System features – View

# Introduction

The engine is designed to assist the user in all parts of the process when making an environment, this also extends to the visualization of said environment, however since our goal is to have as few restrictions on the model our knowledge of how that view’s form is shaped is very limited.

# Concept

The view API which the engine provides is exactly four abstract classes that are meant to be implemented by the user.

The four classes can be seen on fig**. DomainUMLView**, we will go through each class and explain how they are meant to be implemented.

## XmasView

The ##XmasView## class is very simple it only provides a single method that is required to be implemented, when the engine starts the view up it generates a thread for the view and the ##Start## method is the first method to be executed inside that thread. The start method should contain an endless loop that on a time interval updates the view. Another task of the implemented view is also to update its ##ThreadSafeEventManager##, the ##ThreadSafeEventManager## ensures that events that are sent from model thread of the engine is not immediately executed instead they lie dormant in the ##ThreadSafeEventManager## until the view thread is ready to execute them. How many events that one wish to trigger is up to the user, We also provide the appropriate methods so that the user can say exactly how long he wishes to wait for the next event, or if it should timeout. The ##ThreadSafeEventManager## is very important to the view as without it designing view code comes complicated as one need to constantly ensure that no concurrent bugs has been applied to the system.

## WorldView

The ##WorldView## class is added because of the long term benefits, as of now it provides nothing for the designer, however if we found benefits to add to the class, making it ahead of time even if it is empty can have many benefits as the project expands.

## EntityView

Much like the ##WorldView## class the ##EntityView## is also very minimal however, it enforces certain things that the user of the engine should take care off. First off it automatic makes an ##ThreadSafeEventQueue## from the entity and attaches that ##ThreadSafeEventQueue## to the ##ThreadSafeEventManager## which should be provided by the ##XmasView##. The idea is that all events the XmasView wishes to listen to should be done by registering its triggers to the ##ThreadSafeEventQueue##. This will ensure that when the view updates the ##ThreadSafeEventManager## all events pertaining to the specific entity is also updated on the ##EntityView’s## triggers, but done so on the view thread instead of the model thread separating the two threads completely.

## ViewFactory

The ##ViewFactory## is meant to include all objects with low life cycle used by the view, it is also designed specifically to construct new ##EntityViews## during runtime of the engine. In order to know which ##EntityView## belongs to which ##XmasEntity## one is required to register all types of ##XmasEntities## and link it to its counterpart ##EntityView##. For instance assume you have a class inheriting ##XmasEntity## called ##Wall##, and the ##Wall’s## representation called ##WallView##, then you need to manually register inside the ##ViewFactory## that ##Wall## is represented by ##WallView##.

# Summary

The view framework provides four classes each with their own advantages; they each represent a part of the model of engine. They are designed to assist the user in keeping his code threadsafe so that as few problems as possible arise.