Layout

Introduction

This codebase is a simple object-based game engine with no graphical interface. Custom object classes are derived from existing object classes, and instances are defined in .INST files, which are loaded at runtime. This engine lacks some common features of game engines; these will often need to be worked around when using this engine.

This documentation will walk through how to use the engine in an application. It will start with explaining how to include and link the dependent libraries of the engine. Then it will explain how to run the engine from the main() function. It will show how to create content, new object classes and instancing data, for the engine. And at the end, the documentation will show the built-in object classes' attributes and notable functions.

This engine only supports the Windows environment.

Dependencies

This Engine has multiple libraries that it is dependent on. Files and linking instructions can generally be found at each library's website. However, these can be difficult to follow, so a rundown of the dependencies is included here. These instructions are most accurate to working in Visual Studio.

Vulkan – Download the Vulkan SDK from Vulkan.lunarg.com. Add the Include folder to project Includes, and link with the Lib folder with vulkan-1.lib as the dependency file.

GLFW – Download the 64-bit precompiled Windows binaries from glfw.org. Add the Include folder to project Includes, and link with the lib-vc2019 folder with glfw3.lib as the dependency file.

GLM – Download GLM from github.com/g-truc/glm as a zip. Add the extracted directory to project includes.

STB – Download STB from github.com/nothings/stb as a zip. Add the extracted directory to project includes.

SFML Audio – Download SFML from sfml-dev.org. Add the include folder to project includes. Link with the lib folder. Add sfml-system-s.lib, sfml-audio-s.lib, winmm.lib, ogg.lib, vorbis.lib, vorbisfile.lib, vorbisenc.lib, flac.lib, and openal32.lib as dependency files. Copy OpenAl.dll from the bin folder to the project application's running directory. Add SFML_STATIC as a preprocessor definition. When running debug mode, append -d after -s in sfml-system-s.lib and sfml-audio-s.lib.

FreeType – Download the FreeType Windows binaries from github.com/ubawurinna/freetype-windows-binaries. Add the include folder to project includes. Link with the release static win64 folder, with freetype.lib as the dependency file.

Running an Application

In the file you create with the main() function, include “Engines/Main/Main_Engine.h”. Create an instance of the Main_Engine class, and the engine_start_info struct. Call the Main_Engine instance’s start() function with the engine_start_info instance as the parameter to run the application. The application will run until the window is closed, at which point the application will shut down automatically.

Engine_Start_Info Struct

windowTitle, string – the label that appears at the top of the application window

windowWidth, int – the width of the application window in screen coordinates

windowHeight, int – the height of the application window in screen coordinates

enforceAspectRatio, bool – whether or not the aspect ratio of the window is preserved when the window is resized

aspectRatioWidth, int – the width component of the aspect ratio, only has an effect when enforceAspectRatio is set to true

aspectRatioHeight int – the height component of the aspect ratio, only has an effect when enforceAspectRatio is set to true

maxFramerate unsigned int – the forced maximum processing speed of the application in frames per second

keysToPoll vector<pair<string, int>> – the keys on the keyboard that the application will poll state updates for, first value: user-defined label, second value: GLFW key value

AddedObjectCreateFunctions vector<pair<string, Object* (*)()>> – lambda functions for creating user-defined objects, first value: object class identifier, second value: lambda function

rootObject_filePath string – the file path of the .INST file that contains the root object’s instance data

rootObject_objectName string – the object identifier of the root object in the root object file

AddedObjectCreateFunctions example:

StartInfo.AddedObjectCreateFunctions = {

{“Object_Class_Example”, [] { return (Object*) new Object_Example(); }} }

Instance Files

The information for game objects is stored in .INST files. The .INST file structure was designed to be easily human readable and human writable. Format is case sensitive.

Keywords

Object – signifies the start of an instance's data, stops filling the previous object instance's data

Child – signifies another object that will be instanced as a child object of the instance which's data is currently being filled.

Here – when following the Child keyword, signifies that the child object's instance data is in the same file

Integer, Double, Boolean, Character, String – attribute types

Syntax

Defining a new instance:

Object object_name object_class

Filling an attribute:

attribute_type attribute_name attribute_value

if attribute_type is **String,** surround attribute_value with quotes, "attribute_value"

Assigning a Child Instance:

Child filepath child_object_name

Filepath is relative to the application's running directory

Surround filepath with quotes, "filepath"

Filepath can be replaced with **Here** if the child instance is defined in the same file

Comments:

Start a commented line with the pound symbol #

Example

Object example_instance example_object_class

Integer example_integer 99

Child "resources/instances/example.inst" example_instance2

Object Classes

All object classes share Object as a base object. The Object base class provides the standard functionality for object instances. An object class may have one or multiple other object classes as its base classes, through the C++ feature of multi-inheritance.

Attributes

Rather than storing values in individual member variables, values are stored in a map of Attributes. This map is what makes it possible to load data from instance files without modifying the creation process for every object class.

Attributes can be declared as types: Integer, Double, Boolean, Character, or String. These attributes can be cast to or assigned by the corresponding C++ types, or class in the case of String. Integer values are signed, and Double has double-point precision.

Creating an Object Class – Part 1, The Header File

Include the classes that will be used as base classes

```
#include “Objects/Base/Object.h”
```

Optionally include headers with special pointers to be requested from the main engine

```
#include “Engines/Main/Input_Handler”
```

Create constants for the class identifier and any attributes

```
const auto OBJECT_CLASS_EXAMPLE = “example_object_class”;

const auto ATTRIBUTE_EXAMPLE_INTEGER = ”example_integer”;
```

Declare the object class with the base classes declared as public virtual

```
class example_object : public virtual Object {
```

Optionally declare pointers for special pointers requested from the main engine

```
private:

    Input_Handler* input_handler_ptr
```

Declare the constructor and destructor with the destructor as virtual

```
public:

    example_object();

    virtual ~example_object();
```

Optionally declare an afterCreation function, a beforeDestruction function, a process function, and any helper functions

```
public:
    static void afterCreation(Object* selfptr);
    static void beforeDestruction(Object* selfptr);
    static void process(Object* selfptr, float delta);
```

Creating an Object Class – Part 2, The Implementation File

Define the constructor and the destructor. The destructor should be left empty in most cases. Attribute defaults are defined in the constructor. The second value of addProcessFunction is the process's reverse priority, the lower the value, the earlier in the frame it is called.

```
example_object::example_object() {
    addClassIdentifier(OBJECT_CLASS_EXAMPLE);
    createAttribute(ATTRIBUTE_EXAMPLE_INTEGER, Attribute::types::Integer);
    setAttribute(ATTRIBUTE_EXAMPLE_INTEGER, 1);

    addAfterCreationFunction(&afterCreation);
    addBeforeDestructionFunction(&beforeDestruction);
    addProcessFunction(&process, 10);

    addRequestedPointer(PTR_IDENTIFIER::INPUT_HANDLER_PTR,
        &input_handler_ptr);
}
example_object::~example_object() {}
```

Define any declared functions.

Special Functions

Object classes have a few special functions. These are process functions, after creation functions, and before destruction functions. These are lists of static functions which take an object instance as a parameter, and these lists can have additional functions added by each derived class.

After Creation Functions

These functions are called at the end of the creation step of each frame. Any object that was created that frame has its after creation functions called. These functions will be called in no particular order.

static void **afterCreationFunction**(Object* selfPointer)

Before Destruction Functions

These functions are called at the beginning of the destruction step of each frame. Any object that is queued to be destroyed that frame will have its before destruction functions called. These functions will be called in no particular order.

static void **beforeDestructionFunction**(Object* selfPointer)

Process Functions

These functions are called during the processing step of each frame. These functions are sorted in order of the priority value passed to them in addProcessFunction(). These functions will be called in order of lowest priority value to highest. Process functions take a float value as a second parameter. This is delta value, which is the time elapsed between frames in seconds.

static void **processFunction**(Object* selfPointer, float delta)

Missing Features

This engine lacks a few common helpful features, making it more difficult to work with. Firstly, there is no custom editor and no GUI. A custom editor with or without a GUI would be its whole own project. Secondly, there is no function to build an empty instance. All instances must be read from a .INST file. This can be worked around by having a file of empty instances, but this has its limitations. Also, when instancing an object, though attributes can be filled in the creation call, the passed values cannot add or modify child instances of the object, nor can they change object names., This engine uses a strict unique identifier system. This can make some relationships between objects more difficult to make use of.

Other limitations of the engine are specific to individual modules. The graphics engine only runs a single render pass per frame, so the engine cannot manage transparent objects. Repeat images are also not shared, but instead have multiple copies, costing extra memory, especially in the case of text.

Built-In Classes

Object

The base class from which all object classes are derived. Provides basic universal functionalities.

Class Identifier:

OBJECT_CLASS_BASE Object_Base

Protected:

void **CreateAttribute**(string attributeName, Attribute::types attributeType)

Declares a new attribute for objects of the calling class

void **addClassIdentifier**(string classIdentifier)

Adds an identifier that is used to show that the object is of the identified class

void **addRequestedPointer**(PTR_IDENTIFIER pointer_identifier, void* ptr)

Requests a pointer from the main engine. The pointer requested is the pointer designated by the PTR_IDENTIFIER enum. The requested pointer is passed to ptr during instancing. The pointer ptr points to a pointer with a type that matches the requested pointer.

void **addProcessFunction**(void(*func)(Object*, float), int priority)

Adds a function to the class's process functions

void **addAfterCreationFunction**(void(*func)(Object*))

Adds a function to the class's after creation functions

void **addBeforeDestructionFunction**(void(*func)(Object*))

Adds a function to the class's before destruction functions

Public:

Attribute **getAttribute**(string attribute_name)

Returns the instance's attribute of the given name

void **setAttribute**(string attribute_name, Attribute attribute_value)

Sets the instance's attribute of the given name to the passed value

bool **is_class**(string class_identifier)

Returns true if the object either is or is derived from the queried class.

string **getIdentifier**()

Returns the unique identifier of the instance

string **getParentIdentifier**()

Returns the unique identifier of the instance's parent instance

unordered_set<string>& **getChildrenIdentifiers**()

Returns the unique identifiers of the instance's child instances

Object* getObject(string object_identifier)

Returns a pointer to the object with the passed unique identifier. Returns a null pointer if there is no object with the passed identifier.

void **queueCreateObject**(string filepath, string object_identifier, string parent_identifier,

vector<pair<string, Attribute>> modified_attributes = {})

Queues an object for creation that frame. Filepath is the .INST file relative to the application's running directory. object_identifier is the identifier of the object in the .INST file to load. parent_identifier is the unique identifier of the object to assign as the new instance's parent instance. During creation, the new object's identifier will be modified if an object already has its identifier. modified_attributes holds values of any attributes that are to be different from what is in the .INST file.

void **QueueDestroyObject**(string object_identifier)

Queues the object with the passed unique identifier to be destroyed that frame

Void **ChangeParentObject**(string new_parent_identifier)

Changes relation identifiers of objects so that the passed identifier becomes the parent object of the calling instance

Object_Spatial : Object

Class Identifier:

OBJECT_CLASS_SPATIAL Object_Spatial

Attributes:

Double ATTRIBUTE_SPATIAL_POSITION_X SpatialPosX – the x local coordinate

Double ATTRIBUTE_SPATIAL_POSITION_Y SpatialPosY – the y local coordinate

Double ATTRIBUTE_SPATIAL_POSITION_Z SpatialPosZ – the z local coordinate

Double ATTRIBUTE_SPATIAL_ROTATE_NOD SpatialNod – pitch (x axis rotation)

Double ATTRIBUTE_SPATIAL_ROTATE_TURN SpatialTurn – yaw (y axis rotation)

Double ATTRIBUTE_SPATIAL_ROTATE_TILT SpatialTilt – roll (z axis rotation)

Double ATTRIBUTE_SPATIAL_SCALE_X SpatialScaleX

Double ATTRIBUTE_SPATIAL_SCALE_Y SpatialScaleY

Double ATTRIBUTE_SPATIAL_SCALE_Z SpatialScaleZ

Boolean ATTRIBUTE_SPATIAL_PARENT_TRANSFORMATIONS_INHERIT InheritParentTransform

Public:

virtual glm::mat4 **getTransformationMatrix**(glm::mat4 childTransform = glm::mat4(1))

Returns the transformation matrix built by the spatial object's attributes. If InheritParentTransform is true, the parent object calls getTransformationMatrix with the matrix passed as childTransform.

Object_Sprite : Object_Spatial

Class Identifier:

OBJECT_CLASS_SPRITE Object_Sprite

Attributes:

String ATTRIBUTE_SPRITE_IMG_FILEPATH SpriteFilepath

Double ATTRIBUTE_SPRITE_WIDTH SpriteWidth – width of sprite object, not image pixels

Double ATTRIBUTE_SPRITE_HEIGHT SpriteHeight – height of sprite object, not image pixels

Boolean ATTRIBUTE_SPRITE_VISIBLE SpriteIsVisible

Protected:

virtual void **loadImage**()

queues the graphics engine to load the object's image

virtual void **updatePushConstants**()

queues updating the push constant information of the image for the graphics engine

virtual void **unloadImage**()

queues the graphics engine to remove the object's image

Object_AnimatedSprite : Object_Sprite

Class Identifier:

OBJECT_CLASS_ANIMATEDSPRITE Object_AnimatedSprite

Attributes:

String ATTRIBUTE_ANIMATEDSPRITE_SPRITESHEET_FILEPATH SpriteFilepath – alias

Integer ATTRIBUTE_ANIMATEDSPRITE_NUM_ROWS AnimatedSpriteRowCnt – number of rows in spritesheet

Integer ATTRIBUTE_ANIMATEDSPRITE_NUM_COLUMNS AnimatedSpriteColCnt – number of columns in spritesheet

Integer ATTRIBUTE_ANIMATEDSPRITE_FIRST_INDEX AnimatedSpriteFirstIndex – index of first frame in the animation

Integer ATTRIBUTE_ANIMATEDSPRITE_LAST_INDEX AnimatedSpriteLastIndex – index of last frame in the animation

Integer ATTRIBUTE_ANIMATEDSPRITE_CURRENT_INDEX AnimatedSpriteCurrIndex – index of current frame

Double ATTRIBUTE_ANIMATEDSPRITE_TIME_BETWEEN_IMAGES AnimatedSpriteDelay – time between sprite frames in seconds

Boolean ATTRIBUTE_ANIMATEDSPRITE_ISPLAYING AnimatedSpriteIsPlaying

Boolean ATTRIBUTE_ANIMATEDSPRITE_LOOP AnimatedSpriteDoesLoop

Object_Text : Object_Sprite

Class Identifier:

OBJECT_CLASS_TEXT Object_Text

Attributes:

String ATTRIBUTE_TEXT_STRING TextString – text displayed by object

Integer ATTRIBUTE_TEXT_DISPLAY_COUNT TextCount – the number of characters to display

Integer ATTRIBUTE_TEXT_ROW_LENGTH TextRowLen – the number of characters before wrapping to the next line

Integer ATTRIBUTE_TEXT_ROW_COUNT TextRowCount – the number of rows of text

Object_Camera : Object_Spatial

Class Identifier:

OBJECT_CLASS_CAMERA Object_Camera

Attributes:

Double ATTRIBUTE_CAMERA_DEPTH_MIN MinCameraDepth

Double ATTRIBUTE_CAMERA_DEPTH_MAX MaxCameraDepth

Double ATTRIBUTE_CAMERA_FOV CameraFOV – angle of camera view from top to bottom

NOTE: due to differences between Vulkan and GLFW, the actual minimum depth is around the halfway point between minCameraDepth and maxCameraDepth. However, the FOV is still accurate to the set minCameraDepth.

Integer ATTRIBUTE_CAMERA_RATIO_WIDTH CameraRatioWidth – aspect ratio width

Integer ATTRIBUTE_CAMERA_RATIO_HEIGHT CameraRatioHeight – aspect ratio height

Public:

virtual glm::mat4 **getCameraTransformationMatrix**()

Returns the camera's view transformation matrix

Object_SoundOutput : Object

Class Identifier:

OBJECT_CLASS_SOUNDOUTPUT Object_SoundOutput

Attributes:

String ATTRIBUTE_SOUND_FILE SoundFilepath

Double ATTRIBUTE_SOUND_VOLUME SoundVolume – range (0,100)

Boolean ATTRIBUTE_SOUND_ISMUSIC SoundIsMusic – streams audio if true, loads audio if false

Boolean ATTRIBUTE_SOUND_DOES_LOOP SoundLoop

Double ATTRIBUTE_SOUND_LOOP_START SoundStart – the start point of the loop in seconds

Double ATTRIBUTE_SOUND_LOOP_END SoundEnd – the loop point of the track in seconds

Double ATTRIBUTE_SOUND_START_OFFSET SoundStartOffset – the point in the track the object starts playing at in seconds

Public:

void **play**()

Starts playing the audio

void **stop**()

stops playing the audio

float **getOffset**()

gets the current offset of the audio in seconds

Object_Collision : Object_Spatial

Class Identifier:

OBJECT_CLASS_COLLISION Object_Collision

Attributes:

Double ATTRIBUTE_COLLIDER_WIDTH ColliderWidth

Double ATTRIBUTE_COLLIDER_HEIGHT ColliderHeight

Double ATTRIBUTE_COLLIDER_DEPTH ColliderDepth

Integer ATTRIBUTE_COLLIDER_MASK_OWN ColliderMask – the binary mask that signifies for other colliders if this object can be collided with

Integer ATTRIBUTE_COLLIDER_MASK_TARGET ColliderMaskTarget – the binary mask that signifies what collider masks this object will search for for collisions

Public:

vector<string> **getCollidingObjects**()

Returns the unique identifiers of objects that this object overlaps, using this object's collider mask target

Protected:

virtual bool **checkIsColliding**(Object_Collision* collider)

checks if this object overlaps with the other collider

Object_ActiveCollision : Object_Collision

Class Identifier:

OBJECT_CLASS_ACTIVECOLLISION Object_ActiveCollision

Public:

vector<string> **move**(double movX, double movY, double movZ, float delta, double magnitude=1)

Moves the position of the object by each component of xyz multiplied by magnitude and delta. The object does not move in the case of a collision, and the object returns a vector of all objects that it collided with.