the `IGame` interface. Here is an example of implementation for the Centipede game, that you can use as a template for your own implementation :

```
extern "C" std::unique_ptr<IGame> createGame()
{
    return (std::make_unique<Centipede>());
}
```

/!\ We are storing a `unique_ptr` of our game in the `Core` class, so please don't use `shared_ptr` or any of other type of pointer !

In order for your game library to be compatible with our program, you must firstly create your own class that inherits from the `IGame` interface. Several methods are expected inside of it:

- exec(IWindow&, Events&);
- restart();
- getStatus();
- getScore();
- getSize();

In-depth explanation of the methods:

```
void exec(IWindow& window, Events& event)
```

- Execute one tick of the game.
- This method is called in the `Core::executeLoop()` method.
- It takes a reference of our window and the event that needs to be handled in game.

```
void restart()
```

- Method called by the core to restart the game.
- Clears the player's score and progression in-game.

```
Status getStatus() const noexcept
```

- Return `Status`, an enum representing the current status of the game to the `Core`.
- Its return value is then used for event handling in the main loop of the project.
- You can find more informations about the `Status` enum in the `Utils.hpp` file.

```
int getScore() const noexcept
```

- returns the player score when he lost, won or decided to return to the menu in-game.
- Its return value is then used to update the scoreboard displayed in the menu.

```
vec2int getSize() noexcept
```

- Returns the size of the window.
- The `vec2int` structure contains a pair of int `x` and `y`.