Namespace RayGame

Classes

CollisionDetection

Provides a method for collision detection between shapes.

Engine

The main engine class that initializes and runs the game.

GameObject

Represents a game object in the Scene.

Mesh

Represents a 2D mesh consisting of a collection of vertices.

MeshRenderer

A Renderer that renders a `Mesh` associated with a `GameObject`.

SpriteRenderer

Transform

The class that holds all the transformation data for an object

Interfaces

IGameComponent

Interface for game components.

IRenderer

Interface for renderers.

Edit this page

Class CollisionDetection

Provides a method for collision detection between shapes.

Inheritance

↳ object
  ↳ CollisionDetection

Inherited Members

object.Equals(object)
object.Equals(object, object)
object.GetHashCode()
object.GetType()
object.MemberwiseClone()
object.ReferenceEquals(object, object)
object.ToString()

Namespace: RayGame

Assembly: RayGame.dll

Syntax

```
public static class CollisionDetection
```

Remarks

This is a Static class used internally to check the collision between Colliders in `GameObject` s.**Primarily for internal use**

Methods

CheckCollision(Vector2[], Vector2[])

Checks if two shapes are colliding.

Declaration

```
public static bool CheckCollision(Vector2[] shape1, Vector2[] shape2)
```

Parameters

TYPE	NAME	DESCRIPTION
Vector2[]	`shape1`	The first shape.
Vector2[]	`shape2`	The second shape.

Returns

TYPE	DESCRIPTION
bool	True if the shapes are colliding, otherwise false.

Edit this page

Class Engine

The main engine class that initializes and runs the game.

Inheritance

↳ object
  ↳ Engine

Inherited Members

object.Equals(object)
object.Equals(object, object)
object.GetHashCode()
object.GetType()
object.MemberwiseClone()
object.ReferenceEquals(object, object)
object.ToString()

Namespace: RayGame

Assembly: RayGame.dll

Syntax

```
public static class Engine
```

Fields

random

The Random Number Generator initiated with the Engine.

Declaration

```
public static Random random
```

Field Value

TYPE
Random

Properties

GAMETIME

The value associated with the amount of milliseconds passed.

Declaration

```
public static long GAMETIME { get; }
```

Property Value

TYPE
long

Methods

CreateGameObject(string)

Creates a new game object with the specified name.

Declaration

```
public static GameObject CreateGameObject(string name)
```

Parameters

TYPE	NAME	DESCRIPTION
string	name	The name of the game object.

Returns

TYPE	DESCRIPTION
GameObject	The created game object.

CreateGameObject(string, Transform)

Creates a new game object with the specified name and `Transform` >.

Declaration

```
public static GameObject CreateGameObject(string name, Transform transform)
```

Parameters

TYPE	NAME	DESCRIPTION
string	`name`	The name of the game object.
Transform	`transform`	The transform of the game object.

Returns

TYPE	DESCRIPTION
GameObject	The created game object.

CreateGameObject(string, Vector2, float, float)

Creates a new game object with the specified name, position, angle, and scale.

Declaration

```
public static GameObject CreateGameObject(string name, Vector2 Position, float Angle, float Scale)
```

Parameters

TYPE	NAME	DESCRIPTION
string	`name`	The name of the game object.
Vector2	`Position`	The position of the game object.
float	`Angle`	The angle of the game object.
float	`Scale`	The scale of the game object.

Returns

TYPE	DESCRIPTION
GameObject	The created game object.

DeleteGameObject(GameObject)

Deletes the specified game object.

Declaration

```
public static void DeleteGameObject(GameObject Gobj)
```

Parameters

TYPE	NAME	DESCRIPTION
GameObject	`Gobj`	The game object to delete.

DisableColliderRendering()

Disables rendering of colliders for all game objects. Look at `GameObject` for more detail.

Declaration

```
public static void DisableColliderRendering()
```

EnableColliderRendering()

Enables rendering of colliders for all game objects. Look at `GameObject` for more detail.

Declaration

```
public static void EnableColliderRendering()
```

FindObjectByName(string)

Finds a game object by its name.

Declaration

```
public static GameObject FindObjectByName(string name)
```

Parameters

TYPE	NAME	DESCRIPTION
string	name	The name of the game object.

Returns

TYPE	DESCRIPTION
GameObject	The game object with the specified name, or null if not found.

FindObjectOfType<T>()

Finds the first game object that has a component of the specified type.

Declaration

```
public static GameObject FindObjectOfType<T>()
```

Returns

TYPE	DESCRIPTION
GameObject	The game object with the specified component, or null if not found.

Type Parameters

NAME	DESCRIPTION
T	The type of component to look for.

GetGameObjectCount()

Gets the count of game objects currently in the engine.

Declaration

```
public static int GetGameObjectCount()
```

Returns

TYPE	DESCRIPTION
int	The number of game objects.

INIT<T>()

Initializes the game engine with a specified game component. That Custom Component is the entry point into using the Engine.

Declaration

```
public static void INIT<T>() where T : IGameComponent, new()
```

Type Parameters

NAME	DESCRIPTION
T	The type of game component to initialize.

Edit this page

Class GameObject

Represents a game object in the Scene.

Inheritance

↳ object
  ↳ GameObject

Inherited Members

object.Equals(object)
object.Equals(object, object)
object.GetHashCode()
object.GetType()
object.MemberwiseClone()
object.ReferenceEquals(object, object)
object.ToString()

Namespace: RayGame

Assembly: RayGame.dll

Syntax

```
public class GameObject
```

Remarks

The GameObjects are the building blocks of your scene. They contain a few fields that allow for adding logic and properties to make them functional.

Fields

Colliders

The List of Colliders attached to the GameObject. They are represented as a List of `Mesh` 's. You can add Colliders directly.

Declaration

```
public List<Mesh> Colliders
```

Field Value

TYPE
List<Mesh>

DEBUGCOLIDERS

This is a property of the GameObject that determines what is to be rendered. If *false*, all the `IRenderer` 's Update Method will be fired per frame. If *true*, then instead of the `IRenderer` 's, the Colliders attached to your GameObject will be rendered. This is for debugging purposes.

Declaration

```
public bool DEBUGCOLIDERS
```

Field Value

TYPE
bool

Name

This is the Name Associated with the GameObject

Declaration

```
public string Name
```

Field Value

TYPE
string

Transform

This is the `Transform` attached to the GameObject.

Declaration

```
public Transform Transform
```

Field Value

TYPE
Transform

Methods

AddComponent<T>()

Adds a component of the specified type to this game object.

Declaration

```
public T AddComponent<T>() where T : IGameComponent, new()
```

Returns

TYPE	DESCRIPTION
T	The added component.

Type Parameters

NAME	DESCRIPTION
T	The type of component to add.

AddRenderer<T>()

Adds a renderer of the specified type to this game object.

Declaration

```
public T AddRenderer<T>() where T : IRenderer, new()
```

Returns

TYPE	DESCRIPTION
T	The added renderer.

Type Parameters

NAME	DESCRIPTION
T	The type of renderer to add.

DeleteAllRenderers()

Deletes all renderers from this game object.

Declaration

```
public void DeleteAllRenderers()
```

DeleteComponent<T>()

Deletes the first component of the specified type from this game object.

Declaration

```
public void DeleteComponent<T>() where T : IGameComponent
```

Type Parameters

NAME	DESCRIPTION
T	The type of component to delete.

DeleteObjectComponents()

Deletes all components from this game object.

Declaration

```
public void DeleteObjectComponents()
```

DeleteRenderer<T>(int)

Deletes a renderer of the specified type at the given index from this game object.

Declaration

```
public void DeleteRenderer<T>(int Index) where T : class, IRenderer
```

Parameters

TYPE	NAME	DESCRIPTION
int	Index	The index of the renderer to delete.

Type Parameters

NAME	DESCRIPTION
T	The type of renderer to delete.

GetComponentNameList(bool)

Gets a list of the names of all components attached to this game object.

Declaration

```
public List<string> GetComponentNameList(bool Print)
```

Parameters

TYPE	NAME	DESCRIPTION
bool	Print	If true, prints the names to the console.

Returns

TYPE	DESCRIPTION
List<string>	A list of component names.

GetComponent<T>()

Gets the first component of the specified type attached to this game object.

Declaration

```
public T GetComponent<T>()
```

Returns

TYPE	DESCRIPTION
T	The component if found, otherwise null.

Type Parameters

NAME	DESCRIPTION
T	The type of component to get.

GetRendererNameList(bool)

Gets a list of the names of all renderers attached to this game object.

Declaration

```
public List<string> GetRendererNameList(bool Print)
```

Parameters

TYPE	NAME	DESCRIPTION
bool	Print	If true, prints the names to the console.

Returns

TYPE	DESCRIPTION
List<string>	A list of renderer names.

GetRenderer<T>(int)

Gets a renderer of the specified type at the given index from this game object. If the GameObject contains multiple types of renderers, then the index takes into account only the specified T

Declaration

```
public T GetRenderer<T>(int Index) where T : class, IRenderer
```

Parameters

TYPE	NAME	DESCRIPTION
int	Index	The index of the renderer to get.

Returns

TYPE	DESCRIPTION
T	The renderer if found, otherwise null.

Type Parameters

NAME	DESCRIPTION
T	The type of renderer to get.

HasComponent<T>()

Checks if this game object has a component of the specified type.

Declaration

```
public bool HasComponent<T>() where T : IGameComponent
```

Returns

TYPE	DESCRIPTION
bool	True if the component is found, otherwise false.

Type Parameters

NAME	DESCRIPTION
T	The type of component to check for.

HasRenderer<T>()

Checks if this game object has a renderer of the specified type.

Declaration

```
public bool HasRenderer<T>() where T : IRenderer
```

Returns

TYPE	DESCRIPTION
bool	True if the renderer is found, otherwise false.

Type Parameters

NAME	DESCRIPTION
T	The type of renderer to check for.

IsColliding(GameObject)

Checks if this game object is colliding with the specified target game object.

Declaration

```
public bool IsColliding(GameObject Target)
```

Parameters

TYPE	NAME	DESCRIPTION
GameObject	Target	The target game object to check for collisions with.

Returns

TYPE	DESCRIPTION
bool	True if a collision is detected, otherwise false.

SetTransform(Transform)

Sets the transform of this game object to the specified new transform.

Declaration

```
public void SetTransform(Transform newTransform)
```

Parameters

TYPE	NAME	DESCRIPTION
Transform	newTransform	The new transform to set.

ShiftComponent<T>(int)

Shifts a component of the specified type by a given offset in the component list.

Declaration

```
public void ShiftComponent<T>(int offset) where T : IGameComponent
```

Parameters

TYPE	NAME	DESCRIPTION
int	offset	The offset by which to shift the component.

Type Parameters

NAME	DESCRIPTION
T	The type of component to shift.

ShiftRenderer<T>(int)

Shifts a renderer of the specified type by a given offset in the renderer list.

Declaration

```
public void ShiftRenderer<T>(int offset) where T : IRenderer
```

Parameters

TYPE	NAME	DESCRIPTION
int	offset	The offset by which to shift the renderer.

Type Parameters

NAME	DESCRIPTION
T	The type of renderer to shift.

StartActions()

The function that is called when the GameObject is Initiated. Not used in most cases. **Primarily for internal use**

Declaration

```
public void StartActions()
```

UpdateActions()

This is the function that calls to update the state of the GameObject per frame.**Primarily for internal use**

Declaration

```
public void UpdateActions()
```

Edit this page

Interface IGameComponent

Interface for game components.

Namespace: RayGame

Assembly: RayGame.dll

Syntax

```
public interface IGameComponent
```

Remarks

Implement this interface to create custom game components. Each `GameObject` has a List of Components that it runs by itself. Hence, any class that implements `IGameComponent`, will be eligible to perform as a script attached to the Object. By This, It is possible to create *Prefabs* buy creating 1 class component that adds all the required components and modifications.

Properties

Container

The container GameObject for this component.

Declaration

```
GameObject Container { get; set; }
```

Property Value

TYPE
GameObject

Methods

Start()

Called when the component is Added.

Declaration

```
void Start()
```

Update()

Called every frame to update the component.

Declaration

```
void Update()
```

Edit this page

Interface IRenderer

Interface for renderers.

Namespace: RayGame

Assembly: RayGame.dll

Syntax

```
public interface IRenderer
```

Remarks

This is interface is to be inherited by all Renderers, If you wish to make a custom renderer, then u can, but i would recommend sticking to the provided renderers. **Primarily for internal use**

Properties

Container

The container GameObject for this renderer. Any Actions that want to be performed to it, is does through this reference.

Declaration

```
GameObject Container { get; set; }
```

Property Value

TYPE
GameObject

Methods

Start()

The Function Called when the Renderer is Added to an Object.

Declaration

```
void Start()
```

Update()

Called every frame to update the renderer. Primarily holds code to render the content.

Declaration

```
void Update()
```

Edit this page

Class Mesh

Represents a 2D mesh consisting of a collection of vertices.

Inheritance

↳ object
  ↳ Mesh

Inherited Members

object.Equals(object)
object.Equals(object, object)
object.GetHashCode()
object.GetType()
object.MemberwiseClone()
object.ReferenceEquals(object, object)
object.ToString()

Namespace: RayGame

Assembly: RayGame.dll

Syntax

```
public class Mesh
```

Remarks

A Mesh is a collection of Vertices that is used to display/render anything. It is always a closed loop. The Mesh also have functions that can edit it, however, be careful, as all modifications as *permanent*. It is Attached to a `MeshRenderer` to be viewed at the `GameObject`'s Position. The vertices themselves are always represented in local space.

Constructors

Mesh(Vector2[])

Initializes a new instance of the `Mesh` class with an array of vertices specified as `Vector2`.

Declaration

```
public Mesh(Vector2[] vertexArray)
```

Parameters

TYPE	NAME	DESCRIPTION
Vector2[]	`vertexArray`	An array of `Vector2` representing the vertices of the mesh.

Mesh((float, float)[])

Initializes a new instance of the `Mesh` class with an array of vertices specified as tuples.

Declaration

```
public Mesh((float, float)[] vertexArray)
```

Parameters

TYPE	NAME	DESCRIPTION
(float, float)[]	`vertexArray`	An array of tuples representing the vertices of the mesh.

Methods

AddVertex(Vector2)

Adds a single vertex to the mesh.

Declaration

```
public void AddVertex(Vector2 vertex)
```

Parameters

TYPE	NAME	DESCRIPTION
Vector2	`vertex`	The vertex to add to the mesh.

AddVertex((float, float))

Adds a single vertex to the mesh specified as a tuple.

Declaration

```
public void AddVertex((float, float) vertex)
```

Parameters

TYPE	NAME	DESCRIPTION
(float, float)	`vertex`	The vertex to add to the mesh.

AddVertices(Vector2[])

Adds an array of vertices to the mesh.

Declaration

```
public void AddVertices(Vector2[] vertexArray)
```

Parameters

TYPE	NAME	DESCRIPTION
Vector2[]	`vertexArray`	An array of `Vector2` to add to the mesh.

AddVertices((float, float)[])

Adds an array of vertices to the mesh specified as tuples.

Declaration

```
public void AddVertices((float, float)[] vertexArray)
```

Parameters

TYPE	NAME	DESCRIPTION
(float, float)[]	`vertexArray`	An array of tuples representing the vertices to add to the mesh.

DeleteLastVertex()

Deletes a Vertex from the last position.

Declaration

```
public void DeleteLastVertex()
```

DeleteVertex(int)

Deletes a vertex at the specified index.

Declaration

```
public void DeleteVertex(int Index)
```

Parameters

TYPE	NAME	DESCRIPTION
int	`Index`	The index of the vertex to delete.

DeleteVertex((float, float))

Deletes a vertex that matches the specified point.

Declaration

```
public void DeleteVertex((float, float) point)
```

Parameters

TYPE	NAME	DESCRIPTION
(float, float)	`point`	The point representing the vertex to delete.

GetVertexArray()

Gets the vertices of the mesh as an array.

Declaration

```
public Vector2[] GetVertexArray()
```

Returns

TYPE	DESCRIPTION
Vector2[]	An array of `Vector2` representing the vertices of the mesh.

InsertVertex(int, Vector2)

Inserts a vertex at the specified index.

Declaration

```
public void InsertVertex(int Index, Vector2 Vertex)
```

Parameters

TYPE	NAME	DESCRIPTION
int	`Index`	The index at which to insert the vertex.
Vector2	`Vertex`	The vertex to insert.

RotateMesh(float)

Rotates the entire mesh by the specified angle.

Declaration

```
public Mesh RotateMesh(float Angle)
```

Parameters

TYPE	NAME	DESCRIPTION
float	`Angle`	The angle in degrees by which to rotate the mesh.

Returns

TYPE	DESCRIPTION
Mesh	The rotated mesh.

ScaleMesh(float)

Scales the entire mesh by the specified scale factor.

Declaration

```
public Mesh ScaleMesh(float Scale)
```

Parameters

TYPE	NAME	DESCRIPTION
float	`Scale`	The scale factor by which to scale the mesh.

Returns

TYPE	DESCRIPTION
Mesh	The scaled mesh.

ShiftMesh(Vector2)

Shifts the entire mesh by the specified offset in Position.

Declaration

```
public Mesh ShiftMesh(Vector2 Offset)
```

Parameters

TYPE	NAME	DESCRIPTION
Vector2	`Offset`	The offset by which to shift the mesh.

Returns

TYPE	DESCRIPTION
Mesh	The shifted mesh.

ShiftMesh((float, float))

Shifts the entire mesh by the specified offset in Position.

Declaration

```
public Mesh ShiftMesh((float, float) Offset)
```

Parameters

TYPE	NAME	DESCRIPTION
(float, float)	`Offset`	The offset specified as a tuple by which to shift the mesh.

Returns

TYPE	DESCRIPTION
Mesh	The shifted mesh.

Edit this page

Class MeshRenderer

A Renderer that renders a `Mesh` associated with a `GameObject`.

Inheritance

↳ object
  ↳ MeshRenderer

Implements

IRenderer

Inherited Members

object.Equals(object)
object.Equals(object, object)
object.GetHashCode()
object.GetType()
object.MemberwiseClone()
object.ReferenceEquals(object, object)
object.ToString()

Namespace: RayGame

Assembly: RayGame.dll

Syntax

```
public class MeshRenderer : IRenderer
```

Properties

Container

The Container is the reference to the `GameObject` it is connected to.

Declaration

```
public GameObject Container { get; set; }
```

Property Value

TYPE
GameObject

Methods

GetMesh()

Gets the mesh being rendered.

Declaration

```
public Mesh GetMesh()
```

Returns

TYPE	DESCRIPTION
Mesh	The `Mesh` being rendered.

SetMesh(Mesh)

Sets the mesh to be rendered.

Declaration

```
public Mesh SetMesh(Mesh mesh)
```

Parameters

TYPE	NAME	DESCRIPTION
Mesh	`mesh`	The `Mesh` to set.

Returns

TYPE	DESCRIPTION
Mesh	The `Mesh` that was set.

Start()

Initializes the renderer. This method is called when the renderer is first added to a `GameObject` .

Declaration

```
public void Start()
```

Update()

Updates the renderer. This method is called once per frame.

Declaration

```
public void Update()
```

Implements

IRenderer

Edit this page

Class SpriteRenderer

Inheritance

↳ object
  ↳ SpriteRenderer

Implements

IRenderer

Inherited Members

object.Equals(object)
object.Equals(object, object)
object.GetHashCode()
object.GetType()
object.MemberwiseClone()
object.ReferenceEquals(object, object)
object.ToString()

Namespace: RayGame

Assembly: RayGame.dll

Syntax

```
public class SpriteRenderer : IRenderer
```

Fields

transform

Declaration

```
public Transform transform
```

Field Value

TYPE
Transform

Properties

Container

The Container is the reference to the `GameObject` it is connected to.

Declaration

```
public GameObject Container { get; set; }
```

Property Value

TYPE
GameObject

Methods

GetSprite()

Retrieves the current sprite texture.

Declaration

```
public Texture2D GetSprite()
```

Returns

TYPE	DESCRIPTION
Texture2D	The current `Raylib_cs.Texture2D` sprite.

SetSprite(Texture2D)

Sets the sprite texture.

Declaration

```
public void SetSprite(Texture2D ImputSprite)
```

Parameters

TYPE	NAME
Texture2D	ImputSprite

Start()

Initializes the renderer. This method is called when the renderer is first added to a `GameObject` .

Declaration

```
public void Start()
```

Update()

Updates the renderer. This method is called once per frame.

Declaration

```
public void Update()
```

Implements

IRenderer

Edit this page

Class Transform

The class that holds all the transformation data for an object

Inheritance

↳ object
  ↳ Transform

Inherited Members

object.Equals(object)
object.Equals(object, object)
object.GetHashCode()
object.GetType()
object.MemberwiseClone()
object.ReferenceEquals(object, object)
object.ToString()

Namespace: RayGame

Assembly: RayGame.dll

Syntax

```
public class Transform
```

Remarks

A Class that is used to represent the Position, Rotation and Scale of an object. This class super-imposes its properties onto vertices per frame, which means that any object containing this class can enact global transformations within that `GameObject`. By default, any transformations does on the object should be done through its Transform.

Constructors

Transform()

Initializes a new instance of the `Transform` class with default values. Position being (0,0). Rotation being 0. Scale being 1.

Declaration

```
public Transform()
```

Transform(Vector2, float, float)

Initializes a new instance of the `Transform` class with specified position, rotation, and scale.

Declaration

```
public Transform(Vector2 pos, float ang, float sc)
```

Parameters

TYPE	NAME	DESCRIPTION
Vector2	`pos`	The position of the transform.
float	`ang`	The rotation of the transform in degrees.
float	`sc`	The scale of the transform.

Fields

Position

The position of the transform as a `Vector2`.

Declaration

```
public Vector2 Position
```

Field Value

TYPE
Vector2

Scale

The Scale of the transform as a Float.

Declaration

```
public float Scale
```

Field Value

TYPE
float

Methods

ApplyTransform(Vector2[])

Applies the transform to an array of vertices. Applies that transforms onto the Vertices of a `Mesh` 's Vertex Array. Primarily used internally in the Engine.

Declaration

```
public Vector2[] ApplyTransform(Vector2[] VertexArray)
```

Parameters

TYPE	NAME	DESCRIPTION
Vector2[]	`VertexArray`	The array of vertices to transform.

Returns

TYPE	DESCRIPTION
Vector2[]	The transformed array of vertices.

GetRotation()

Gets the rotation of the transform.

Declaration

```
public float GetRotation()
```

Returns

TYPE	DESCRIPTION
float	The rotation in returned as Degrees.

Rotate(float)

Rotates the transform by the specified angle. Takes an Angle in degrees, and stores them internally as radians.

Declaration

```
public void Rotate(float Angle)
```

Parameters

TYPE	NAME	DESCRIPTION
float	`Angle`	The angle in degrees.

SetRotation(float)

Sets the rotation of the transform to the specified angle. Takes an Angle in degrees, and stores them internally as radians.

Declaration

```
public void SetRotation(float Angle)
```

Parameters

TYPE	NAME	DESCRIPTION
float	`Angle`	The angle in degrees.

Translate(Vector2)

Translates the transform by the specified offset.

Declaration

```
public void Translate(Vector2 Offset)
```

Parameters

TYPE	NAME	DESCRIPTION
Vector2	`Offset`	The offset as a `Vector2`.

Translate((float, float))

Translates the transform by the specified offset.

Declaration

```
public void Translate((float, float) Offset)
```

Parameters

TYPE	NAME	DESCRIPTION
(float, float)	`Offset`	The offset as a tuple(`float`, `float`)

Namespace RayGame.Demo

Classes

Bird

Demo

Manager

Edit this page

Class Bird

Inheritance

↳ object
  ↳ Bird

Implements

IGameComponent

Inherited Members

object.Equals(object)
object.Equals(object, object)
object.GetHashCode()
object.GetType()
object.MemberwiseClone()
object.ReferenceEquals(object, object)
object.ToString()

Namespace: RayGame.Demo

Assembly: RayGame.dll

Syntax

```
public class Bird : IGameComponent
```

Properties

Container

The container GameObject for this component.

Declaration

```
public GameObject Container { get; set; }
```

Property Value

TYPE
GameObject

Methods

Start()

Called when the component is Added.

Declaration

```
public void Start()
```

Update()

Called every frame to update the component.

Declaration

```
public void Update()
```

Implements

IGameComponent

Edit this page

Class Demo

Inheritance

↳ object
  ↳ Demo

Inherited Members

object.Equals(object)
object.Equals(object, object)
object.GetHashCode()
object.GetType()
object.MemberwiseClone()
object.ReferenceEquals(object, object)
object.ToString()

Namespace: RayGame.Demo

Assembly: RayGame.dll

Syntax

```
public static class Demo
```

Methods

Main()

Declaration

```
public static void Main()
```

Edit this page

Class Manager

Inheritance

↳ object
  ↳ Manager

Implements

IGameComponent

Inherited Members

object.Equals(object)
object.Equals(object, object)
object.GetHashCode()
object.GetType()
object.MemberwiseClone()
object.ReferenceEquals(object, object)
object.ToString()

Namespace: RayGame.Demo

Assembly: RayGame.dll

Syntax

```
public class Manager : IGameComponent
```

Fields

PipeInstances

Declaration

```
public List<GameObject> PipeInstances
```

Field Value

TYPE

List<GameObject>

Running

Declaration

```
public bool Running
```

Field Value

TYPE

bool

Properties

Container

The container GameObject for this component.

Declaration

```
public GameObject Container { get; set; }
```

Property Value

TYPE

GameObject

Instance

Declaration

```
public static Manager Instance { get; }
```

Property Value

TYPE

Manager

Methods

RestartGame()

Declaration

```
public void RestartGame()
```

SpawnObject()

Declaration

```
public void SpawnObject()
```

Start()

Called when the component is Added.

Declaration

```
public void Start()
```

Update()

Called every frame to update the component.

Declaration

```
public void Update()
```

Implements

IGameComponent