Cyfrowe Labirynty

Table of Contents

Script Reference

GradientSkybox.shader

AudioManager.cs

Classes

Sound

```
[System.Serializable]
public class Sound
```

Serializable class that represents a sound clip with some properties.

Constructors

Sound

```
public  Sound()
```

Constructor that initializes the default values for volume, pitch and loop.

Variables

name

```
public string name
```

Name of the sound.

clip

```
public AudioClip clip
```

The actual AudioClip that represents the sound.

volume

```
[Range(0,1)]
public float volume
```

The volume level of the sound, ranging from 0 to 1.

pitch

```
[Range(-3,3)]
public float pitch
```

The pitch level of the sound, ranging from -3 to 3.

loop

```
public bool loop
```

Whether the sound should loop or not.

source

```
public AudioSource source
```

AudioSource component that plays the sound.

AudioManager

```
public class AudioManager
```

Singleton class that manages audio playback.

Variables

sounds

```
public Sound[] sounds
```

Array of Sound objects to manage.

instance

```
public static AudioManager instance
```

Static instance of the AudioManager.

Methods

Awake

```
void Awake()
```

Awake is called when the script instance is being loaded. It initializes the AudioManager and its sounds.

Play

```
public void Play(
     string name)
```

Play a sound with the given name. If the sound does not exist, it will log a warning.

name: Name of the sound to play.

Stop

```
public void Stop(
     string name)
```

Stop the sound with the given name.

name: Name of the sound to stop.

C# BlinkingLight.cs

Classes

BlinkingLight

```
public class BlinkingLight
```

Controls a light source to create a blinking effect.

Variables

lightSource

```
public Light lightSource
```

The Light source component that will be controlled by this script.

speed

```
public float speed
```

The speed at which the blinking effect takes place.

minIntensity

```
public float minIntensity
```

The minimum intensity for the light during the blinking effect.

maxIntensity

```
public float maxIntensity
```

The maximum intensity for the light during the blinking effect.

randomOffset

```
private float randomOffset
```

Random offset to differentiate blinking pattern from other blinking lights.

Methods

Start

```
void Start()
```

Called before the first frame update. Initializes the random offset for the blinking effect.

Update

```
void Update()
```

Called once per frame. Updates the light intensity to create a blinking effect.

C# ExamCheckSystem.cs

Classes

ExamCheckSystem

```
public class ExamCheckSystem
```

Checks if all inventory slots are filled.

Variables

inventorySlots

```
public InventorySlot[] inventorySlots
```

Array of InventorySlot objects representing the slots in the inventory.

inventory

```
[Header("Inventory")]
public GameObject inventory
```

The GameObject representing the inventory.

canvas

```
public GameObject canvas
```

The canvas that will be activated when all slots are filled.

Methods

Update

```
private void Update()
```

Updates each frame and checks if all slots are filled. If all slots are filled, the canvas is activated.

AreAllSlotsFilled

```
public bool AreAllSlotsFilled()
```

Checks if all the slots in the inventory are filled with items.

Returns: True if all slots are filled, false otherwise.

C# FadeScreen.cs

Classes

FadeScreen

```
public class FadeScreen
```

Class to manage screen fading effects.

Variables

fadeOnStart

```
public bool fadeOnStart
```

Flag to determine whether to fade in at the start.

fadeDuration

```
public float fadeDuration
```

Duration of the fade effect in seconds.

fadeColor

```
public Color fadeColor
```

Color of the fade effect.

fadeCurve

```
public AnimationCurve fadeCurve
```

Animation curve to control the progression of the fade effect.

colorPropertyName

```
public string colorPropertyName
```

The name of the color property of the Renderer's material.

rend

```
private Renderer rend
```

The Renderer component attached to the GameObject.

Methods

Start

```
void Start()
```

Called before the first frame update. Gets the Renderer component and initializes the fade effect.

FadeIn

```
public void FadeIn()
```

Fades in from the fade color to transparent.

FadeOut

```
public void FadeOut()
```

Fades out from transparent to the fade color.

Fade

```
public void Fade(
     float alphaIn,
     float alphaOut)
```

Triggers a fade effect from a given initial alpha value to a final alpha value.

alphaIn: Initial alpha value.
alphaOut: Final alpha value.

FadeRoutine

```
public IEnumerator FadeRoutine(
     float alphaIn,
     float alphaOut)
```

Coroutine that performs the fade effect from a given initial alpha value to a final alpha value.

alphaIn: Initial alpha value.
alphaOut: Final alpha value.

Returns: IEnumerator for coroutine.

C# GameStartMenu.cs

Classes

GameStartMenu

```
public class GameStartMenu
```

Controls the behaviour of the game start menu.

Variables

mainMenu

```
[Header("UI Pages")]
public GameObject mainMenu
```

The GameObject representing the main menu.

options

```
public GameObject options
```

The GameObject representing the options menu.

about

```
public GameObject about
```

The GameObject representing the about menu.

startButton

```
[Header("Main Menu Buttons")]
public Button startButton
```

The Button component representing the start button.

optionButton

```
public Button optionButton
```

The Button component representing the options button.

aboutButton

```
public Button aboutButton
```

The Button component representing the about button.

quitButton

```
public Button quitButton
```

The Button component representing the quit button.

returnButtons

```
public List<Button> returnButtons
```

The List of Button components representing the return buttons.

Methods

Start

```
void Start()
```

Called before the first frame update. Initializes the menu and the button events.

QuitGame

```
public void QuitGame()
```

Quits the game.

StartGame

```
public void StartGame()
```

Starts the game by transitioning to the next scene.

HideAll

```
public void HideAll()
```

Hides all the menus.

EnableMainMenu

```
public void EnableMainMenu()
```

Enables the main menu and hides the other menus.

EnableOption

```
public void EnableOption()
```

Enables the options menu and hides the other menus.

EnableAbout

```
public void EnableAbout()
```

Enables the about menu and hides the other menus.

C# InputToInteractable.cs

Classes

InputToInteractable

```
public class InputToInteractable
```

This class allows the interaction of an XRGrabInteractable object based on the text input in a TMP_InputField.

Variables

inputField

```
public TMP_InputField inputField
```

The input field where the text input is received.

drawerInteractable

```
public XRGrabInteractable drawerInteractable
```

The XRGrabInteractable object that is controlled based on the input field's value.

Methods

Start

```
void Start()
```

Called before the first frame update. Initializes the input field event.

CheckInputValue

```
private void CheckInputValue(
     string inputValue)
```

Checks the value of the input field and enables the XRGrabInteractable object if the input matches "CABINET OPEN".

inputValue: The value of the input field.

C# InventoryItem.cs

Classes

InventoryItem

```
public class InventoryItem
```

This class represents an item in an inventory.

Variables

itemObject

```
public GameObject itemObject
```

The GameObject representing the item.

slotIndex

```
public int slotIndex
```

The index of the slot in which the item is placed in the inventory.

C# InventoryManager.cs

Classes

InventoryManager

```
public class InventoryManager
```

Manages the inventory system in the game, including item placement, activation and deactivation.

Variables

inventorySlots

```
public InventorySlot[] inventorySlots
```

Array of InventorySlot, representing the slots in the inventory.

inventoryItems

```
public List<InventoryItem> inventoryItems
```

List of InventoryItem, storing the items when the inventory is inactive.

inventoryUI

```
[Header("Inventory")]
public GameObject inventoryUI
```

The GameObject representing the inventory UI.

Methods

ActivateInventory

```
public void ActivateInventory()
```

Activates the inventory, repositioning items back to their slots.

DeactivateInventory

```
public void DeactivateInventory()
```

Deactivates the inventory, storing the items in a list and deactivating their GameObjects.

C# InventorySlot.cs

Classes

InventorySlot

```
public class InventorySlot
```

This class represents an inventory slot that inherits from XRSocketInteractor, allowing for interactions in VR.

Variables

currentItem

```
private GameObject currentItem
```

The GameObject currently placed in the inventory slot.

Methods

OnSelectEntered

```
[System.Obsolete]
protected override void OnSelectEntered(
     XRBaseInteractable interactable)
```

This method is called when an interactable object enters the slot.

interactable: The interactable object that entered the slot.

OnSelectExited

```
[System.Obsolete]
protected override void OnSelectExited(
     XRBaseInteractable interactable)
```

This method is called when an interactable object exits the slot.

interactable: The interactable object that exited the slot.

HasItem

```
public bool HasItem()
```

Checks whether the slot currently holds an item.

Returns: True if the slot has an item, false otherwise.

GetItem

```
public GameObject GetItem()
```

Retrieves the item currently placed in the slot.

Returns: The GameObject of the item if present, null otherwise.

SetItem

```
public void SetItem(
     GameObject item)
```

Sets the current item of the inventory slot.

item: The GameObject representing the item to set.

C# KeyPad.cs

Classes

KeyPad

```
public class KeyPad
```

This class manages the operations of a Keypad, including storing the correct password, capturing user input, and invoking events based on the correctness of the input.

Variables

correctPassword

```
public List<int> correctPassword
```

Stores the correct password as a list of integers.

inputPasswordList

```
public List<int> inputPasswordList
```

Stores the input password as a list of integers.

codeDisplay

```
[SerializeField]
private TMP_InputField codeDisplay
```

UI element to display the code.

resetTime

```
[SerializeField]
private float resetTime
```

Time after which the keypad is reset.

successText

```
[SerializeField]
private string successText
```

Text to be displayed upon successful password entry.

onCorrectPassword

```
[Space(5f)]
[Header("Keypad Entry Events")]
public UnityEvent onCorrectPassword
```

Event to be invoked upon successful password entry.

onIncorrectPassword

```
public UnityEvent onIncorrectPassword
```

Event to be invoked upon unsuccessful password entry.

numberOfEntry

```
[Header("Number Of Entry")]
public int numberOfEntry
```

Number of entries allowed in the password.

allowMultipleActivations

```
public bool allowMultipleActivations
```

Defines whether multiple successful activations are allowed.

hasUsedCorrectCode

```
private bool hasUsedCorrectCode
```

Tracks if the correct password has already been used.

Properties

HasUsedCorrectCode

```
public bool HasUsedCorrectCode
     { get;
     }
```

Public accessor for hasUsedCorrectCode

Methods

UserNumberEntry

```
public void UserNumberEntry(
     int selectNum)
```

Method to handle user number entry. Adds the input number to the password list, updates the display, and checks the password if enough numbers have been entered.

CheckPassword

```
private void CheckPassword()
```

Method to check if the entered password is correct. If any number in the input password does not match the correct password, it calls the IncorrectPassword() method.

correctPasswordGiven

```
private void correctPasswordGiven()
```

Method to handle the event of correct password input. It invokes the onCorrectPassword event, updates the display to show the successText, and starts a ResetKeyCode coroutine.

IncorrectPassword

```
private void IncorrectPassword()
```

Method to handle the event of incorrect password input. It invokes the onIncorrectPassword event and starts a ResetKeyCode coroutine.

UpdateDisplay

```
private void UpdateDisplay()
```

Method to update the display with the currently entered password.

DeleteEntry

```
public void DeleteEntry()
```

Method to delete an entry from the entered password.

ResetKeyCode

```
IEnumerator ResetKeyCode()
```

Coroutine to reset the entered password after a certain period of time (defined by resetTime).

C# LeverSequenceChecker.cs

Classes

LeverSequenceChecker

```
public class LeverSequenceChecker
```

Class responsible for checking lever sequences.

Variables

resultText

```
[SerializeField]
private TMP_InputField resultText
```

UI element to display the result text.

hand1

```
[Header("Cabinet Hand One")]
[SerializeField]
public GameObject hand1
```

Game object representing the first hand.

hand2

```
[Header("Cabinet Hand Two")]
[SerializeField]
public GameObject hand2
```

Game object representing the second hand.

expectedSequence

```
private bool[] expectedSequence
```

The expected lever sequence. True means the lever is active and False means the lever is inactive.

firstLever

```
private bool firstLever
```

States of the individual levers.

secondLever

```
private bool secondLever
```

thirdLever

```
private bool thirdLever
```

fourthLever

```
private bool fourthLever
```

fifthLever

```
private bool fifthLever
```

Methods

activeFirstLaver

```
public void activeFirstLaver()
```

Methods to change the state of each lever.

disActivateFirstLaver

```
public void disActivateFirstLaver()
```

activeSecondLaver

```
public void activeSecondLaver()
```

disActivateSecondLaver

```
public void disActivateSecondLaver()
```

activeThirdLaver

```
public void activeThirdLaver()
```

disActivateThirdLaver

```
public void disActivateThirdLaver()
```

activeFourthLaver

```
public void activeFourthLaver()
```

disActivateFourthLaver

```
public void disActivateFourthLaver()
```

activeFifthLaver

```
public void activeFifthLaver()
```

disActivateFifthLaver

```
public void disActivateFifthLaver()
```

Start

```
private void Start()
```

Update

```
private void Update()
```

checkSequence

```
public void checkSequence()
```

Method to compare the current state of the levers to the expected sequence. If the sequence is correct, it sets the result text to "Cabinet Open!" and enables the XRGrabInteractable components on the hands. If the sequence is incorrect, it sets the result text to "Sequence Incorrect!".

OpenBook.cs

Classes

OpenBook

```
public class OpenBook
```

Class responsible for managing the book's open and close mechanism.

Variables

Cover

```
public GameObject Cover
```

Game object representing the cover of the book.

myHinge

```
public HingeJoint myHinge
```

The HingeJoint component of the book cover which allows it to open and close.

Methods

Start

```
void Start()
```

Method called at the start of the first frame, it gets the HingeJoint component of the book cover and disables its motor.

OpenSesame

```
public void OpenSesame()
```

Method responsible for opening the book by enabling the motor of the HingeJoint component of the book cover.

C# Photo.cs

Classes

Photo

```
[RequireComponent(typeof(ApplyPhysics))]
public class Photo
```

A class responsible for managing photos, including their ejection over a certain period of time and setting their image texture.

Variables

imageRenderer

```
public MeshRenderer imageRenderer
```

The MeshRenderer of the image.

currentCollider

```
private Collider currentCollider
```

The current collider of the photo.

applyPhysics

```
private ApplyPhysics applyPhysics
```

The ApplyPhysics component of the photo.

Methods

Awake

```
private void Awake()
```

Awake is called when the script instance is being loaded.

Start

```
private void Start()
```

Start is called just before any of the Update methods is called the first time.

EjectOverSeconds

```
public IEnumerator EjectOverSeconds(
     float seconds)
```

Eject the photo over a certain number of seconds.

seconds: The number of seconds over which to eject the photo.

Returns: An IEnumerator for use in a coroutine.

SetImage

```
public void SetImage(
     Texture2D texture)
```

Set the image texture of the photo.

texture: The texture to set.

EnablePhysics

```
public void EnablePhysics()
```

Enable physics for the photo and unparent it from its current parent.

C# Polaroid.cs

Classes

Polaroid

```
public class Polaroid
```

This class manages the functionality of a Polaroid camera, including taking photos and creating render textures.

Variables

photoPrefab

```
public GameObject photoPrefab
```

The prefab to use for photos.

screenRenderer

```
public MeshRenderer screenRenderer
```

The MeshRenderer used for the screen.

spawnLocation

```
public Transform spawnLocation
```

The location at which photos should spawn.

renderCamera

```
private Camera renderCamera
```

The camera used for rendering.

Methods

Awake

```
private void Awake()
```

Awake is called when the script instance is being loaded.

Start

```
private void Start()
```

Start is called just before any of the Update methods is called the first time.

CreateRenderTexture

```
private void CreateRenderTexture()
```

Create a new RenderTexture and assign it to the render camera and screen material.

TakePhoto

```
public void TakePhoto()
```

Take a photo using the Polaroid camera.

CreatePhoto

```
private Photo CreatePhoto()
```

Create a new photo at the spawn location.

Returns: The Photo component of the new photo.

SetPhotoImage

```
private void SetPhotoImage(
     Photo photo)
```

Set the image of a photo using the render camera.

photo: The photo for which to set the image.

RenderCameraToTexture

```
private Texture2D RenderCameraToTexture(
     Camera camera)
```

Render the camera view to a Texture2D.

camera: The camera to render.

Returns: A Texture2D of the camera view.

TurnOn

```
public void TurnOn()
```

Turn on the Polaroid camera.

TurnOff

```
public void TurnOff()
```

Turn off the Polaroid camera.

C# SceneTransitionManager.cs

Classes

SceneTransitionManager

```
public class SceneTransitionManager
```

This class manages scene transitions, including both synchronous and asynchronous scene loading.

Variables

fadeScreen

```
public FadeScreen fadeScreen
```

The screen fading effect used during scene transitions.

singleton

```
public static SceneTransitionManager singleton
```

The singleton instance of the SceneTransitionManager.

Methods

Awake

```
private void Awake()
```

Awake is called when the script instance is being loaded.

GoToScene

```
public void GoToScene(
     int sceneIndex)
```

Go to a specified scene synchronously.

sceneIndex: The build index of the scene to go to.

GoToSceneRoutine

```
IEnumerator GoToSceneRoutine(
     int sceneIndex)
```

Coroutine that handles the process of fading out the screen and synchronously loading a new scene.

sceneIndex: The build index of the scene to go to.

GoToSceneAsync

```
public void GoToSceneAsync(
     int sceneIndex)
```

Go to a specified scene asynchronously.

sceneIndex: The build index of the scene to go to.

GoToSceneAsyncRoutine

```
IEnumerator GoToSceneAsyncRoutine(
     int sceneIndex)
```

Coroutine that handles the process of fading out the screen and asynchronously loading a new scene.

sceneIndex: The build index of the scene to go to.

C# SetOptionFromUI.cs

Classes

SetOptionFromUI

```
public class SetOptionFromUI
```

This class handles updating the user's preferences from the UI, including volume and turn type.

Variables

volumeSlider

```
public Scrollbar volumeSlider
```

Volume control slider.

turnDropdown

```
public TMPro.TMP_Dropdown turnDropdown
```

Dropdown selection for the turn type.

turnTypeFromPlayerPref

```
public SetTurnTypeFromPlayerPref turnTypeFromPlayerPref
```

Script to apply turn type based on player's preferences.

Methods

Start

```
private void Start()
```

Invoked at the start of the scene, sets up the volume and turn type from player preferences.

SetGlobalVolume

```
public void SetGlobalVolume(
     float value)
```

Sets the global audio volume.

value: The value to set the global volume to, on a scale from 0 to 1.

SetTurnPlayerPref

```
public void SetTurnPlayerPref(
     int value)
```

Sets the player preference for turn type and applies the change.

value: The value representing the turn type in player preferences.

C# SetTurnTypeFromPlayerPref.cs

Classes

SetTurnTypeFromPlayerPref

```
public class SetTurnTypeFromPlayerPref
```

This class manages the turn type setting based on the user's preferences.

Variables

snapTurn

```
public ActionBasedSnapTurnProvider snapTurn
```

Reference to the Snap Turn Provider.

continuousTurn

```
public ActionBasedContinuousTurnProvider continuousTurn
```

Reference to the Continuous Turn Provider.

Methods

Start

```
void Start()
```

At the start of the scene, apply the turn type setting from player preferences.

ApplyPlayerPref

```
public void ApplyPlayerPref()
```

Applies the player's preferred turn type. If "turn" key has value 0, snap turn is enabled. If it has value 1, continuous turn is enabled.

C# ShowKeyboard.cs

Classes

ShowKeyboard

```
public class ShowKeyboard
```

This class manages the display of a keyboard for the input field.

Variables

inputField

```
private TMP_InputField inputField
```

Reference to the InputField component attached to the GameObject.

Methods

Start

```
void Start()
```

At the start of the scene, it gets the input field component and sets up the event listener.

OpenKeyboard

```
public void OpenKeyboard()
```

Opens the keyboard and sets it to the text currently in the input field.

C# UIAudio.cs

Classes

UIAudio

```
public class UIAudio
```

The UIAudio class is used to play specific audio clips in response to user interactions with the UI.

Variables

clickAudioName

```
public string clickAudioName
```

The name of the audio clip to play when the UI element is clicked.

hoverEnterAudioName

```
public string hoverEnterAudioName
```

The name of the audio clip to play when the mouse pointer enters the UI element.

hoverExitAudioName

```
public string hoverExitAudioName
```

The name of the audio clip to play when the mouse pointer exits the UI element.

Methods

OnPointerClick

```
public void OnPointerClick(
     PointerEventData eventData)
```

Called when the UI element is clicked. Plays the click audio clip.

eventData: Data associated with the PointerClick event.

OnPointerEnter

```
public void OnPointerEnter(
     PointerEventData eventData)
```

Called when the mouse pointer enters the UI element. Plays the hover enter audio clip.

eventData: Data associated with the PointerEnter event.

OnPointerExit

```
public void OnPointerExit(
     PointerEventData eventData)
```

Called when the mouse pointer exits the UI element. Plays the hover exit audio clip.

eventData: Data associated with the PointerExit event.