Tiny Game Engine V2
Written by Ridge Batty

KEY CONCEPTS IN TGE

TGE is a fast, flexible, and extendable barebones 2D game engine/framework for creating retro style arcade action games and mini games. The concepts presented in this document are important pieces of the core functionality of TGE.

ENGINE CLASS

Engine class is a single instance (singleton) class which contains the game state, utilities, and tools for creating, debugging, and running a TGE game.

In TGE v2 Engine contains an instance of ***CanvasRenderer*** class, accessible via *renderingSurface* property. It represents the drawing surface for all game content.

To start a new game project, you typically need to import the **Engine** module which gives access to *Engine* object.

JavaScript

```
import * as TGE from '/engine/v2/engine.js';
const Engine = TGE.Engine;
```

GAMELOOP CLASS

GameEngine class is also a singleton class which is automatically created for you. It is accessible via *Engine* object's gameLoop property.

Typically, you should not need to access *GameEngine* object at all. The most important methods are surfaced by the *Engine* object. For example, to start and stop the game loop, you can call Engine.start() and Engine.pause() respectively.

GameLoop takes care of **ticking** (applying game logic) and **updating** (compositing and rendering) of all game objects in two distinct threads which run in parallel.

TECHNICAL STUFF

On every tick (default 60 times per second) *GameLoop* walks through all objects in Tickables and Actors arrays and calls their tick() method. Make sure the logic you run in tick() method is as fast as possible. If game logic fails to keep up with the fixed tick rate, it may cause your game to behave inconsistently or make it outright unplayable. This is a special concern in time and physics-based games.

In a separate thread, *GameLoop* renders the graphics in its internal _render() method which is fired periodically by window.requestAnimationFrame() when the *GameLoop* is running. Rendering is done as often as the hardware allows. It is capped by VSync, i.e. on a typical 60Hz monitor the rendering is capped at 60 frames per second.

In its internal _render() method *GameLoop* calls update() method for all objects in its zLayers array. When new *Actor* is created using GameLoop.add(), it will get added into the zLayers and Actors arrays.

You can add your own custom objects in the GameLoop.tickables array which need to be updated once per tick. The added object must have tick() method implemented which may contain any custom game logic. The main difference between Tickables and Actors arrays is that the latter is managed. In other words, you should not add or remove objects from the Actors array manually. TGE takes care of removing destroyed actors from the array. Instead, use Actor.destroy() method to delete and GameLoop.add() or Engine.addActor() to create new Actors.

ACTOR CLASS

Actor is an essential component of an arcade game. It represents any game object with properties such as transform, collision channel, and visual representation as well as a bunch of other properties and methods. ***Actor*** is a low-level class which you can extend to create your own custom game objects. For convenience, several extended classes are already provided for you: ***Player***, ***Enemy***, ***Projectile***, ***Consumable*** and ***Layer***.

RENDERING PIPELINE

TGE uses HTMLCanvasElement for rendering graphics. The canvas element is automatically created by the engine on a ***CanvasSurface***, which is a wrapper class for HTML canvas.

Although ***GameLoop*** class plays a central role in displaying graphics, it does not directly render anything on the screen. Instead, it loops through its internal object arrays and calls the update() function of those objects which in turn execute the actual draw calls. *GameLoop* acts as only a framework for coordinating the game logic and graphics composition.

All objects which need to be rendered on screen must implement update() method, which should contain only the minimum processing required to draw the object. The render target is typically *Engine.renderingSurface*.

JAVASCRIPT

```
update() {
    if (this.isVisible) {
        Engine.renderingSurface.drawImage(this.position, this.img);
    }
}
```

When an *Actor* is created using Engine.gameLoop.add() or its alias Engine.addActor(), it is automatically inserted into the *GameLoop* update loop. Calling *Actor.destroy()* method its allocated memory is released, and the *Actor* gets removed from the *GameLoop*.

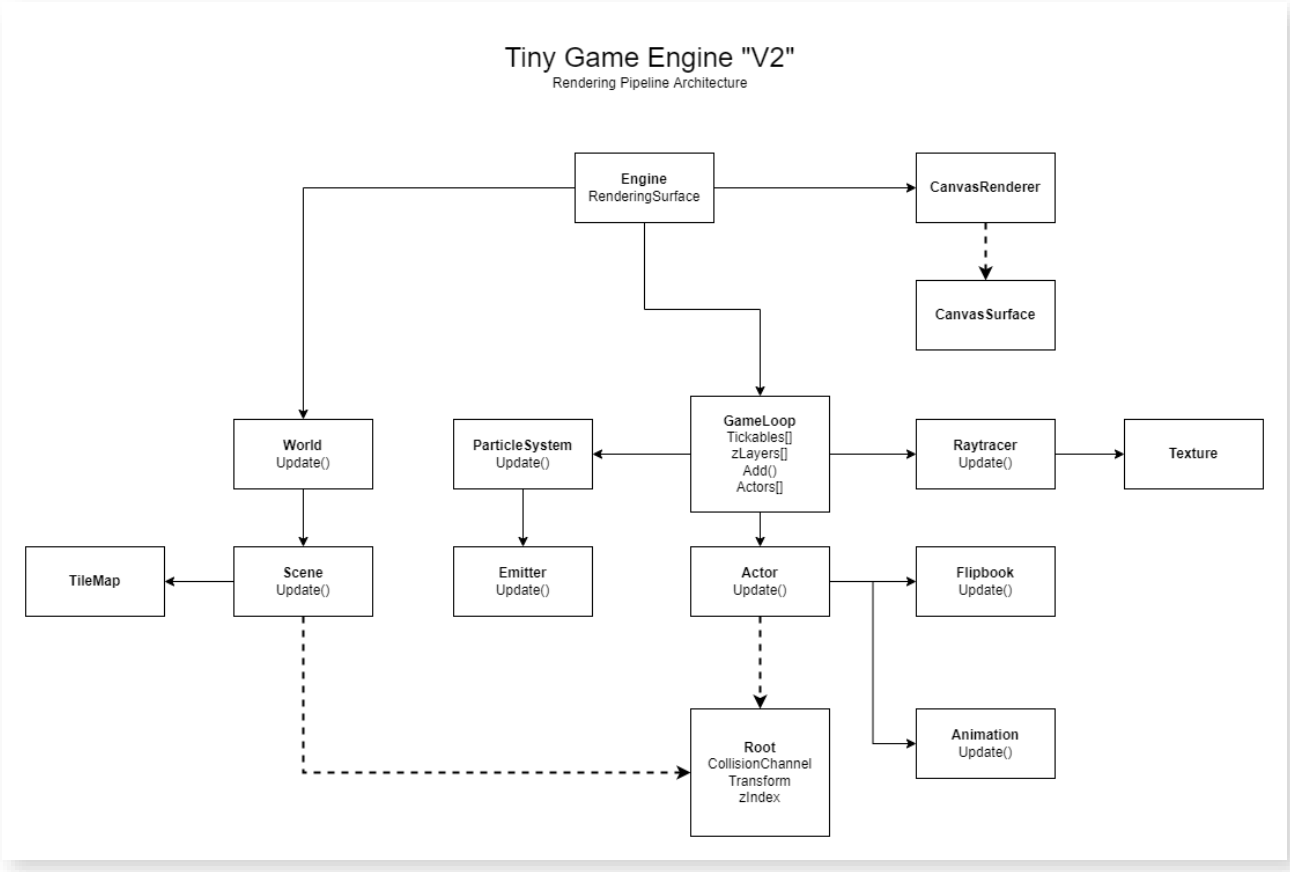

Figure 1. The graph illustrates how several game classes implement update() and tick() methods and are driven by *GameLoop*.

Inheritance is marked with dashed line pointing from descendant to ancestor class. For example, ***Root*** is the ancestor of ***Actor*** class. Solid line is pointing from the class which has a property to the class which is the value (or one of the values) of that property. For example, an instance of *GameLoop* is accessible as a property of *Engine*.