KivEnt Documentation

Release 2.2.0

Jacob Kovac

Oct 17, 2020

Contents

If you are just starting you may be interested in the tutorials.

KivEnt is a framework for building performant, dynamic real-time scenes in Kivy. While not as powerful as something like the Unreal engine or Unity3d, KivEnt is capable of creating games that handle several thousands to tens of thousands of entities, depending on what type of processing we are doing on them. You can easily have a hundreds thousand static sprites rendered in the background if they do not have any dynamic processing. At the same time, almost the entire API is accessible through Python, with a more performant cythonic API for those looking to get closer to the metal. Even without creating any cython gamesystems, you ought to be able to create games that feature up to several thousand game objects at once.

The only dependency for the kivent_core module is Kivy itself. Additional modules may have other requirements, such as kivent_cymunk module being based on Chipmunk2d and its cymunk wrapper.

An entity-component architecture is used to control game object state and the logic of processing the game objects. This means that your game objects will be made up of collections of independent components that stricly hold data; each component corresponds to a GameSystem that will perform all data processing on the components, in the update loop each frame, and as a result of user interaction or other programmaticaly generated events. All memory for the built-in components is allocated statically: if you would like learn more about memory management, read here.

KivEnt is built with a modular architecture and designed to have both a python api and a c-level cython api that allows more performant access to your game data. This makes it suitable for quickly prototyping a mechanic completely in python, and relatively trivial to then deeply cythonize that GameSystem if you find it to be performance sensitive. This process has already been done for the built-in components meaning they are ready for you to build new, performant game systems on top of them.

The entire framework is made available to you with an MIT license so that you have the freedom to build whatever you want on top of it and monetize it however you like.

CHAPTER 1

GameWorld

CHAPTER 2

Entity

This Class has cdefed functions that cannot be read by Sphinx. Read the source if you want to find out more about using them.

CHAPTER 3

Game Systems

Most of these Classes have cdefed functions that cannot be read by Sphinx. Read the source if you want to find out more about using them.

3.1 Python GameSystems

3.2 Cython GameSystems

3.3 Aggregators

The Aggregator classes make it easier to write the logic for a GameSystem's update loop. Rather than having to fetch the components yourself from each GameSystem MemoryBlock, you can instead expend a little more memory and store an array of pointers to the appropriate components for your processing.

This expends a little more memory, and adds a layer of indirection but makes it significantly easier to write the update loop in cython.

3.4 Position Systems

3.5 Rotate Systems

3.6 Scale Systems

3.7 Color Systems

3.8 Rendering Systems

3.9 Controlling the Viewing Area

CHAPTER 4

Managers

Most of these Classes have cdefed functions that cannot be read by Sphinx. Read the source if you want to find out more about using them.

4.1 System Management

4.2 Resource Managers

Resource handling is not quite done yet so consider these sections slightly experimental and know they will probably change in the future. As of 2.1 a new ModelManager has been introduced with slightly different functionality.

CHAPTER 5

The Cymunk Module

These classes have cdefed functions that cannot be read by Sphinx. Read the source if you want to find out more about using them.

This module extends the base functionality of KivEnt by integrating it with the Chipmunk2d Physics engine. This will allow you to perform complex collision detection and real-time 2d physics calculations on large numbers of entities simultaneously.

5.1 Physics

5.2 Interaction

CHAPTER 6

Particles

The particles module can be used to easily add particle effects to your app. You will be most concerned with using the EmitterSystem module directly, but be sure you have an EmitterSystem, ParticleSystem, and ParticleRenderer. You can use the Particle Panda 2 application to produce your own effects using a graphical editor.

6.1 Emitters

6.2 Particles

6.3 Renderers

CHAPTER 7

Rendering

Most of these Classes have cdefed functions that cannot be read by Sphinx. Read the source if you want to find out more about using them.

7.1 VBOs

7.2 Instructions

7.3 Models

7.4 Vertex Formats

7.5 Frame Objects

7.6 Batching

CHAPTER 8

Memory Handlers

Most of these Classes have cdefed functions that cannot be read by Sphinx. Read the source if you want to find out more about using them.

8.1 Buffer

8.2 MemoryBlock

8.3 MemoryPool

8.4 MemoryZone

8.5 BlockZone

8.6 ZonedBlock

8.7 BlockIndex

8.8 PoolIndex

8.9 ZoneIndex

8.10 IndexedMemoryZone

8.11 memrange

CHAPTER 9

The Maps Module

These classes have cdefed functions that cannot be read by Sphinx. Read the source if you want to find out more about using them.

This module extends the base functionality of KivEnt by integrating it with the Tiled map editor. See example 14_tmx_loader for a practical demonstration of using these systems.

9.1 Tiled Managers

If you are using the kivent_maps module, an additional manager will be available to aid in the use of Tiled's map format.

9.2 Systems

9.3 Utils

9.4 Map Data