Current multiplatform game engines are complicated for new developers to use and lack the ability to be fully customized and experimented on.

A modern multiplatform game engine needs a input handling subsystem that is latency free and easy to work with.

A game engine also needs a messaging system that allows subsystems to notify each other whenever the state of a subsystem is changed.

Requirements

Current multiplatform game engines are

A game engine also needs a messaging

A modern multiplatform game engine needs a

The above diagram shows the open layered architecture of the game engine. I worked in the engine core layer on the Input and message subsystems.

Overview

The Unreal Engine is a very popular game engine used throughout the industry. However, its very high cost typically make it an unreasonable choice for smaller studios.

The system shall provide the ability to render 2D/3D graphics, these graphics will be represented by assets like models, textures, and shaders.

The system will provide a way to initialize and control the application using a core game loop that will be maintained by the use.

The system allows support for mouse and keyboard as well as PC joysticks in an easy to use and latency free design.

The system supports all required game controller.

Methods were written to also allow joystick support. Specifically the Microsoft Xbox controller. The system is user friendly and allows a user to execute functions or any action after a key/ joystick state has been changed.

The message system was created to allow a subject of any class to send a message of any type to an observer that is also of any class. This allows any subsystem to implement the subject or observer class. This also allows the messaging system to be adaptable to any future class a user might create. This was done using C++ Anytypes.

The system shall provide a user input handling subsystem which will allow handling of current keys and mouse buttons the mapping of keys to functions.

The system shall provide a massaging system which will provide the ability to manage and update the state of all core system objects within the game world.

The requirements that relate to my role on the project are:

The system shall provide a user input handling subsystem which will allow handling of current keys and mouse buttons the mapping of keys to functions.

The entire project was implemented with a test driven development approach. Unit tests were implemented as the functionality was added to the project.

A bottom-up approach was used to test the entire project. Google test was used for test automation.

System Design

The design of the user input handler is a series of Boolean that return true if keys are pressed.

Along with methods to control the mouse.

Implementation

The input handling system was created to allow fast raw input directly from the I/O devices. The system was created with platform independence in mind. SFML was used to help achieve platform independence. It provides methods to directly access devices connected to the computer such as a keyboard or joystick.

Methods were written to also allow joystick support. Specifically the Microsoft Xbox controller. The system is user friendly and allows a user to execute functions or any action after a key/ joystick state has been changed.

Acknowledgement

This project was improved thanks to the help I received from my group members Brian Lara, Michael Weschler, Antonio Diaz, and Julian Nodarse.