- GameManager Class
  - Overview
  - Properties
  - Fields
  - Methods
- LevelManager
  - Overview
  - Properties
  - Fields
  - Methods
- MovementController
  - Overview
  - Properties
  - Methods
- InputController
  - Overview
  - Methods
    - Start
- PrizeManager
  - Overview
  - Properties
  - Fields
  - Methods
- ObstacleManager Class
  - Overview
  - Fields
    - m_obstacles
  - Methods
    - Initialize
    - EnableObstaclesGravity
    - HitToObstacle
- CoinManager Class
  - Overview
  - Properties
  - Fields
  - Methods
- CutManager Class
  - Overview
  - Properties
  - Fields
  - Methods
- Level Class
  - Overview
  - Properties
  - Fields
  - Methods

- LevelPrizeSelector
  - Overview
  - Properties
  - Fields
  - Methods
- GameManagerData Class
  - Overview
  - Properties
  - Fields
  - Methods
- SurfaceData Struct
  - Overview
  - Properties
    - Color
    - SpiralVelocity
    - SpiralYScale
  - Fields
  - Methods
- Events Class
  - Overview
  - Fields
    - onStartScrapEvent
    - onStopScrapEvent
- Prize Class
  - Overview
  - Properties
  - Fields
  - Methods
- Intersections Class
  - Overview
  - Fields
  - Methods
    - BoundPlaneIntersect
    - Intersect
    - TrianglePlaneIntersect
- MeshCutter
  - Overview
  - Properties
  - Fields
  - Methods
- MeshUtils Class
  - Overview
  - Methods
    - FindCenter(List pairs)
    - ReorderList(List pairs)
    - SwitchPairs(List pairs, int pos1, int pos2)

- TempMesh Class
  - Overview
  - Properties
  - Fields
  - Methods
- Coin Class
  - Overview
  - Properties
  - Fields
  - Methods
- DestroyableObject
  - Overview
  - Properties
  - Fields
  - Methods
- Obstacle Class
  - Overview
  - Fields
  - Methods
    - Initialize(ObstacleManager manager)
    - EnableGravity()
    - OnCollisionEnter(Collision other)
- ReduceSize Class
  - Overview
  - Methods
    - OnTriggerStay(Collider other)
- ScraperableWood Class
  - Overview
  - Properties
  - Fields
  - Methods
- ScraperTool Class
  - Overview
  - Properties
  - Fields
  - Methods
- Spiral Class
  - Overview
  - Properties
  - Fields
  - Methods

GameManager Class

Overview

The `GameManager` class is a singleton MonoBehaviour that manages the game state and coordinates between different game components. It initializes the game, manages game start and end states, handles interactions with obstacles and scraperable objects, and controls the generation and stopping of spirals.

Properties

- `Instance`: A static property that returns the singleton instance of the `GameManager` class. If the instance is null, it finds the `GameManager` in the scene.
- `GameStarted`: A static boolean property that indicates whether the game has started.

Fields

- `m_events`: A private serialized field of type `Events` that holds the game events.
- `m_cutManager`, `m_movementController`, `m_coinManager`, `m_prizeManager`, `m_levelManager`, `m_obstacleManager`: Private serialized fields that reference the managers for different game components.
- `m_gameManagerData`: A private serialized field that holds the game manager data.
- `hitToScraperableObject`: A private serialized boolean field that indicates whether an object that can be scraped has been hit.
- `m_surfaceData`: A private serialized field of type `SurfaceData` that holds the data of the surface that has been hit.

Methods

- `Start()`: A private method that initializes the game.
- `Initialize(bool useTween)`: A public method that initializes the game and its components.
- `HitToScraperableObject(SurfaceData surfaceData)`: A public method that sets the hit object's surface data and sets `hitToScraperableObject` to true.
- `StartScraping()`: A public method that invokes the start scrap event.
- `GenerateSpiral()`: A public method that generates a spiral if `hitToScraperableObject` is true.
- `StopScraping()`: A public method that invokes the stop scrap event and stops generating a spiral if `hitToScraperableObject` is true.
- `LevelIsReady()`: A public method that enables the gravity of obstacles.
- `LevelIsFinished()`: A public method that sets `GameStarted` to false and initializes the next level if it exists.
- `HitToObstacle()`: A public method that sets `GameStarted` to false, stops scraping, and triggers the hit to obstacle event.

LevelManager

Overview

The `LevelManager` class is a MonoBehaviour that manages the levels in a game. It keeps track of the current level, the index of the current level, and a list of all level prefabs. It provides methods to initialize, hide, and show levels, as well as a method to handle the event when a level is finished.

Properties

- `public bool HaveNextLevel`: This property checks if there is a next level available by comparing the current level index with the count of level prefabs.
- `public Transform ScrapperStartTransform`: This property returns the start position of the scrapper in the current level.

Fields

- `[SerializeField] private List<Level> m_levelsPrefab`: This field stores a list of all level prefabs.
- `[SerializeField] private Level m_currentLevel`: This field stores the current level.
- `[SerializeField] private int m_levelIndex`: This field stores the index of the current level.

Methods

- `public void Initialize()`: This method initializes the level manager by setting the current level index from player preferences, instantiating the current level, initializing it, and showing it.
- `public void HideCurrentLevel()`: This method hides the current level.
- `public void ShowNextLevel()`: This method shows the next level if it exists, by instantiating it, initializing it, and showing it.
- `public void LevelIsFinished()`: This method is called when a level is finished. It calls the `LevelIsFinished` method of the `GameManager` instance and increments the level index if there is a next level.

MovementController

Overview

The `MovementController` class is a `MonoBehaviour` script in Unity that controls the movement of a scraper tool in a game. It provides functionality to initialize the tool, update its position, start and stop scraping, generate a spiral, and handle hitting an obstacle. The movement of the scraper tool can be controlled with or without tween animation.

Properties

- `m_scraperTool`: A `ScraperTool` object that the script controls.

- `m_moveSpeed`: A `float` that determines the speed at which the scraper tool moves.
- `scraperUpPosition`: A `Transform` that represents the position of the scraper when it is up.
- `scraperDownPosition`: A `Transform` that represents the position of the scraper when it is down.

Methods

- `Initialize(float moveSpeed,Transform scrapperStartTransform,bool useTweenAnimation)`: Initializes the scraper tool, sets its move speed, and positions it at the start transform. If `useTweenAnimation` is true, the tool's position is animated to the start transform; otherwise, it is set directly.
- `Update()`: Updates the position of the scraper tool if the game has started.
- `GenerateSpiral(SurfaceData surfaceData)`: Starts the scraper tool scraping the provided surface data.
- `StopGeneratingSpiral()`: Stops the scraper tool from scraping.
- `StartScraping()`: Starts the scraper tool scraping, moving it to the down position with a tween animation.
- `StopScraping()`: Stops the scraper tool from scraping, moving it to the up position with a tween animation.
- `HitToObstacle()`: Triggers the scraper tool's destroyable state when it hits an obstacle.

InputController

Overview

The `InputController` class is a MonoBehaviour that is used to handle mouse input in a Unity game. It uses the UniRx library to create Observables that react to mouse button events.

The class contains a `Start` method which is called before the first frame update. This method sets up Observables that listen for the mouse button being pressed and released, and perform actions based on these events.

Methods

Start

This method is called before the first frame update. It initializes two boolean variables, `mouseDown` and `startScrabing`, and sets up three Observables:

- The first Observable listens for the mouse button being pressed (Input.GetMouseButton(0)). When this event occurs, it sets `mouseDown` to true.
- The second Observable listens for the mouse button being released (Input.GetMouseButtonUp(0)). When this event occurs, it sets `mouseDown` to false.
- The third Observable samples the frame every 4 frames and performs actions based on the state of `mouseDown` and `startScrabing`. If the mouse button is down and scraping has not started, it calls `GameManager.Instance.StartScraping()` and sets `startScrabing` to true. If the mouse button is not down and scraping has started, it calls `GameManager.Instance.StopScraping()` and sets `startScrabing` to false. When the Observable is destroyed, it logs "destroy".

PrizeManager

Overview

The `PrizeManager` class is a MonoBehaviour that manages the state of all `Prize` objects in the scene. It provides methods to initialize all prizes and disable all other prizes when a specific prize is hit.

Properties

There are no public properties in this class.

Fields

- `m_prizes`: A private serialized field of type `List<Prize>`. This list contains all the `Prize` objects in the scene.

Methods

- `Initialize()`: This method initializes the `m_prizes` list with all the `Prize` objects found in the scene. It also calls the `Initialize()` method of each `Prize` object, passing `this` as an argument.
- `DisableOtherPrizes(Prize hittedPrize)`: This method disables the `BoxCollider` of all `Prize` objects in the `m_prizes` list except for the `Prize` object passed as an argument. This is done by finding all `Prize` objects in the `m_prizes` list that are not equal to `hittedPrize` and disabling their `BoxCollider`.

ObstacleManager Class

Overview

The `ObstacleManager` class is a Unity MonoBehaviour that manages the obstacles in the game. It is responsible for initializing the obstacles, enabling their gravity, and handling the event when a player hits an obstacle.

Fields

m_obstacles

```
[SerializeField] private List<Obstacle> m_obstacles;
```

A serialized private field that holds a list of `Obstacle` objects. This list is populated in the `Initialize` method by finding all objects of type `Obstacle` in the scene.

Methods

Initialize

```
public void Initialize()
```

This method initializes the `m_obstacles` list by finding all `Obstacle` objects in the scene. It then calls the `Initialize` method on each `Obstacle`, passing `this` as the argument.

EnableObstaclesGravity

```
public void EnableObstaclesGravity()
```

This method enables the gravity for each `Obstacle` in the `m_obstacles` list by calling the `EnableGravity` method on each `Obstacle`.

HitToObstacle

```
public void HitToObstacle()
```

This method is called when a player hits an obstacle. It calls the `HitToObstacle` method on the `GameManager` instance.

CoinManager Class

Overview

The `CoinManager` class is a Unity MonoBehaviour that manages the behavior of Coin objects in the game. It is responsible for initializing the coins and their animations, as well as handling the collection of coins.

Properties

There are no public properties in this class.

Fields

- `m_coins`: A serialized private list of Coin objects. This list is populated in the `Initialize` method with all the Coin objects present in the scene.
- `m_tweens`: A serialized private list of Tween objects. This list is used to store the rotation animations for each Coin object.

Methods

- `Initialize()`: This method initializes the `m_coins` list with all the Coin objects present in the scene. It also creates a rotation animation for each coin and stores it in the `m_tweens` list. Each coin is also initialized with its rotation animation and a reference to the `CoinManager`.

- `CollectCoin(Coin coin)`: This method is responsible for the behavior when a coin is collected. It scales the coin down to zero over half a second, and once that animation is complete, it stops the rotation animation and destroys the coin object.

CutManager Class

Overview

The `CutManager` class is a Unity MonoBehaviour that manages the cutting or slicing of objects in a 3D game. It uses a plane to slice objects and separates the resulting parts. It also provides functionality to draw the slicing plane, initialize the cutting system, and stop the cutting process.

Properties

- `plane`: A GameObject representing the plane used for slicing.
- `ObjectContainer`: A Transform object that holds the sliced objects.
- `separation`: A float value that determines how far the resulting objects are separated after slicing.
- `drawPlane`: A boolean value that determines whether to draw the slicing plane or not.

Fields

- `m_ScraperableWoods`: A list of `ScraperableWood` objects that can be sliced.
- `m_lastHittedWood`: A `ScraperableWood` object that was last hit for slicing.
- `slicePlane`: A Plane object used for slicing.
- `meshCutter`: A `MeshCutter` object used to slice the mesh of the objects.
- `biggerMesh`, `smallerMesh`: `TempMesh` objects to hold the resulting meshes after slicing.

Methods

- `Initialize(float reduceScaleSpeed)`: Initializes the cutting system.
- `Cut(ScraperableWood scraperableWood,SurfaceData surfaceData,Transform parentTransform,Vector3 start, Vector3 end, Vector3 depth)`: Performs the cutting operation.
- `DrawPlane(Vector3 start, Vector3 end, Vector3 normalVec)`: Draws the slicing plane.
- `SliceObjects(GameObject gameObject,Vector3 point, Vector3 normal)`: Slices the given GameObject.
- `SliceObject(ref Plane slicePlane, GameObject obj, List<Transform> positiveObjects, List<Transform> negativeObjects)`: Slices a single GameObject.
- `ReplaceMesh(Mesh mesh, TempMesh tempMesh, MeshCollider collider = null)`: Replaces the mesh of a GameObject with a temporary mesh.
- `SeparateMeshes(Transform posTransform, Transform negTransform, Vector3 localPlaneNormal)`: Separates the resulting meshes after slicing.
- `SeparateMeshes(List<Transform> positives, List<Transform> negatives, Vector3 worldPlaneNormal)`: Separates a list of resulting meshes after slicing.
- `StopScraping()`: Stops the cutting process.

Level Class

Overview

The `Level` class is a Unity MonoBehaviour script that manages the behavior of a level in the game. It handles the initialization, showing, hiding, and finishing of a level. It uses the DOTween library for smooth movement animations.

Properties

- `ScrapperStartPosition`: This is a public read-only property that returns the private field `m_scrapperStartPosition`. It represents the start position of the scrapper in the level.

Fields

- `m_scrapperStartPosition`: This is a private serialized Transform field that represents the start position of the scrapper in the level.
- `m_levelHideTransformUnderWater`: This is a private serialized Transform field that represents the position where the level hides under water.
- `m_levelPrizeSelector`: This is a private serialized LevelPrizeSelector field that is used to select the prize for the level.
- `m_levelManager`: This is a private LevelManager field that manages the level.

Methods

- `Initialize(LevelManager levelManager)`: This method initializes the level with the given LevelManager. It sets the position of the level to the hide position under water and initializes the LevelPrizeSelector.
- `Show()`: This method shows the level by moving it to the y position of 0 in 3 seconds. When the movement is complete, it calls the `LevelIsReady` method of the GameManager instance.
- `Hide()`: This method hides the level by moving it to the y position of the hide position under water in 3 seconds.
- `LevelIsFinished()`: This method is called when the level is finished. It calls the `LevelIsFinished` method of the LevelManager.

LevelPrizeSelector

Overview

The `LevelPrizeSelector` class is a Unity MonoBehaviour script that is used to manage the level prizes in a game. It is designed to be attached to a GameObject in the Unity scene. The script listens for a collision event with an object tagged as "Scraper". When such a collision occurs, it triggers the end of the level by calling the `LevelIsFinished` method on the `Level` manager object.

Properties

There are no public properties in this class.

Fields

- `private Level m_manager;` - A private field that holds a reference to the `Level` manager object. This field is initialized via the `Initialize` method.

Methods

- `public void Initialize(Level manager)` - This method is used to initialize the `m_manager` field. It takes a `Level` object as a parameter.
- `private void OnTriggerEnter(Collider other)` - This method is a Unity callback that is triggered when the GameObject this script is attached to collides with another GameObject. If the other GameObject has a tag of "Scraper", it calls the `LevelIsFinished` method on the `m_manager` object.

GameManagerData Class

Overview

The `GameManagerData` class is a `ScriptableObject` in Unity that stores the move speed for a game object. This class is used to create a custom asset menu in Unity, allowing developers to easily adjust the move speed of game objects within the Unity editor.

Properties

- `MoveSpeed`: A public float that represents the speed at which a game object moves.

Fields

There are no additional fields in this class.

Methods

There are no methods in this class.

SurfaceData Struct

Overview

The `SurfaceData` struct is a serializable data structure in Unity that is used to store information about a surface. This includes its color, spiral velocity, and spiral Y scale. This struct is marked as serializable, which means it can be used in Unity's inspector and its values can be saved and loaded.

Properties

Color

```
public Color Color;
```

The `Color` property represents the color of the surface. It is of type `UnityEngine.Color`.

SpiralVelocity

```
public Vector3 SpiralVelocity;
```

The `SpiralVelocity` property represents the velocity of the spiral on the surface. It is of type `UnityEngine.Vector3`.

SpiralYScale

```
public float SpiralYScale;
```

The `SpiralYScale` property represents the Y scale of the spiral on the surface. It is of type `float`.

Fields

This struct does not have any fields.

Methods

This struct does not have any methods.

Events Class

Overview

The `Events` class in Unity is designed to handle custom events within the game. It uses delegates to define two types of events: `onStartScrapEvent` and `onStopScrapEvent`. These events can be used to trigger specific actions or sequences within the game when certain conditions are met.

Fields

onStartScrapEvent

This is a delegate of type `ScrapEvent`. It is triggered at the start of a scrap event.

onStopScrapEvent

This is a delegate of type `ScrapEvent`. It is triggered at the end of a scrap event.

Prize Class

Overview

The `Prize` class is a Unity MonoBehaviour that represents a prize in a game. It has a box collider and can interact with other game objects through Unity's physics system. The class also has methods for initializing the

prize with a `PrizeManager` and for handling Unity's `OnTriggerEnter` and `OnTriggerStay` events.

Properties

- `BoxCollider BoxCollider`: A public property that holds a reference to the BoxCollider component attached to the prize game object.

Fields

- `private float startTime`: A private field that stores the time when the prize first comes into contact with another game object tagged "Spiral".
- `private float endTime`: A private field that stores the time when the prize is still in contact with the "Spiral" tagged game object.
- `private PrizeManager _manager`: A private field that holds a reference to the `PrizeManager` that manages this prize.

Methods

- `public void Initialize(PrizeManager prizeManager)`: This method is used to initialize the prize with a `PrizeManager`. It takes a `PrizeManager` as a parameter and assigns it to the `_manager` field.
- `private void OnTriggerEnter(Collider other)`: This is a Unity event method that is called when the prize first comes into contact with another game object. If the other game object is tagged "Spiral", it sets the `startTime` to the current time.
- `private void OnTriggerStay(Collider other)`: This is a Unity event method that is called every frame where the prize is still in contact with another game object. If the other game object is tagged "Spiral" and the contact has lasted for more than 1 second, it calls the `DisableOtherPrizes` method on the `_manager` and moves the prize to the left.

Intersections Class

Overview

The `Intersections` class is a Unity C# script that provides methods for calculating intersections between different geometric objects such as planes, line segments, and triangles. It includes methods for determining if a bounding box intersects with a plane, finding the intersection point between a plane and a line segment, and determining if a triangle intersects with a plane.

Fields

- `v`: A private readonly field that holds an array of Vector3 objects.
- `u`: A private readonly field that holds an array of Vector2 objects.
- `t`: A private readonly field that holds an array of integers.
- `positive`: A private readonly field that holds an array of booleans.
- `edgeRay`: A private field of type Ray used in the `Intersect` method.

Methods

BoundPlaneIntersect

```
public static bool BoundPlaneIntersect(Mesh mesh, ref Plane plane)
```

This static method checks if a bounding box of a mesh intersects with a plane. It returns a boolean value indicating whether the intersection occurs.

Intersect

```
public ValueTuple<Vector3, Vector2> Intersect(Plane plane, Vector3 first, Vector3 second, Vector2 uv1, Vector2 uv2)
```

This method finds the intersection between a plane and a line segment defined by vectors `first` and `second`. It returns a tuple containing the intersection point as a Vector3 and the interpolated UV coordinates as a Vector2.

TrianglePlaneIntersect

```
public bool TrianglePlaneIntersect(List<Vector3> vertices, List<Vector2> uvs, List<int> triangles, int startIdx, ref Plane plane, TempMesh posMesh, TempMesh negMesh, Vector3[] intersectVectors)
```

This method checks if a triangle intersects with a plane. The triangle is defined by a list of vertices, UVs, and triangle indices. The method also takes in a starting index for the triangle, a reference to a plane, two TempMesh objects representing the positive and negative meshes, and an array of intersecting vectors. It returns a boolean value indicating whether the intersection occurs.

MeshCutter

Overview

The `MeshCutter` class is a utility for slicing a mesh object in Unity. It provides methods to slice a mesh by a plane, get the first vertex of the mesh, and fill the boundary of the sliced mesh. The class also maintains the positive and negative meshes resulting from the slicing operation.

Properties

- `PositiveMesh`: The mesh on the same side as the plane's normal after slicing.
- `NegativeMesh`: The mesh on the opposite side of the plane's normal after slicing.

Fields

- `addedPairs`: A list of Vector3 pairs added during the slicing operation.
- `ogVertices`, `ogTriangles`, `ogNormals`, `ogUvs`: Lists that store the original vertices, triangles, normals, and UVs of the mesh.

- `intersectPair`, `tempTriangle`: Arrays used during the slicing operation.
- `intersect`: An instance of the `Intersections` class used to determine if a triangle intersects with the slicing plane.
- `threshold`: A small float value used to determine if two vectors are colinear.

Methods

- `MeshCutter(int initialArraySize)`: Constructor that initializes the `MeshCutter` with an initial array size.
- `SliceMesh(Mesh mesh, ref Plane slice)`: Slices the given mesh by the given plane. Returns a boolean indicating if the slicing operation was successful.
- `GetFirstVertex()`: Returns the first vertex of the original mesh.
- `FillBoundaryGeneral(List<Vector3> added)`: Fills the boundary of the sliced mesh using a general method.
- `FillBoundaryFace(List<Vector3> added)`: Fills the boundary of the sliced mesh by creating a face.
- `FindRealPolygon(List<Vector3> pairs)`: Extracts a polygon from pairs of vertices.
- `AddTriangle(List<Vector3> face, int t1, int t2, int t3)`: Adds a triangle to the positive and negative meshes.

MeshUtils Class

Overview

The `MeshUtils` class is a static utility class in Unity, written in C#. It provides methods for manipulating and processing lists of Vector3 pairs, which can be used to represent polygons in 3D space. The class includes methods for finding the center of a polygon and for reordering a list of Vector3 pairs to form a closed polygon.

Methods

FindCenter(List pairs)

This method calculates and returns the center of a polygon represented by a list of Vector3 pairs. The center is calculated by averaging the vertices of the polygon.

Parameters:

- `pairs`: A list of Vector3 pairs representing the vertices of a polygon.

Returns:

- A Vector3 representing the center of the polygon.

ReorderList(List pairs)

This method reorders a list of Vector3 pairs so that they form a closed polygon. The method modifies the input list in-place.

Parameters:

- `pairs`: A list of Vector3 pairs to be reordered.

SwitchPairs(List pairs, int pos1, int pos2)

This is a private helper method used by the `ReorderList` method. It swaps two pairs of Vector3s in the input list.

Parameters:

- `pairs`: A list of Vector3 pairs.
- `pos1`: The position of the first pair to be swapped.
- `pos2`: The position of the second pair to be swapped.

TempMesh Class

Overview

The `TempMesh` class is a custom data structure used to represent a temporary mesh in Unity. It stores the vertices, normals, UVs, and triangles of the mesh, as well as a mapping of indices from the original mesh to the new mesh. It also calculates and stores the surface area of the mesh. The class provides methods to clear the mesh, add points, triangles, and vertices, check if the mesh contains certain keys, and calculate the area of a triangle.

Properties

- `vertices`: A list of Vector3 objects representing the vertices of the mesh.
- `normals`: A list of Vector3 objects representing the normals of the mesh.
- `uvs`: A list of Vector2 objects representing the UVs of the mesh.
- `triangles`: A list of integers representing the triangles of the mesh.
- `surfacearea`: A float representing the surface area of the mesh.

Fields

- `vMapping`: A dictionary mapping indices from the original mesh to the new mesh.

Methods

- `TempMesh(int vertexCapacity)`: Constructor that initializes the lists and dictionary with a given capacity and sets the surface area to 0.
- `Clear()`: Clears all the lists and the dictionary, and sets the surface area to 0.
- `AddPoint(Vector3 point, Vector3 normal, Vector2 uv)`: Adds a point and its normal to the respective lists.
- `AddOgTriangle(int[] indices)`: Adds triangles from the original mesh and computes the triangle area.
- `AddSlicedTriangle(int i1, Vector3 v2, Vector2 uv2, int i3)`: Adds a sliced triangle to the mesh and computes the triangle area.
- `AddSlicedTriangle(int i1, Vector3 v2, Vector3 v3, Vector2 uv2, Vector2 uv3)`: Adds a sliced triangle to the mesh and computes the triangle area.

- `AddTriangle(Vector3[] points)`: Adds a completely new triangle to the mesh and computes the triangle area.
- `ContainsKeys(List<int> triangles, int startIdx, bool[] isTrue)`: Checks if the mesh contains certain keys.
- `AddVertex(List<Vector3> ogVertices, List<Vector3> ogNormals, List<Vector2> ogUvs, int index)`: Adds a vertex from the original mesh while storing its old index in the dictionary of index mappings.
- `GetTriangleArea(int i)`: Calculates and returns the area of a triangle.

Coin Class

Overview

The `Coin` class is a Unity MonoBehaviour script that is used to manage the behavior of a coin in a game. It includes a reference to a `CoinManager` object, a `BoxCollider` and a `Tween` object from the DG.Tweening namespace. The `Coin` class is responsible for initializing the coin's rotation and managing its interaction with other game objects.

Properties

- `RotateTween`: This is a public property of type `Tween` from the DG.Tweening namespace. It is used to control the rotation animation of the coin.

Fields

- `m_manager`: This is a private field of type `CoinManager`. It is used to manage the behavior of the coin in the game.
- `m_collider`: This is a private serialized field of type `BoxCollider`. It is used to handle the coin's collision in the game.

Methods

- `Initialize(Tween rotateTween,CoinManager coinManager)`: This is a public method that takes a `Tween` object and a `CoinManager` object as parameters. It is used to initialize the `RotateTween` and `m_manager` fields.
- `OnTriggerEnter(Collider other)`: This is a private method that takes a `Collider` object as a parameter. It is called when the coin enters a trigger collider. It is used to collect the coin in the game.

DestroyableObject

Overview

The `DestroyableObject` class is a Unity script attached to a GameObject. This script is used to manage and activate the Rigidbody components of the GameObject and its children. It includes methods to initialize the list of Rigidbodies and to activate them with a random force.

Properties

There are no public properties in this script.

Fields

- `m_rigidbodies`: A private serialized field of type `List<Rigidbody>`. This list stores the Rigidbody components of the GameObject and its children.

Methods

- `Initialize()`: This public method is used to initialize the `m_rigidbodies` list. It gets all the Rigidbody components from the GameObject's children and adds them to the list.
- `ActiveRigidBodies()`: This public method is used to activate the GameObject and apply a random force to all the Rigidbodies in the `m_rigidbodies` list. The force is applied as an impulse.

Obstacle Class

Overview

The `Obstacle` class is a Unity MonoBehaviour script that is used to manage the behavior of obstacles in the game. It includes functionalities for initializing the obstacle, enabling gravity on the obstacle, and handling collision events with other game objects. The class is dependent on the `ObstacleManager` class for managing the state of the obstacle.

Fields

- `private ObstacleManager _manager`: An instance of the `ObstacleManager` class that manages the state of the obstacle.
- `private BoxCollider _boxCollider`: The `BoxCollider` component attached to the obstacle game object.
- `private Rigidbody _rigidbody`: The `Rigidbody` component attached to the obstacle game object.
- `[SerializeField] private bool m_isStaticObstacle`: A serialized private boolean field that determines if the obstacle is static or not.

Methods

Initialize(ObstacleManager manager)

This method is used to initialize the obstacle. It takes an instance of the `ObstacleManager` class as a parameter. It assigns the `BoxCollider` and `Rigidbody` components of the obstacle game object to the `_boxCollider` and `_rigidbody` fields respectively. It also assigns the `ObstacleManager` instance to the `_manager` field.

EnableGravity()

This method is used to enable gravity on the obstacle. It checks if the obstacle is not static by checking the `m_isStaticObstacle` field. If the obstacle is not static, it sets the `useGravity` property of the `Rigidbody` component to true.

OnCollisionEnter(Collision other)

This is a Unity callback method that is called when the obstacle collides with another game object. It checks if the other game object has a tag of "Scraper" or "Spiral". If the tag is "Scraper", it calls the `HitToObstacle` method of the `ObstacleManager` instance. If the tag is "Spiral" and the obstacle is not static, it disables the `BoxCollider` component and removes all constraints from the `Rigidbody` component.

ReduceSize Class

Overview

The `ReduceSize` class is a Unity script attached to a GameObject. This script is used to reduce the size of the parent GameObject when it collides with another GameObject tagged as "Scraper". The reduction in size is done along the Z-axis of the parent GameObject's local scale.

Methods

OnTriggerStay(Collider other)

This is a Unity-specific method that is called once per frame for every Collider `other` that is touching the trigger. In this script, it checks if the `other` collider has a tag of "Scraper". If it does, it reduces the local scale of the parent GameObject along the Z-axis by 0.01 units.

```
private void OnTriggerStay(Collider other)
{
    if(other.CompareTag("Scraper"))
        transform.parent.localScale -= Vector3.forward * .01f;
}
```

Parameters:

- `Collider other`: The other Collider involved in this collision.

Return Value: No return value.

Note: This method requires a Collider to be attached to the same GameObject as this script and the Collider's `isTrigger` property to be set to `true`.

ScraperableWood Class

Overview

The `ScraperableWood` class is a `MonoBehaviour` that represents a piece of wood that can be scraped or cut. It is used in conjunction with a `CutManager` to manage the cutting process. The class also uses a `BoxCollider` to detect collisions with other objects, specifically a "Scraper". When a collision with a "Scraper" is detected, the `CutManager` is called to cut the wood and the `ReduceSize` component is enabled to visually represent the reduction in size of the wood.

Properties

- `Collider`: A `BoxCollider` that is used to detect collisions with other objects.

Fields

- `_manager`: A `CutManager` that is used to manage the cutting process.
- `m_reduceScaleSpeed`: A `float` that determines the speed at which the wood's size is reduced when it is cut.
- `m_reduceSize`: A `ReduceSize` component that is used to visually represent the reduction in size of the wood.
- `m_data`: A `SurfaceData` object that contains information about the surface of the wood.

Methods

- `Initialize(float reduceScaleSpeed, CutManager cutManager)`: This method is used to initialize the `ScraperableWood` object. It sets the `reduceScaleSpeed` and `cutManager` fields.
- `OnCollisionEnter(Collision other)`: This method is called when the `ScraperableWood` object collides with another object. If the other object has a tag of "Scraper", the `CutManager` is called to cut the wood and the `ReduceSize` component is enabled.

ScraperTool Class

Overview

The `ScraperTool` class is a `MonoBehaviour` script in Unity that is used to control the behavior of a scraping tool in a game. The scraping tool is represented by a spiral object that can be initialized, started, stopped, and shown as destroyable. The class uses several serialized fields to reference the spiral object, its game object, a prefab for the spiral, a box collider, a main blade, and a destroyable blade. It also uses a private float field to store the start time of the scraping.

Properties

- `Spiral _spiral`: A reference to the Spiral object that the scraper tool controls.
- `GameObject spiralGameObject`: A reference to the game object of the Spiral.
- `GameObject _spiralPrefab`: A reference to the prefab used to instantiate the Spiral.
- `BoxCollider m_collider`: A reference to the BoxCollider component of the scraper tool.
- `GameObject m_mainBlade`: A reference to the main blade of the scraper tool.
- `DestroyableObject m_destroyableBlade`: A reference to the destroyable blade of the scraper tool.

Fields

- `private float m_startTime`: A private field used to store the start time of the scraping.

Methods

- `public void Initialize()`: Initializes the destroyable blade.

- `public void StartScraping(SurfaceData surfaceData)`: Starts the scraping process. If the Spiral object is null, it instantiates a new Spiral object, initializes it with the provided surface data, and starts the scraping.
- `public void StopScraping()`: Stops the scraping process. It enables the BoxCollider and stops the scraping of the Spiral object.
- `public void ShowDestroyable()`: Shows the destroyable blade and deactivates the main blade.

Spiral Class

Overview

The `Spiral` class is a `MonoBehaviour` that represents a spiral shape in a Unity scene. This class is responsible for creating and updating the mesh of the spiral, initializing its properties, and controlling its behavior such as stopping its scraping.

This class requires a `MeshFilter` component and can be executed in edit mode.

Properties

- `radius`: A float that determines the radius of the spiral. It can be set within the range of 0.01 to 1000.
- `width`: A float that determines the width of the spiral. It can be set within the range of 0.1 to 10.
- `height`: A float that determines the height of the spiral. It can be set within the range of 0.1 to 10.
- `length`: A public float that determines the length of the spiral. It can be set within the range of 0.01 to 1000.
- `sides`: An integer that determines the number of sides of the spiral. It can be set within the range of 4 to 720.
- `offset`: A float that determines the offset of the spiral. It can be set within the range of 0 to 100.
- `m_spiralVelocity`: A public Vector3 that determines the velocity of the spiral when the scraping stops.
- `torque`: A public float that determines the torque of the spiral.

Fields

- `mesh`: A Mesh object that represents the mesh of the spiral.
- `meshRenderer`: A private MeshRenderer that renders the mesh of the spiral.
- `radiusSurfaceVertices`: A private Vector3 array that stores the vertices of the radius surface of the spiral.
- `vertices`: A private Vector3 array that stores the vertices of the spiral.
- `triangles`: A private integer array that stores the triangles of the spiral.

Methods

- `Initialize(SurfaceData m_surfaceData)`: Initializes the spiral with the given surface data.
- `Refresh()`: Refreshes the spiral by creating and updating its mesh.
- `CreateMesh()`: Creates the mesh of the spiral.
- `UpdateMesh()`: Updates the mesh of the spiral.

- `StopScraping(float holdTime)`: Stops the scraping of the spiral and sets its velocity and torque based on the given hold time.