PTB GAME ENGINE

May 21, 2019

Nigel Bess

Contents

1 Overview

The purpose of the game engine is to simplify software development, troubleshooting, and to remove the developer's need to interact with underlying game structure. It does this by encapsulating tasks within the software into objects of type `GameObject`, and interacting with `GameObject`s automatically.

Upon instantiation of a game engine object, it finds all the `GameObject`s in the workspace. When the game engine is started it first calls the `Awake()` method on each of these `GameObject`s. Then it enters a loop in which it calls the `Update()` method on each `GameObject`, and continues to do so until the `Quit()` method is called on the game engine.

This game engine is used for the Autobehavior Training Rig software, but could be used to create any 2D game.

1.1 Outline of Basic Data Types

Below are short summaries of the purpose of the data types included in the game engine. For usage and more details, refer to section 3.

1.1.1 GameEngine

There should exist one and only one `GameEngine` in the workspace. Simply speaking, the `GameEngine` finds all `GameObject`s, calls each `GameObject's Awake()` method, then calls each `GameObject`'s `Update()` method repeatedly until the game is quit.

1.1.2 GameObject

`GameObject` is the class from which all objects interacting with the game loop should inherit. The following classes inherit from `GameObject`.

1.1.3 Renderer

There should exist no more than one `Renderer` in the workspace. The `Renderer` is a `GameObject` that handles all rendering of images to the screen. When the `Renderer`'s `Update()` function is called, it draws each `Renderable` to the screen, according to the `Renderable`'s render order.

1.1.4 Renderable

`Renderable`s are `GameObject`s that render to the screen.

1.1.5 PhysicsObject

`PhysicsObject`s are `Renderable`s that maintain persistent velocity and update their position automatically.

2 Setting up and Running a Game

1. Create all `GameObject`s desired.
2. Initialize desired parameters of all `GameObject`s.
3. Create a `GameEngine`.
4. Start the game

3 Detailed Documentation of Data Types

This section includes detailed usage of the each basic type in the game engine. Any methods or properties not listed are only used for the underlying game engine structure.

3.1 GameEngine

3.1.1 Constructor: GameEngine(sceneFileName)

Finds all objects in the workspace. If using the optional parameter, [sceneFileName], it loads the workspace saved in [sceneFileName].mat.

3.1.2 GameEngine.Start()

The `Start` method begins the game. First, it passes a reference to itself to each `GameObject` in the workspace. It then calls the `Awake` method on each `GameObject`. It then initializes the Renderer. Finally, it enters the main game loop, in which the `Update` method is called for each enabled `GameObject`. **Note:** The order in which `Update` is called for each `GameObject` is consistent, but for best practice code should be written in a way that order does not matter.

3.1.3 GameEngine.Quit()

The `Quit` method breaks from the main game loop. It calls the `OnQuit` method for each `GameObject` if available. It then closes the rendering window.

3.1.4 GameEngine.FindObjectsOfType(typeChar)

Inputs `typeChar`, a character array defining the name of a class or base class (case sensitive). Returns all objects (from those in the workspace before the game engine was created) of class `typeChar`. This includes objects that inherit from class `typeChar`.

3.1.5 GameEngine.minTimeDelta

Private property defining a time (in seconds) to pause between frames. Set to zero for maximum performance without background processes. Increase the value to improve background process performance on a CPU with low thread count. Default value 0.01 seconds. Public setter at `GameEngine.SetMinTimeDelta(time)`.

3.1.6 GameEngine.GetTime()

Returns time (in seconds) since the game was started.

3.1.7 GameEngine.GetTimeDelta()

Returns time (in seconds) since the last frame.

3.2 GameObject

3.2.1 GameObject.Game

The current active `GameEngine`.

3.2.2 GameObject.Renderer

The current active `Renderer`.

3.2.3 GameObject.Awake()

Called before the first frame, and before rendering has begun.

3.2.4 GameObject.enabled

A public property (boolean) determining if the object should be updated this frame.

3.2.5 GameObject.Update()

Called once per frame, as long as the object is enabled.

3.2.6 GameObject.DelayedCall(methodName,time,param0,param1...paramN)

Calls [methodName]([params]) on this `GameObject` after [time] seconds. methodName should be a character array.

3.2.7 GameObject.DisableFor(time)

Prevents the `Update` method from being called for [time] seconds;

3.2.8 GameObject.StopAllDelayedCalls()

Prevents any pending delayed calls from resolving. If the object was temporarily disabled, it will not automatically re-enable. This also stops any delayed calls that were still going to resolve this frame.

3.2.9 GameObject.OnQuit()

Called when the game is quit.

3.2.10 GameObject.OnError()

Called when the game encounters a fatal error. If an error is encountered in a `GameObject.OnError`, the game engine will throw a warning.

3.2.11 GameObject.BaseUpdate() (Advanced)

For best practice, it is recomended to use `BaseUpdate` sparingly. While it works exactly the same as the Update method, and can be used interchangeably, The purpose of `BaseUpdate` is to pass an update function from a base class to its children without impeding on the children's update function.

3.3 Renderer

3.3.1 Constructor: Renderer(screenNum,backgroundColor)

Initializes a `Renderer` to output to the screen defined by [screenNum], with a default background color of [backgroundColor]. screenNum: 0 - single display mode, 1 - main display, 2- secondary display.

3.3.2 Renderer.WindowSize()

Returns the size of the rendering window (in pixels) in the form: [width,height].

3.3.3 Renderer.SetBackgroundColor(color)

Sets the background color to [color].

3.3.4 Renderer.ResetBackgroundColor()

Resets the background color to the default color.

3.4 Renderable

Inherits from `GameObject`

3.4.1 Renderable.image

A protected property containing the image of this `Renderable` (in image matrix form) that gets rendered to the screen

3.4.2 Renderable.position

A protected property defining the position in pixels of this `Renderable` in the form [x,y]. [0,0] is defined as the center of the screen. It has a public setter/getter at `Renderable.SetPosition` and `Renderable.GetPosition`.

3.4.3 Renderable.size

A protected property defining the size in pixels of the rect that this `Renderable` occupies. Size has the form [width, height].

3.4.4 Renderable.screenBounded

A protected property (boolean) determining whether the `Renderable` is allowed to leave the screen. If `screenBounded` is true, the `Renderable`'s position will be clamped such that its entire rect is always visible on screen.

3.4.5 Renderer.renderLayer

A public property (double) determining the order in which the `Renderable` is rendered to the screen. A `Renderer` with higher `renderLayer` will appear to be below other `Renderable`s on screen.

3.4.6 Renderable.GenerateImage()

Defines the `Renderable`'s image (in image matrix form), which gets rendered to the screen. It must be defined for any `Renderable` object.

3.4.7 Renderable.GetScreenHits(index)

Returns whether any part of the `Renderable`'s rect is outside of the rendering window along optional parameter index.
Index: 1 - Left/Right, 2 - Up/Down
Returns: -1 - Left/Top, 0 Inside window, 1 - Right/Bottom
Omitting the index parameter returns [horizotal intersection, vertical intersection]

3.4.8 Renderable.Distance(other,index)

Returns the center-to-center distance (in pixels) to `Renderable` other. Index parameter is optional, and returns distance along a specified axis.
Index: 1 - Horizontal, 2 - Vertical

3.5 PhysicsObject

Inherits from `Renderable`.

3.5.1 PhysicsObject.Velocity

Protected property defining the 2D velocity (in pixels per second) with the form [xVelocity,yVelocity]

3.5.2 PhysicsObject.SetVelocity(velocity,index)

Setter for `PhysicsObject.velocity`. Optional parameter [index] is used to alter only one entry of `PhysicsObject.velocity`

4 Conventions

4.1 Case Conventions

For reference, Pascal case and Camel case are defined as follows:

`PascalCase`

`camelCase`

Class Names	Pascal
Method Names	Pascal
Private/Protected Property Names	Camel
Public Property Names	Camel
Constant Property Names	Camel
Property Names Referring to Singletons	Pascal

4.2 Access

Methods should be used to interact between objects, and only in very rare situations should public properties be used. Every property in the game engine is either private, protected or constant except for `GameObject.enabled`, and it is recommended to follow this convention when creating games.

The exception was made for `GameObject.enabled` in order to match the Unity Game Engine.

4.3 Other

It is recommended to follow the S.O.L.I.D. object-oriented programming principles when possible.