How to create a game

Arcade documentation

April Revision

First steps

Your game will take the form of a shared library (`.so` file).

You need to compile your game with the `-fPIC` and `-fno-gnu-unique` flags.

Your game should export a function named `expose` with the following signature:

```
extern "C" arc::game::IGame* expose(void);
```

This function will be called by the engine to create your game, and will serve as an entry point for your game.

Your game should also export a function named `unexpose` with the following signature:

```
extern "C" void unexpose(arc::game::IGame* game);
```

It will be called as needed by the engine. It serves as a free function to free your game from the memory.

Your game also needs to export a function named `getType` with the following signature:

```
extern "C" arc::DLType getType(void);
```

It will be called by the engine to determine the type of your shared library.

In our case, the type will be `arc::DLType::GAME`.

```
arc::DLType getType(void) {
    return arc::DLType::GAME;
}
```

Your game can also optionally export a function named `getName` with the following signature:

```
extern "C" const char* getName(void);
```

It will be used in the game list in order to display the name of your game. If not exported, the name displayed will be the name of the shared library file.

2 — Creating the game

In your game, you should use the `arc::game::IGame` interface to interact with the engine.

Your game is separated in 4 parts:

- The `init` method, called by the engine when the game is loaded.
- The `update` method, called by the engine every frame to handle events and update the game.
- The `render` method, called by the engine every frame to handle the rendering.
- The `destroy` method, called by the engine when the game is unloaded.

Your game must also provide the following methods:

- `setManager`: called by the engine to pass the game manager instance to the game.
- `loadGraphic`: called by the engine each time a graphic library is loaded.
- `unloadGraphic`: called by the engine each time a graphic library is unloaded.

You must also provide the `mustLoadAnotherGraphic` method, but it is not called by the engine anymore (the graphic switching is now handled by the game manager itself).

You must have an `std::shared_ptr<arc::grph::ICanvas>` member variable that will hold the current library canvas, used to render the game.

It must be handled in the `loadGraphic` and `unloadGraphic` methods to ensure that the canvas is correctly set, thanks to the `IGraphic::loadCanvas(ICanvas)` and `IGraphic::unloadCanvas()` methods.

3 — Drawing

First, you need to create a `arc::grph::Palette` instance, which will be used to store the colors of the game.

```
arc::grph::Palette palette;
```

Then, you need to set its colors.

```
void MyGame::init() {
    palette.setColor(0, 'P', RED);
    palette.setColor(1, 'A', GREEN);
}
```

The `render` method is called by the engine every frame to draw the game.

The first thing you should do is to clear the screen with the `clear` method.

```
void MyGame::render() {
    graphic->clear();
}
```

Then, you can do your drawing between the `startDraw` and `endDraw` methods of the canvas. When you are done, you can call the `render` method of the graphical library to display the drawings on the screen.

```
void MyGame::render() {
    graphic->clear();
    canvas->startDraw();

    canvas->drawText(10, 10, "Hello World!", palette[1]);
    canvas->drawPoint(10, 10, palette[0]);

    canvas->endDraw();
    graphic->render();
}
```

4 — Events

Events are expected to be handled by the game in the `arc::game::IGame::update` method, thanks to `arc::core::IManager::pollEvent` method. An `arc::Event` argument should be passed to the poll event method, which will be filled with the event data.

```
void MyGame::update() {
    arc::Event event;

    while (manager->pollEvent(event)) {

        if (event.type == Event::KEYDOWN) {
            if (event.keyboardInput.keyCode == arc::KeyCode::ESCAPE) {
                // ...
                graphic->close();
            }
            // ...
        }

    }
}
```

5 — Scoreboard

Your game will probably have a score system, and you might want to display the score of the player in the game menu scoreboard.

You can get and set the high score of the current player thanks to the `arc::game::IManager::getHighScore` and `arc::game::IManager::setHighScore` methods.

```
void MyGame::myGameOverMethod() {
    //...
    if (this->manager->getHighScore("mygameid") < score) {
        this->manager->setHighScore("mygameid", score);
    }
}
```