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.

PlayerController Class

Overview

The `PlayerController` class is a Unity MonoBehaviour script used for controlling a player in a game. It provides methods to initialize the player, add force to the player, rotate the player, move the player, and finish the level.

Properties

- `m_player` : This is a private serialized field of type `Player`. It represents the player that this controller is controlling.

Methods

- `Initialize(Transform playerStartTransform,LayerMask platformLayerMask)` : This method is used to initialize the player. It takes the start transform and platform layer mask as parameters and initializes the player at the given position and rotation, with the given platform layer mask.
- `AddForceToPlayer(float force)` : This method is used to add force to the player. It takes a float value representing the force to be added.

- `RotatePlayer(float angle)` : This method is used to rotate the player. It takes a float value representing the angle by which the player should be rotated.
- `MovePlayer(float value)` : This method is used to move the player. It takes a float value representing the distance by which the player should be moved.
- `LevelIsFinished()` : This method is called when the level is finished. It triggers the `FinishLevel` method of the player.

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.

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".

AudioManager Class

Overview

The `AudioManager` class is a MonoBehaviour that manages the audio in a Unity game. It includes methods for initializing the audio system and playing a specific sound effect when a hit to the platform occurs. The class uses Unity's `AudioClip` and `AudioSource` classes to handle audio data and playback.

Properties

- `m_hitToPlatform`: This is a serialized private field of type `AudioClip`. It represents the audio clip that is played when a hit to the platform occurs.
- `m_soundEffectAudioSource`: This is a serialized private field of type `AudioSource`. It is the audio source that plays the sound effects in the game.
- `m_backgroundMusicAudioSource`: This is a serialized private field of type `AudioSource`. It is the audio source that plays the background music in the game.

Methods

- `Initialize()`: This method initializes the audio system. It sets the global volume to 1 and loads the audio data for the hit to platform sound effect.

- `HitToPlatformSound(int levelNum)`: This method plays the hit to platform sound effect. It sets the clip of the sound effect audio source to the hit to platform audio clip and then plays it. The `levelNum` parameter is currently not used in the method.

CameraManager

Overview

The `CameraManager` class is a Unity script for managing camera movement in a game. It is responsible for setting the initial position and rotation of the camera, and updating the camera's position and rotation to follow the player during the game.

Properties

- `m_camera`: A private serialized field of type `Camera`. This is the main camera that the script will control.
- `m_startCameraPosition`: A private serialized field of type `Transform`. This is the reference to the starting position of the camera.
- `m_followPlayerCameraPosition`: A private serialized field of type `Transform`. This is the reference to the position of the player that the camera should follow.

Methods

- `Initialize()`: This public method sets the camera's position and rotation to the starting position. It is typically called at the start of the game or when the camera needs to be reset.
- `FixedUpdate()`: This private method is called every fixed framerate frame. If the game has started and the distance between the camera's current position and the player's position is greater than 0.01, it updates the camera's position and rotation to gradually move towards the player's position. The speed of the camera's movement is determined by the `Lerp` function and the `Time.deltaTime` value.

ParticleManager Class

Overview

The `ParticleManager` class is a MonoBehaviour class in Unity, used for managing particles. It contains a reference to the final particles and methods to initialize and play these particles.

Properties

There are no public properties in the `ParticleManager` class.

Fields

- `m_finishParticles`: A private serialized field of type `List<ParticleSystem>`. This field holds the reference to the final particles.

Methods

- `Initialize(List<ParticleSystem> particles)`: This public method is used to initialize the `m_finishParticles` field. It takes a list of `ParticleSystem` objects as a parameter.
- `PlayFinishParticles()`: This public method is used to play the final particles. It does not take any parameters and does not return any value. It plays each particle in the `m_finishParticles` list.

PlatformManager Class

Overview

The `PlatformManager` class is a Unity MonoBehaviour script used for managing the jump platforms in a game. It stores a list of platforms and the last platform in the list is considered as the final platform. The class also provides methods to initialize the platforms and handle the events when a player hits a platform or the final platform.

Properties

- `FinalPlatform`: This is a serialized field that gets the last platform from the platform list as the final platform.

Fields

- `m_platforms`: This is a private serialized field that stores the list of platforms.

Methods

- `Initialize(List<Platform> jumpPlatforms)`: This method is used to initialize the platforms. It takes a list of platforms as a parameter and assigns it to the `m_platforms` field. It also initializes each platform in the list with a level name counter and the current instance of the `PlatformManager` class.
- `HitToPlatform(int platformNum, Transform player, float force)`: This method is called when a player hits a platform. It notifies the game manager about the hit by calling the `HitToPlatform` method of the `GameManager` instance. It takes the platform number, player transform, and force as parameters.
- `HitToLastPlatform(int platformNum)`: This method is called when a player hits the final platform. It notifies the game manager that the level is finished by calling the `LevelIsFinished` method of the `GameManager` instance. It takes the platform number as a parameter.

UIManager Class

Overview

The `UIManager` class is a MonoBehaviour derived class used for managing UI elements in Unity. It is responsible for controlling the visibility and behavior of various UI elements such as the level progress bar, the "Perfect!" hit to platform text, and the "tap to start" text.

Fields

- `m_levelProgressUI`: A private serialized field of type `LevelProgressUI`. It is used as a reference to the level slider bar object in the UI.
- `m_hitToPlatformText`: A private serialized field of type `TextMeshProUGUI`. It is used as a reference to the "Perfect!" hit to platform text in the UI.
- `m_startGameText`: A private serialized field of type `TextMeshProUGUI`. It is used as a reference to the "tap to start" text mesh in the UI.

Methods

- `Initialize(int jumpPlatformsNum, int currentLevelNum, int nextLevelNum)`: This public method is used to initialize the UI elements. It takes three integer parameters - `jumpPlatformsNum`, `currentLevelNum`, and `nextLevelNum`. It sets the alpha of `m_hitToPlatformText` to 0 and calls the `Initialize` method of `m_levelProgressUI`.
- `GameStarted()`: This public method is used to fade out the "tap to start" text when the game starts.
- `HitToJumpPlatform(int jumpPlatformNum)`: This public method is used to show the hit to platform text with tweening. It takes an integer parameter `jumpPlatformNum`. If `jumpPlatformNum` is not -1, it calls the `SetLevelProgression` method of `m_levelProgressUI`. It also sets the alpha of `m_hitToPlatformText` to 0, scales it to `Vector3.one`, and then performs a sequence of tweening operations on it.

FinalPlatform Class

Overview

The `FinalPlatform` class is a derived class from the `Platform` base class. It represents the final platform in a game level. When a collision occurs on this platform, it triggers a victory event. The class also contains a list of particle systems that can be used to create a victory effect.

Properties

- `VictoryParticles`: A list of `ParticleSystem` objects. These can be used to create a victory effect when the player reaches this platform.

Methods

OnCollisionEnter(Collision other)

This method is called when a collision occurs. It calls the `HitToLastPlatform` method of the `Manager` object, passing the `PlatformNum` as an argument.

Initialize(int platformNum, PlatformManager platformManager)

This is an overridden method from the `Platform` base class. It initializes the `FinalPlatform` object with a platform number and a reference to the `PlatformManager` object. The `PlatformNum` and `Manager` fields are set to the provided arguments.

JumpPlatform Class

Overview

The `JumpPlatform` class is a subclass of the `Platform` class. It represents a jump platform in a game, which has various properties such as being destroyable, having a hit count, and having a UFO spaceship design. The class also includes methods for initializing the platform, starting hit animations, and handling the destruction of the platform.

Properties

- `ForceToPlayer`: A public float that represents the force applied to the player.

Fields

- `m_mainGameObject`: A private GameObject that represents the main game object.
- `m_isDestroyable`: A private boolean that indicates whether the platform is destroyable.
- `m_destroyHitCount`: A private integer that represents the number of hits required to destroy the platform.
- `m_destroyableObject`: A private DestroyableObject that represents the destroyable object.
- `m_levelNameObject`: A private LevelNameObject that represents the level name object.
- `m_sibelParts`: A private list of SibelPart objects that represent the sibel parts of the jump platform.
- `repeatNumber`: A private integer that represents the number of times the scale changing by hit is repeated.
- `m_trampoline`: A private Trampoline that represents the trampoline object.
- `m_ufo`: A private UFO that represents the UFO object from the first design.
- `m_hitParticle`: A private ParticleSystem that represents the hit particle of the platform.
- `particleMainModule`: A private MainModule that represents the main module of the particle system.
- `m_scaleTween`: A private Tween that represents the scale tween animations.
- `m_rotateTween`: A private Tween that represents the rotate tween animations.
- `m_manager`: A private PlatformManager that represents the platform manager.
- `repeatCounter`: A private integer that acts as a counter for repeating the scale changing by hit.

Methods

- `Initialize(int platformNum,PlatformManager manager)`: Initializes the jump platform.
- `StartHitAnimation(float hitValue)`: Starts the hit animation of the sibel.
- `HitAnimation(float hitValue)`: Handles the hit animation of the sibel.
- `SibelPartAnimation(SibelPart sibelPart,float hitValue, float delay,int sibelPartIndex,Action<int> afterDoneCallback)`: Handles the tween for each sibel part scale and Y axis move.
- `HitToPlatform(Transform playerTransform)`: Called when the player hits the platform collider.
- `ShowDestruction()`: Called when the platform must be destroyed.
- `ShowDestroyableObject()`: Shows the fragmented object before one hit remains to destroy.

Platform Class

Overview

The `Platform` class is an abstract base class for a Jump platform in Unity. It inherits from the `MonoBehaviour` class, which is the base class from which every Unity script derives. This class is designed to be extended by other classes that implement the `Initialize` method.

Properties

- `HighJumpPlatform`: A boolean value that indicates whether the platform is a high jump platform. It is decorated with the `Header` attribute to categorize it in the Unity inspector.

- `PlatformNum`: An integer value that represents the number of the platform.

- `Manager`: An instance of the `PlatformManager` class that manages the platform.

Methods

- `Initialize(int platformNum, PlatformManager platformManager)`: An abstract method that initializes the platform. It takes an integer `platformNum` and a `PlatformManager` instance `platformManager` as parameters. The implementation of this method is left to the classes that extend the `Platform` class.

Trampoline Class

Overview

The `Trampoline` class is a `MonoBehaviour` that represents a trampoline object in a game. This class is responsible for managing the destruction of the trampoline based on the number of hits it receives from the player. The trampoline can also show a destroyable object when it is about to be destroyed.

Fields

- `m_destroyHitCount`: A private integer field that represents the number of hits required to destroy the trampoline.
- `destroyCounter`: A serialized integer field that keeps track of the number of hits the trampoline has received.
- `m_jumpPlatform`: A serialized `JumpPlatform` object that represents the platform the trampoline is on.
- `m_isDestroy`: A serialized boolean field that indicates whether the trampoline is destroyed or not.

Methods

- `Initialize(int destroyHitCount, JumpPlatform jumpPlatform)`: This method is used to initialize the trampoline with a specific number of hits required to destroy it and the platform it is on. It also resets the destroy counter.

- `ShowDestroyable()`: This method increments the destroy counter and checks if the trampoline is about to be destroyed or is destroyed. If the trampoline is about to be destroyed, it shows a destroyable object. If the trampoline is destroyed, it executes the destruction method after a delay. This method returns a boolean indicating whether the trampoline is destroyed or not.

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.

TubeRenderer

Overview

The `TubeRenderer` script is a Unity C# script that is used to generate and render a tube-like mesh in the Unity game engine. The tube's shape and size are determined by a series of points (_positions) and two radii (_radiusOne and _radiusTwo). The tube can be rendered in either world space or local space, and can use either one or two radii for its generation.

The script also provides methods for setting the positions of the points, generating the mesh of the tube, and enabling or disabling the mesh renderer. It also includes a property for getting and setting the material of the mesh.

Properties

- `Material material`: The material of the tube's mesh. Can be get or set.

Fields

- `Vector3[] _positions`: The positions of the points that determine the shape and size of the tube.
- `int _sides`: The number of sides of the tube.
- `float _radiusOne`: The first radius of the tube.
- `float _radiusTwo`: The second radius of the tube.
- `bool _useWorldSpace`: Determines whether the tube is rendered in world space or local space.
- `bool _useTwoRadii`: Determines whether one or two radii are used for the generation of the tube.
- `Vector3[] _vertices`: The vertices of the tube's mesh.
- `Mesh _mesh`: The mesh of the tube.
- `MeshFilter _meshFilter`: The mesh filter component of the tube.
- `MeshRenderer _meshRenderer`: The mesh renderer component of the tube.

Methods

- `void Awake()`: Initializes the mesh filter and mesh renderer components, and creates a new mesh for the tube.
- `void SetPosition(int index, Vector3 position)`: Sets the position of a point at a specific index.
- `void OnEnable()`: Enables the mesh renderer when the script is enabled.
- `void OnDisable()`: Disables the mesh renderer when the script is disabled.
- `void Update()`: Generates the mesh of the tube every frame.
- `void OnValidate()`: Ensures that the number of sides of the tube is at least 3.
- `void SetPositions(Vector3[] positions)`: Sets the positions of the points and generates the mesh of the tube.
- `void GenerateMesh()`: Generates the mesh of the tube based on the positions of the points and the radii.
- `Vector2[] GenerateUVs()`: Generates the UVs for the tube's mesh.
- `int[] GenerateIndices()`: Generates the indices for the tube's mesh.
- `Vector3[] CalculateCircle(int index)`: Calculates the vertices of a circle at a specific index based on the positions of the points and the radii.