

# 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](#).

### [Transform](#)

The class that holds all the transformation data for an object

## Interfaces

### [IGameComponent](#)

Interface for game components.

### [IRenderer](#)

Interface for renderers.

# Class CollisionDetection

Namespace: [RayGame](#)

Assembly: RayGame.dll

Provides a method for collision detection between shapes.

```
public static class CollisionDetection
```

## Inheritance

[object](#)  ← CollisionDetection

## Inherited Members

[object.Equals\(object\)](#)  , [object.Equals\(object, object\)](#)  , [object.GetHashCode\(\)](#)  , [object.GetType\(\)](#)  , [object.MemberwiseClone\(\)](#)  , [object.ReferenceEquals\(object, object\)](#)  , [object.ToString\(\)](#) 

## Remarks

This is a Static class used internally to check the collision between Colliders in [GameObjects](#). **Primarily for internal use**

## Methods

### CheckCollision(Vector2[], Vector2[])

Checks if two shapes are colliding.

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

## Parameters

**shape1** [Vector2](#)  []

The first shape.

**shape2** [Vector2](#)  []

The second shape.

## Returns

[bool](#) 

True if the shapes are colliding, otherwise false.

# Class Engine

Namespace: [RayGame](#)

Assembly: RayGame.dll

The main engine class that initializes and runs the game.

```
public static class Engine
```

## Inheritance

[object](#)  ← Engine

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#) 

# Fields

## random

The Random Number Generator initiated with the Engine.

```
public static Random random
```

## Field Value

[Random](#) 

# Properties

## GAMETIME

The value associated with the amount of milliseconds passed.

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

Property Value

[long](#)

## Methods

### CreateGameObject(string)

Creates a new game object with the specified name.

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

#### Parameters

name [string](#)

The name of the game object.

#### Returns

[GameObject](#)

The created game object.

### CreateGameObject(string, Transform)

Creates a new game object with the specified name and [Transform](#).

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

#### Parameters

name [string](#)

The name of the game object.

transform [Transform](#)

The transform of the game object.

## Returns

### [GameObject](#)

The created game object.

## CreateGameObject(string, Vector2, float, float)

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

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

## Parameters

**name** [string](#)

The name of the game object.

**Position** [Vector2](#)

The position of the game object.

**Angle** [float](#)

The angle of the game object.

**Scale** [float](#)

The scale of the game object.

## Returns

### [GameObject](#)

The created game object.

## DeleteGameObject(GameObject)

Deletes the specified game object.

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

## Parameters

Gobj [GameObject](#)

The game object to delete.

## DisableColliderRendering()

Disables rendering of colliders for all game objects. Look at [GameObject](#) for more detail.

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

## EnableColliderRendering()

Enables rendering of colliders for all game objects. Look at [GameObject](#) for more detail.

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

## FindObjectByName(string)

Finds a game object by its name.

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

## Parameters

name [string](#)<sup>↗</sup>

The name of the game object.

## Returns

[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.

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

Returns

[GameObject](#)

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

Type Parameters

**T**

The type of component to look for.

## GetGameObjectCount()

Gets the count of game objects currently in the engine.

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

Returns

[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.



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

## Type Parameters

**T**

The type of game component to initialize.

# Class GameObject

Namespace: [RayGame](#)

Assembly: RayGame.dll








Represents a game object in the Scene.

```
public class GameObject
```

## Inheritance

[object](#)  ← GameObject

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#) 

## 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.

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

Field Value

[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.

```
public bool DEBUGCOLIDERS
```

Field Value

[bool](#)

## Name

This is the Name Associated with the GameObject

```
public string Name
```

Field Value

[string](#)

## Transform

This is the [Transform](#) attached to the GameObject.

```
public Transform Transform
```

Field Value

[Transform](#)

## Methods

### AddComponent<T>()

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

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

## Returns

T

The added component.

## Type Parameters

T

The type of component to add.

## AddRenderer<T>()

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

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

## Returns

T

The added renderer.

## Type Parameters

T

The type of renderer to add.

## DeleteAllRenderers()

Deletes all renderers from this game object.

```
public void DeleteAllRenderers()
```

## DeleteComponent<T>()

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

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

## Type Parameters

**T**

The type of component to delete.

## DeleteObjectComponents()

Deletes all components from this game object.

```
public void DeleteObjectComponents()
```

## DeleteRenderer<T>(int)

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

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

## Parameters

**Index** [int](#)

The index of the renderer to delete.

## Type Parameters

**T**

The type of renderer to delete.

## GetComponentNameList(bool)

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

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

## Parameters

Print [bool](#)

If true, prints the names to the console.

## Returns

[List](#) <[string](#)>

A list of component names.

## GetComponent<T>()

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

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

## Returns

T

The component if found, otherwise null.

## Type Parameters

T

The type of component to get.

## GetRendererNameList(bool)

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

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

## Parameters

**Print** [bool](#)

If true, prints the names to the console.

## Returns

[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

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

## Parameters

**Index** [int](#)

The index of the renderer to get.

## Returns

T

The renderer if found, otherwise null.

## Type Parameters

T

The type of renderer to get.

## HasComponent<T>()

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

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

## Returns

[bool](#)

True if the component is found, otherwise false.

## Type Parameters

**T**

The type of component to check for.

## HasRenderer<T>()

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

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

## Returns

[bool](#)

True if the renderer is found, otherwise false.

## Type Parameters

**T**

The type of renderer to check for.

## IsColliding(GameObject)

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

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



## Parameters

Target [GameObject](#)

The target game object to check for collisions with.

## Returns

[bool](#)

True if a collision is detected, otherwise false.

## SetTransform(Transform)

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

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

## Parameters

newTransform [Transform](#)

The new transform to set.

## ShiftComponent<T>(int)

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

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

## Parameters

offset [int](#)

The offset by which to shift the component.

## Type Parameters

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.

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

### Parameters

offset [int](#)

The offset by which to shift the renderer.

### Type Parameters

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**

```
public void StartActions()
```

## UpdateActions()

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

```
public void UpdateActions()
```

# Interface IGameComponent

Namespace: [RayGame](#)

Assembly: RayGame.dll

Interface for game components.

```
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* by creating 1 class component that adds all the required components and modifications.

## Properties

### Container

The container GameObject for this component.

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

Property Value

[GameObject](#)

## Methods

### Start()

Called when the component is Added.

```
void Start()
```

# Update()

Called every frame to update the component.

```
void Update()
```

# Interface IRenderer

Namespace: [RayGame](#)

Assembly: RayGame.dll

Interface for renderers.

```
public interface IRenderer
```

## Remarks

This interface is to be inherited by all Renderers. If you wish to make a custom renderer, then you 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 done through this reference.

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

Property Value

[GameObject](#)

## Methods

### Start()

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

```
void Start()
```

# Update()

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

```
void Update()
```

# Class Mesh

Namespace: [RayGame](#)

Assembly: RayGame.dll

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

```
public class Mesh
```

## Inheritance

[object](#)  ← Mesh

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#) 

## 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](#) .

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

## Parameters

**vertexArray** [Vector2](#)  []

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.

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

### Parameters

**vertexArray** ([float](#), [float](#))[[]]

An array of tuples representing the vertices of the mesh.

## Methods

### AddVertex(Vector2)

Adds a single vertex to the mesh.

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

### Parameters

**vertex** [Vector2](#)

The vertex to add to the mesh.

### AddVertex((float, float))

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

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

### Parameters

**vertex** ([float](#), [float](#))

The vertex to add to the mesh.



## AddVertices(Vector2[])

Adds an array of vertices to the mesh.

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

### Parameters

**vertexArray** [Vector2](#)[]

An array of [Vector2](#) to add to the mesh.

## AddVertices((float, float)[])

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

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

### Parameters

**vertexArray** ([float](#), [float](#))[]

An array of tuples representing the vertices to add to the mesh.

## DeleteLastVertex()

Deletes a Vertex from the last position.

```
public void DeleteLastVertex()
```

## DeleteVertex(int)

Deletes a vertex at the specified index.

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

## Parameters

Index [int](#)

The index of the vertex to delete.

## DeleteVertex((float, float))

Deletes a vertex that matches the specified point.

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

## Parameters

point ([float](#), [float](#))

The point representing the vertex to delete.

## GetVertexArray()

Gets the vertices of the mesh as an array.

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

## Returns

[Vector2](#)[]

An array of [Vector2](#) representing the vertices of the mesh.

## InsertVertex(int, Vector2)

Inserts a vertex at the specified index.

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

## Parameters

Index [int](#)

The index at which to insert the vertex.

Vertex [Vector2](#)

The vertex to insert.

## RotateMesh(float)

Rotates the entire mesh by the specified angle.

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

### Parameters

Angle [float](#)

The angle in degrees by which to rotate the mesh.

### Returns

[Mesh](#)

The rotated mesh.

## ScaleMesh(float)

Scales the entire mesh by the specified scale factor.

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

### Parameters

Scale [float](#)

The scale factor by which to scale the mesh.

### Returns

## [Mesh](#)

The scaled mesh.

## ShiftMesh(Vector2)

Shifts the entire mesh by the specified offset in Position.

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

### Parameters

**Offset** [Vector2](#)

The offset by which to shift the mesh.

### Returns

## [Mesh](#)

The shifted mesh.

## ShiftMesh((float, float))

Shifts the entire mesh by the specified offset in Position.

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

### Parameters

**Offset** ([float](#), [float](#))

The offset specified as a tuple by which to shift the mesh.

### Returns

## [Mesh](#)

The shifted mesh.

# Class MeshRenderer

Namespace: [RayGame](#)

Assembly: RayGame.dll

A Renderer that renders a [Mesh](#) associated with a [GameObject](#).

```
public class MeshRenderer : IRenderer
```

## 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\(\)](#)

# Properties

## Container

The Container is the reference to the [GameObject](#) it is connected to.

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

## Property Value

[GameObject](#)

# Methods

## GetMesh()

Gets the mesh being rendered.

```
public Mesh GetMesh()
```

Returns

[Mesh](#)

The [Mesh](#) being rendered.

## RenderMesh(Vector2[], Color)

Renders the specified vertices as lines with the given color. **Primarily for internal use**

```
public static void RenderMesh(Vector2[] Vertices, Color color)
```

Parameters

**Vertices** [Vector2](#) []

An array of [Vector2](#) representing the vertices of the mesh.

**color** Color

The color to use for rendering the lines.

## SetMesh(Mesh)

Sets the mesh to be rendered.

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

Parameters

**mesh** [Mesh](#)

The [Mesh](#) to set.

Returns

[Mesh](#)

The [Mesh](#) that was set.

## Start()

Initializes the renderer. This method is called when the renderer is first added to a [GameObject](#).

```
public void Start()
```

## Update()

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

```
public void Update()
```

# Class Transform

Namespace: [RayGame](#)

Assembly: RayGame.dll

The class that holds all the transformation data for an object

```
public class Transform
```

## Inheritance

[object](#)  ← Transform

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#) 

## 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.

```
public Transform()
```

### Transform(Vector2, float, float)

Initializes a new instance of the [Transform](#) class with specified position, rotation, and scale.

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



## Parameters

**pos** [Vector2](#)

The position of the transform.

**ang** [float](#)

The rotation of the transform in degrees.

**sc** [float](#)

The scale of the transform.

## Fields

### Position

The position of the transform as a [Vector2](#).

```
public Vector2 Position
```

#### Field Value

[Vector2](#)

### Scale

The Scale of the transform as a Float.

```
public float Scale
```

#### Field Value

[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.

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

## Parameters

**VertexArray** [Vector2](#)[]

The array of vertices to transform.

## Returns

[Vector2](#)[]

The transformed array of vertices.

# GetRotation()

Gets the rotation of the transform.

```
public float GetRotation()
```

## Returns

[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.

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

## Parameters

Angle [float](#)

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.

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

## Parameters

Angle [float](#)

The angle in degrees.

## Translate(Vector2)

Translates the transform by the specified offset.

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

## Parameters

Offset [Vector2](#)

The offset as a [Vector2](#).

## Translate((float, float))

Translates the transform by the specified offset.

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

## Parameters

**Offset** ([float](#), [float](#))

The offset as a tuple([float](#), [float](#))

# Namespace RayGame.Demo

## Classes

[Bird](#)

[Demo](#)

[Manager](#)

# Class Bird

Namespace: [RayGame.Demo](#)

Assembly: RayGame.dll

```
public class Bird : IGameComponent
```








## 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\(\)](#) 

# Properties

## Container

The container GameObject for this component.

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

## Property Value

[GameObject](#)

# Methods

## Start()

Called when the component is Added.

```
public void Start()
```

# Update()

Called every frame to update the component.

```
public void Update()
```

# Class Demo

Namespace: [RayGame.Demo](#)








Assembly: RayGame.dll

```
public static 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\(\)](#) 

## Methods

### Main()

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



# Class Manager

Namespace: [RayGame.Demo](#)

Assembly: RayGame.dll

```
public class Manager : IGameComponent
```








## 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\(\)](#) 

## Fields

### PipeInstances

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

### Field Value

[List](#)  <[GameObject](#)>

### Running

```
public bool Running
```

### Field Value

[bool](#) 

# Properties

## Container

The container GameObject for this component.

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

Property Value

[GameObject](#)

## Instance

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

Property Value

[Manager](#)

# Methods

## RestartGame()

```
public void RestartGame()
```

## SpawnObject()

```
public void SpawnObject()
```

## Start()

Called when the component is Added.

```
public void Start()
```

## Update()

Called every frame to update the component.

```
public void Update()
```