Implement new games

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 is currently compatible with a limited amount of libraries.
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`

However, you can add your own libraries as long as they are correctly implemented. 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.

Here is an example of implementation for the Nibbler game, that you can use as a template for your own implementation :

Filename: NibblerLib.cpp

```
extern "C"
{
    std::unique_ptr<IGame> entryPoint(void)
    {
        return (std::make_unique<Nibbler>());
    }
}
```

General methods:

`std::string getSpritePath()`

- Returns a string containing the file path to the game's sprites.

`void display(IGraphic &)`

- Displays the game using the specified graphical library.

`int init()`

- Initializes the game state and returns an integer indicating whether initialization was successful.

`void reset()`

- Resets the game state.

`int updateGame(eventKey evtKey)`

- Updates the game state based on the user input provided by eventKey evtKey. The method returns an integer indicating whether the update was successful.

`elemSize getDisplaySize(void)`

- Get the size of the window currently displayed.

You can then add the methods you want to make your own game.