Documentation

Manager

An abstract class with no parameters. It has an abstract method `Initialize` with no parameters.

Manager

An abstract class with one generic parameter `T`. It has an abstract method `Initialize` with one parameter of type `T`.

Manager<T, T1>

An abstract class with two generic parameters `T` and `T1`. It has an abstract method `Initialize` with two parameters of type `T` and `T1`.

GameManager Class

The `GameManager` class is a Singleton that manages the game's state and interactions between different components of the game.

Properties

- `DesignData`: Instance of scriptable object of design data.
- `EventManager`, `UIManager`, `ParticleManager`: Instances of various managers.
- `LightSaberController`, `AIController`: Instances of controllers.
- `AIEnabled`: A flag to define if AI is enabled.

Methods

- `Initialize()`: Initializes all managers and controllers, and assigns methods to events based on conditions.
- `Simulate()`: Invokes the `OnSimulateEvent`.
- `SimulateFinishedWithoutHit()`: Calls the `GameOver` method.
- `ResetToStart()`: Resets everything to the initial state by calling the `Initialize` method.
- `OnRotateChange(LightSaberName, PlayerType, RotateDirection, float)`: Invokes the `OnLightSaberRotateEvent`.
- `LightSabersHitted(Vector3)`: Invokes the `OnHitEvent` and calls the `GameOver` method.
- `GameOver()`: Invokes the `OnGameOverEvent`.

EventManager Class

The `EventManager` class is a part of the `FireByte` namespace. It extends from the `Manager` base class and is responsible for managing different types of events within the game.

Properties

- `OnLightSaberRotateEvent`: An event that is triggered when the lightsaber rotates.
- `OnHitEvent`: An event that is triggered when a hit occurs.
- `OnSimulateEvent`: An event that is triggered to simulate an action.
- `OnGameOverEvent`: An event that is triggered when the game is over.

Methods

- `Initialize()`: This method is used to initialize the `EventManager` class. It sets all the event properties to null.

ParticleManager Class

This class is responsible for managing the particle system in the game.

Properties

- `m_hitParticle`: A private ParticleSystem variable. This represents the particle system that is used when a hit occurs.

Methods

- `Initialize()`: This method is used to stop the particle system. It is called when the ParticleManager is initialized.

- `StartHitParticle(Vector3 hitPosition)`: This method is used to start the particle system at a specific position. It takes a 3D vector as an argument which represents the position where the hit occurred. The particle system is then moved to this position and started.

UIManager Class

The `UIManager` class is responsible for managing the user interface elements in the game. It extends the `Manager` class with float and bool parameters.

Methods

- `Initialize(float fadeDuration,bool aiEnabled)`: Initializes the UI elements. Sets up the simulate button and initializes the sliders and game over popup.

- `OnSliderValueChange(LightSaberName lightSaberName, RotateDirection rotateDirection, float value)`: Handles the event of slider value change. It calls the `OnRotateChange` function of the `GameManager` instance.

- `ShowGameOverPopup()`: Shows the game over popup. It removes the value change listeners from the sliders and shows the game over popup.

- `SimulateButtonInit()`: Initializes the simulate button. It sets the text of the button to "Simulate" and sets the onClick event to the `onSimulateButtonClick` function.
- `ChangeSimulateButtonToResetButton()`: Changes the simulate button to a reset button. It sets the text of the button to "Reset" and sets the onClick event to the `onResetButtonClick` function.
- `onSimulateButtonClick()`: Handles the click event of the simulate button. It calls the `Simulate` function of the `GameManager` instance.
- `onResetButtonClick()`: Handles the click event of the reset button. It calls the `ResetToStart` function of the `GameManager` instance.

Controller Overview

This code defines two abstract classes for a Unity game, both named `Controller`. These classes are meant to be inherited by other classes in the game.

Controller Class

The `Controller` class is an abstract class that inherits from Unity's `MonoBehaviour`. It declares an abstract `Initialize` method with no parameters.

```
public abstract class Controller : MonoBehaviour
{
    public abstract void Initialize();
}
```

Controller Class

The `Controller<T>` class is also an abstract class that inherits from `MonoBehaviour`. This class is a generic version of the `Controller` class. It declares an abstract `Initialize` method that takes one parameter of a generic type `T`.

DesignData

Overview

The `DesignData` class is a `ScriptableObject` in the `FireByte` namespace. This class is used to store design-related data for the game. It contains several public fields that can be adjusted in the Unity editor. These fields are categorized into AI, Light Saber Animation, Animation Duration, and UI.

Fields

- `AIEnabled`: A boolean field that determines if the AI is enabled or not.
- `AIStartRotationX`, `AIEndRotationX`, `AIStartRotationZ`, `AIEndRotationZ`: These are float fields that define the start and end rotation angles of the AI on the X and Z axes. The values for these fields

can range from 0 to 360.

- `GoBackAnimationAngle`, `RotationAnimationAngle`: These float fields define the angles for the Go Back and Rotation animations of the Light Saber. The values for these fields can range from 0 to 360.

- `BeforeAnimationDuration`, `AnimationDelayTime`, `BeforeStartAnimationDuration`, `RotateAnimationDuration`: These float fields define various durations related to the animations. The values for these fields can range from 0.1 to 10.

- `FadeDuration`: This float field defines the duration of the UI fade. The value for this field can range from 0.1 to 2.

Please note that all the fields in this class are public and can be set directly from the Unity editor.

GameOverPopup Class

Overview

The `GameOverPopup` class is a subclass of `UIPopUp`. It represents a game over popup in the game which can be shown or hidden with fade effects. It uses the DOTween library to animate the fade effect.

Properties

There are no public properties in this class.

Fields

- `private CanvasGroup m_canvasGroup`: A serialized private field that holds the CanvasGroup component of the game over popup.
- `private Text m_gameOverText`: A serialized private field that holds the Text component used to display the game over message.
- `private float m_fadeDuration`: A serialized private field that holds the duration of the fade effect.

Methods

- `public override void Initialize(float fadeDuration)`: This method is used to initialize the game over popup. It sets the fade duration and hides the popup.
- `public override void Show()`: This method is used to show the game over popup. It sets the game object to active and starts the fade in animation.
- `public override void Hide()`: This method is used to hide the game over popup. It starts the fade out animation and sets the game object to inactive when the animation is completed.

UIButton Class Documentation

Overview

The `UIButton` class is a MonoBehaviour derived class in Unity3D, designed to handle UI button interactions and text display. It uses Unity's UI Button and Text components to perform its operations. The class provides methods to set button click events and to set the text displayed on the UI.

Properties

There are no public properties in this class.

Fields

- `private Button m_button`: A serialized private field of type `Button`. This represents the button that the `UIButton` class will manipulate.
- `private Text m_text`: A serialized private field of type `Text`. This represents the text that will be displayed on the UI.

Methods

- `public void SetButtonOnClickEvent(bool removeOnClickListener, Action action)`: This method is used to set the click event for the button. It takes two parameters: a boolean `removeOnClickListener` and an `Action` delegate `action`. If `removeOnClickListener` is true, it removes all listeners from the button's onClick event before adding the new listener.
- `public void SetText(string textStr)`: This method is used to set the text displayed on the UI. It takes a string `textStr` as a parameter and sets the text of the `m_text` field to this string.

UISlider

Overview

The `UISlider` class is a Unity MonoBehaviour script that manages a UI Slider component. It is responsible for initializing the slider, handling value changes, and managing listeners for the slider's value change event. The slider's interactivity can be toggled based on whether AI is enabled and the type of light saber associated with the slider.

Fields

- `private UIManager m_manager;` - A reference to the UIManager that manages the UI elements.
- `[SerializeField] private Slider m_slider;` - The Slider component that this script manages.
- `[SerializeField] private Text m_label;` - A Text component that displays the label for the slider.
- `[SerializeField] private LightSaberName m_lightSaberName;` - An enum value that represents the type of light saber associated with the slider.
- `[SerializeField] private RotateDirection m_rotateDirection;` - An enum value that represents the rotation direction of the light saber.

Methods

- `public void Initialize(UIManager manager,bool aiEnabled)` - This method initializes the slider. If AI is enabled and the light saber is blue, the slider is set to non-interactable. The method also sets the initial value of the slider and adds a listener for the slider's value change event.

- `public void RemoveOnValueChangeListener()` - This method removes all listeners from the slider's value change event.
- `private void OnValueChange(Single value)` - This method is called when the slider's value changes. It calls the `OnSliderValueChange` method of the UIManager, passing in the light saber type, rotation direction, and new value.

Unity C# Script Documentation

Overview

This script defines two public enumerations, `PlayerType` and `LightSaberName`.

`PlayerType` is used to distinguish between different types of players in the game, specifically whether the player is a human player or an AI-controlled player.

`LightSaberName` is used to specify the color of a lightsaber, which can either be red or blue.

Enumerations

PlayerType

This enumeration has two possible values:

- `Player`: Represents a human player.
- `AI`: Represents an AI-controlled player.

LightSaberName

This enumeration has two possible values:

- `Red`: Represents a red lightsaber.
- `Blue`: Represents a blue lightsaber.

Events

SimulateEvent

This delegate represents the method that will handle an event with no parameters. It's typically used for simulation events.

LightSaberRotateEvent

This delegate represents the method that will handle an event with four parameters: `LightSaberName lightSaberName`, `PlayerType playerType`, `RotateDirection rotateDirection`, and `float value`. It's typically used for lightsaber rotation events.

HitEvent

This delegate represents the method that will handle an event with a single `Vector3` parameter. It's typically used for hit detection events.

ResetEvent

This delegate represents the method that will handle an event with no parameters. It's typically used for game reset events.

GameOverEvent

This delegate represents the method that will handle an event with no parameters. It's typically used for game over events.

IRotateWithTweener Interface Documentation

Overview

The `IRotateWithTweener` interface is a Unity C# script that provides a contract for any class that implements it to define the `RotateWithTween` method. This method is intended to rotate an object with a tweening effect along the x and z axes.

Methods

RotateWithTween

```
void RotateWithTween(float x, float z)
```

This method is used to rotate an object with a tweening effect along the x and z axes. The method takes two parameters: `x` and `z`, which represent the rotation values along the x and z axes respectively. Any class implementing this interface must provide an implementation for this method.