How to Implement a New Game for Arcade

This document provides a step-by-step guide on how to implement a new game for the Arcade platform, based on the provided interfaces.

Table of Contents

Overview

The Arcade platform uses dynamic libraries to load games at runtime. To implement a new game, you need to create a shared library that conforms to the IGameModule interface provided by the platform.

Your game library should be independent of any graphics libraries, as the rendering is handled by the display modules. Your game only needs to manage game logic and provide objects to be rendered.

Required Files

To implement a new game, you'll need to create the following files:

1. A header file for your game class
2. An implementation file for your game class
3. An entry point file with the required factory function

Implementing the Game Module

1. Create a Game Class

Your game class must inherit from Arcade::IGameModule and implement all required methods:

2. Implement the Game Logic

In your implementation file, you'll need to:

- Initialize the game state in the constructor
- Implement the game logic in the update method
- Manage the game objects

3. Create the Entry Point

You need to provide a factory function that creates an instance of your game:

```
#include "MyGame.hpp"

extern "C" std::unique_ptr<Arcade::IGameModule> createInstance()
{
  return std::make_unique<MyArcade::MyGame>();
}
```

Game Objects

Games in Arcade use objects that implement the IObject interface. These objects represent elements in your game that will be rendered by the display modules.

Object Types

The platform supports different types of objects:

- SPRITE: For graphical elements
- TEXT: For text elements

Creating and Managing Objects

1. Create objects with unique identifiers
2. Set their positions, properties, and textures
3. Store them in the _objects map
4. Update their properties in the update method based on game logic

Object Properties

For sprite objects, you can set the following properties:

```
Arcade::IObject::SpriteProperties props;
props.size = {width, height};        // Size of the sprite
props.offset = {offsetX, offsetY};    // Offset in the sprite sheet
props.textSize = {textWidth, textHeight}; // Size of text if applicable
props.textOffset = {textOffsetX, textOffsetY}; // Text offset if applicable
props.scale = {scaleX, scaleY};      // Scaling factor
props.textColor = COLOR(A, R, G, B);  // Text color if applicable
```

For text objects:

```
Arcade::IObject::TextProperties props;
props.color = COLOR(A, R, G, B);     // Text color
props.characterSize = size;          // Font size
props.text = "Your text here";       // The text to display
```

Compilation

To compile your game as a shared library:

```
g++ -std=c++17 -shared -fPIC -o arcade_mygame.so MyGame.cpp Object.cpp -I../include
```

Place the compiled library in the ./lib/ directory of the Arcade project.

Best Practices

1. **Separation of Concerns**: Keep your game logic separate from rendering concerns
2. **Efficient Updates**: Only update objects when necessary
3. **Clean Resource Management**: Properly initialize and clean up resources
4. **Error Handling**: Use appropriate error handling for robustness
5. **Game Loop**: Implement a proper game loop in the update method
6. **Input Handling**: Handle user input appropriately
7. **Object Management**: Use descriptive names for objects in your _objects map

By following this guide, you should be able to implement a new game for the Arcade platform that will work with any of the available graphics libraries.