using UnityEngine;

using System.Collections;

using DG.Tweening;

namespace AppAdvisory.MathGame

{

public class AnimTitle : MonoBehaviour

{

void OnEnable()

{

for (int i = 0; i < transform.childCount; i++)

{

transform.GetChild (i).localScale = Vector3.one;

}

StartCoroutine (Anim ());

}

IEnumerator Anim()

{

yield return new WaitForSeconds (0.3f);

while (true)

{

float time = 0.2f\*5;

int childCount = transform.childCount;

for (int i = 0; i < childCount; i++)

{

float delay = i \* 0.05f\*5;

transform.GetChild (i).DOScale (Vector3.one\*0.7f,time).SetDelay(delay).SetEase(Ease.InBack);

transform.GetChild (i).DOScale (Vector3.one\*1f,time).SetDelay(/\*2\*\*/time + delay).SetEase(Ease.OutBack);

}

yield return new WaitForSeconds (1\*(time + childCount \* 0.05f\*4) + 2f);

}

}

void OnDisable()

{

StopAllCoroutines ();

for (int i = 0; i < transform.childCount; i++)

{

DOTween.Kill(transform.GetChild (i));

}

}

}

}

using UnityEngine;

using System.Collections;

using UnityEngine.UI;

namespace AppAdvisory.MathGame

{

public class AnswerInGameButton: ButtonHelper

{

Text text;

//do the state for mobile normal on highlightened

void Awake()

{

if(Application.isMobilePlatform)

GetComponent<Button> ().animationTriggers.highlightedTrigger = "Normal";

else

GetComponent<Button> ().animationTriggers.highlightedTrigger = "Highlighted";

}

override public void OnClicked()

{

if(text == null)

text = GetComponentInChildren<Text> ();

gameManager.OnClicked (text);

}

}

}

using UnityEngine;

using System.Collections;

namespace AppAdvisory.MathGame

{

public class ButtonCloseSetting : ButtonHelper

{

override public void OnClicked()

{

print ("OnClicked : " + gameObject.name);

menuManager.CloseSettings ();

RemoveListener();

}

}

}

using UnityEngine;

using System.Collections;

namespace AppAdvisory.MathGame

{

public class ButtonFacebook : MonoBehaviour

{

public void OnClicked(){

string facebookApp = "fb://profile/515431001924232" ;

string facebookAddress = "https://www.facebook.com/appadvisory";

float startTime;

startTime = Time.timeSinceLevelLoad;

//open the facebook app

Application.OpenURL(facebookApp);

if (Time.timeSinceLevelLoad - startTime <= 1f)

{

//fail. Open safari.

Application.OpenURL(facebookAddress);

}

}

}

}

using UnityEngine;

using UnityEngine.UI;

using System.Collections;

using MenuBarouch;

namespace AppAdvisory.MathGame

{

public class ButtonHelper : MonoBehaviour

{

MenuManager \_menuManager;

public MenuManager menuManager

{

get

{

if (\_menuManager == null)

\_menuManager = FindObjectOfType<MenuManager> ();

return \_menuManager;

}

}

GameManager \_gameManager;

public GameManager gameManager

{

get

{

if (\_gameManager == null)

\_gameManager = FindObjectOfType<GameManager> ();

return \_gameManager;

}

}

virtual public void OnClicked(){}

void OnEnable()

{

GetComponent<Button> ().onClick.AddListener (OnClicked);

}

void OnDisable()

{

RemoveListener();

}

public void RemoveListener()

{

GetComponent<Button> ().onClick.RemoveListener(OnClicked);

}

}

}

using UnityEngine;

using System.Collections;

#if APPADVISORY\_LEADERBOARD

using AppAdvisory.social;

#endif

namespace AppAdvisory.MathGame

{

public class ButtonLeaderboard : ButtonHelper

{

override public void OnClicked()

{

OnClickedOpenLeaderboard();

}

/// <summary>

/// If player clics on the leaderbord button, we call this method. Works only on mobile (iOS & Android) if using Very Simple Leaderboard by App Advisory : http://u3d.as/qxf

/// </summary>

public void OnClickedOpenLeaderboard()

{

#if APPADVISORY\_LEADERBOARD

LeaderboardManager.ShowLeaderboardUI();

#else

Debug.LogWarning("OnClickedOpenLeaderboard : works only on mobile (iOS & Android), with Very Simple Leaderboard : http://u3d.as/qxf");

#endif

}

}

}

using UnityEngine;

using System.Collections;

namespace AppAdvisory.MathGame

{

public class ButtonSetting : ButtonHelper

{

override public void OnClicked()

{

print ("OnClicked : " + gameObject.name);

menuManager.OpenSettings ();

RemoveListener();

}

}

}

using UnityEngine;

using UnityEngine.UI;

using System.Collections;

using MenuBarouch;

namespace AppAdvisory.MathGame

{

public class ButtonShare : ButtonHelper

{

#if !VS\_SHARE

public RectTransform buttonVerySimpleShare;

public string VerySimpleAdsURL = "http://u3d.as/oWD";

#endif

override public void OnClicked()

{

#if !VS\_SHARE

Debug.LogWarning("To take and share screenshots, you need Very Simple Share: " + VerySimpleAdsURL);

Debug.LogWarning("Very Simple Share: " + VerySimpleAdsURL);

Debug.LogWarning("Very Simple Share is ready to use in the game template: " + VerySimpleAdsURL);

// AnimVerySimpleShare(false);

#endif

RemoveListener();

}

}

}

using AppAdvisory.MathGame;

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class ContinueButton : ButtonHelper

{

override public void OnClicked()

{

GameManager.isContinue = true;

print("OnClicked : " + gameObject.name);

menuManager.GoToGame();

RemoveListener();

}

}

using UnityEngine;

using System.Collections;

namespace AppAdvisory.MathGame

{

public class PlayButton : ButtonHelper

{

override public void OnClicked()

{

print ("OnClicked : " + gameObject.name);

menuManager.GoToGame();

RemoveListener();

}

}

}

using UnityEngine;

using System.Collections;

namespace AppAdvisory.MathGame

{

public class RateUsButton : ButtonHelper

{

override public void OnClicked()

{

print ("OnClicked : " + gameObject.name);

}

}

}

using UnityEngine;

using System.Collections;

namespace AppAdvisory.MathGame

{

public class FixRotationOnEnable : MonoBehaviour

{

void OnEnable()

{

transform.rotation = Quaternion.identity;

}

void OnDisable()

{

transform.rotation = Quaternion.identity;

}

}

}

using UnityEngine;

using UnityEngine.UI;

using System;

using System.Linq;

using System.Collections;

using System.Collections.Generic;

using MenuBarouch;

#if APPADVISORY\_LEADERBOARD

using AppAdvisory.social;

#endif

#if APPADVISORY\_ADS

using AppAdvisory.Ads;

#endif

namespace AppAdvisory.MathGame

{

public class GameManager : MonoBehaviour

{

public int numberOfPlayToShowInterstitial = 5;

public string VerySimpleAdsURL = "http://u3d.as/oWD";

static System.Random \_random = new System.Random();

public AudioClip musicBackground;

public AudioClip goodAnswerSound;

public AudioClip falsedAnswerSound;

public int timeTotalGame;

public int timeMalus;

public int timeBonus;

public Color NORMAL\_COLOR;

public Color GOOD\_COLOR;

public Color FAIL\_COLOR;

public Image BACKGROUND\_BACK;

//public ParticleSystem particleSuccessPrefab;

public Text point;

public GameObject QUESTIONS\_GO;

public GameObject BUTTONS\_GO;

public GameObject POINTS;

public Text pointsText;

public Text questionNumber1;

public Text questionOperator; //+=0 -=1 \*=2 /=3

public Text questionNumber2;

public Text questionResult;

public Text[] reponses;

public int level; //responsible to change the speed of the fill out => so the difficulty

public int \_score;

public int GOODANSWER; //count the number of good answer, ie. the score

public Slider slider; //the slider in the top of the game screen

public Text levelText; //the text to see the level during the game

private Vector2 pivotPoint;

int \_result = 0;

int \_number1 = 0;

int \_number2 = 0;

int \_operateur = 0;

public delegate void \_GameOver();

public static event \_GameOver OnGameOver;

//todo

public static bool isContinue = false;

//play fx when answer is wrong

void PlaySoundFalse()

{

GetComponent<AudioSource>().PlayOneShot(falsedAnswerSound);

}

//play fx when answer is good

void PlaySoundGood()

{

GetComponent<AudioSource>().PlayOneShot(goodAnswerSound);

}

//play the music

void PlayMusic()

{

GetComponent<AudioSource>().Play();

}

//stop the music

void StopMusic()

{

GetComponent<AudioSource>().Stop();

}

void OnEnable()

{

Application.targetFrameRate = 60;

if(!isContinue)

{

StartGame();

}

else

{

ContinueGame();

}

}

void OnDisable()

{

StopMusic();

}

//method to start the game

private void StartGame()

{

PlayMusic();

//reset the score

\_score = 0;

//reset the level

level = 1;

point.text = \_score.ToString();

levelText.text = "等级 " + level.ToString();

//create the first question

ChooseOperator();

//start the game

StartCoroutine(TimerStart());

}

//decrease continiously the timer (= the slider), and if = 0 ==> gameover

IEnumerator TimerStart()

{

slider.maxValue = timeTotalGame;

slider.value = timeTotalGame;

while (true)

{

float timer = 0.01f + ((float)Mathf.Sqrt(level)) / 100f;

slider.value -= timer;

//if the slider == 0 ===> game over

if (slider.value == 0)

{

break;

}

yield return new WaitForSeconds(0.01f);

}

GameOver();

}

private void GameOver()

{

ScoreManager.SaveScore(\_score, level);

FindObjectOfType<MenuBarouch.MenuManager>().GoToMenu();

ReportScoreToLeaderboard(\_score);

ShowAds();

if (OnGameOver != null)

OnGameOver();

}

public void ContinueGame()

{

PlayMusic();

point.text = \_score.ToString();

levelText.text = "等级 " + level.ToString();

//create the first question

ChooseOperator();

//start the game

StartCoroutine(TimerStart());

isContinue = false;

}

//choose operateur for the question : + = 0 - = 1 \* = 2 / = 3

void ChooseOperator()

{

int operateur = 0;

if (level == 1)

{

operateur = UnityEngine.Random.Range(0, 2);

}

else if (level <= 3)

{

operateur = UnityEngine.Random.Range(0, 3);

}

else

{

operateur = UnityEngine.Random.Range(0, 4);

}

CreateQuestion(operateur);

}

void CreateQuestion(int operateur)

{

int result = 0;

int number1 = 0;

int number2 = 0;

int essai = 0;

while (true)

{

essai++;

bool isOK = true;

if (operateur == 3)

{

int mult = UnityEngine.Random.Range(2 + (int)Mathf.Log(level), 2 + 2 \* (int)Mathf.Log(level));

number2 = UnityEngine.Random.Range(2 + level / 2, 3 + level);

number1 = mult \* number2;

}

else if (operateur == 1)

{

number2 = UnityEngine.Random.Range(1 + (int)Mathf.Log(level) / 2, 5 + (int)Mathf.Log(level));

number1 = UnityEngine.Random.Range(number2 + (int)Mathf.Log(level) / 2, number2 + 5 + 2 \* (int)Mathf.Log(level));

}

else

{

number1 = UnityEngine.Random.Range(1 + (int)Mathf.Log(level) / 2, 5 + 2 \* (int)Mathf.Log(level));

number2 = UnityEngine.Random.Range(1 + (int)Mathf.Log(level) / 2, 5 + 2 \* (int)Mathf.Log(level));

}

result = GetResult(number1, number2, operateur);

if (operateur == 1 || operateur == 3)

{

int resultDIV = 0;

int resultMINUS = 0;

resultDIV = number1 / number2;

resultMINUS = number1 - number2;

if (resultDIV == resultMINUS)

{

isOK = false;

}

}

if (operateur == 0 || operateur == 2)

{

int resultMULT = 0;

int resultPLUS = 0;

resultMULT = number1 \* number2;

resultPLUS = number1 + number2;

if (resultMULT == resultPLUS)

{

isOK = false;

}

}

if (\_result == result && \_number1 == number1 && \_number2 == number2 && \_operateur == operateur)

{

isOK = false;

}

if (result <= 0)

{

isOK = false;

}

if (operateur == 3)

{

if (number1 % number2 != 0)

{

isOK = false;

}

if (number1 / number2 == 0)

{

isOK = false;

}

if (number1 / number2 == 1)

{

isOK = false;

}

}

else

{

if (operateur == 2)

{

if (number1 == 0 || number1 == 1 || number2 == 0 || number2 == 1 || result == 0 || result == 1)

{

isOK = false;

}

}

}

if (level <= 2)

{

if (result > 9)

{

isOK = false;

}

if (result <= 0 || number1 <= 0 || number2 <= 0)

{

isOK = false;

}

}

else if (level <= 4)

{

if (result > 50)

{

isOK = false;

}

if (result <= 0 || number1 <= 0 || number2 <= 0)

{

isOK = false;

}

}

else if (level <= 6)

{

if (result > 99)

{

isOK = false;

}

}

if (result > 99)

{

isOK = false;

}

//CHECK!!!

if (isOK)

{

if (operateur == 0)

{

int resultTest = number1 + number2;

if (resultTest != result)

{

isOK = false;

}

}

if (operateur == 1)

{

int resultTest = number1 - number2;

if (resultTest != result)

{

isOK = false;

}

}

if (operateur == 2)

{

int resultTest = number1 \* number2;

if (resultTest != result)

{

isOK = false;

}

}

if (operateur == 4)

{

int resultTest = number1 / number2;

if (resultTest != result)

{

isOK = false;

}

}

}

if (isOK)

{

\_result = result;

\_number1 = number1;

\_number2 = number2;

\_operateur = operateur;

break;

}

}

SetText(number1, number2, operateur, result);

}

//set the question text

private void SetText(int n1, int n2, int oper, int result)

{

int TYPE\_QUESTION = UnityEngine.Random.Range(0, 4);

if (TYPE\_QUESTION == 0)

{

questionNumber1.text = "?";

questionNumber2.text = n2.ToString();

questionOperator.text = GetOperator(oper);

questionResult.text = result.ToString();

GOODANSWER = n1;

}

if (TYPE\_QUESTION == 1)

{

questionNumber1.text = n1.ToString();

questionNumber2.text = n2.ToString();

questionOperator.text = "?";

questionResult.text = result.ToString();

GOODANSWER = oper;

}

if (TYPE\_QUESTION == 2)

{

questionNumber1.text = n1.ToString();

questionNumber2.text = "?";

questionOperator.text = GetOperator(oper);

questionResult.text = result.ToString();

GOODANSWER = n2;

}

if (TYPE\_QUESTION == 3)

{

questionNumber1.text = n1.ToString();

questionNumber2.text = n2.ToString();

questionOperator.text = GetOperator(oper);

questionResult.text = "?";

GOODANSWER = result;

}

questionNumber1.transform.parent.Find("Selected").gameObject.SetActive(TYPE\_QUESTION == 0);

questionOperator.transform.parent.Find("Selected").gameObject.SetActive(TYPE\_QUESTION == 1);

questionNumber2.transform.parent.Find("Selected").gameObject.SetActive(TYPE\_QUESTION == 2);

questionResult.transform.parent.Find("Selected").gameObject.SetActive(TYPE\_QUESTION == 3);

if (TYPE\_QUESTION != 1)

{

int[] answers = new int[4];

List<int> l = new List<int>();

l.Add(GOODANSWER);

while (true)

{

int ans = 0;

int addRange = 0;

while (true)

{

bool isOk = true;

ans = UnityEngine.Random.Range(1, GOODANSWER \* 2 + 3);

if (ans <= 0)

isOk = false;

if (isOk)

break;

addRange++;

}

if (!l.Contains(ans))

l.Add(ans);

if (l.Count == 4)

break;

}

l.Sort();

answers = l.ToArray();

Array.Sort(answers);

for (int i = 0; i < 4; i++)

{

reponses[i].fontSize = 90;

reponses[i].text = answers[i].ToString();

}

}

else

{

reponses[0].text = "+";

reponses[0].fontSize = 130;

reponses[1].text = "-";

reponses[1].fontSize = 130;

reponses[2].text = "x";

reponses[3].text = "÷";

}

}

public void OnClicked(Text text)

{

int myAnswer;

bool isMaybeOperator = text.text.Length <= 1;

if (text.text.Contains("+") && isMaybeOperator)

myAnswer = 0;

else if (text.text.Contains("-") && isMaybeOperator)

myAnswer = 1;

else if (text.text.Contains("x") && isMaybeOperator)

myAnswer = 2;

else if (text.text.Contains("÷") && isMaybeOperator)

myAnswer = 3;

else

myAnswer = int.Parse(text.text);

if (GOODANSWER == myAnswer)

{

\_score++;

if (\_score % 5 == 0)

{

level++;

levelText.text = "等级 " + level.ToString();

}

pointsText.text = \_score.ToString();

StartCoroutine(GoodAnswerAnim());

slider.value += timeTotalGame;

AnimColorBACKGROUND\_BACK(true);

BUTTONS\_GO.GetComponent<Animator>().Play("AnimButtonGame");

PlaySoundGood();

}

else

{

slider.value -= timeTotalGame / 5;

PlaySoundFalse();

AnimColorBACKGROUND\_BACK(false);

}

}

private int GetResult(int n1, int n2, int oper)

{

if (oper == 0)

return n1 + n2;

else if (oper == 1)

return n1 - n2;

else if (oper == 2)

return n1 \* n2;

else if (oper == 3)

return n1 / n2;

else

return 0;

}

private string GetOperator(int oper)

{

if (oper == 0)

return "+";

else if (oper == 1)

return "-";

else if (oper == 2)

return "x";

else if (oper == 3)

return "÷";

else

return "";

}

IEnumerator GoodAnswerAnim()

{

float time = 0.2f;

QUESTIONS\_GO.GetComponent<Animator>().Play("AnimQuestionChange");

//var goParticle = Instantiate(particleSuccessPrefab.gameObject) as GameObject;

//var tParticle = goParticle.transform;

//tParticle.position = new Vector3(point.transform.position.x, point.transform.position.y, point.transform.position.z + 2);

//tParticle.rotation = Quaternion.identity;

//goParticle.SetActive(true);

//goParticle.GetComponent<ParticleSystem>().Emit(50);

yield return new WaitForSeconds(time + 0.01f);

ChooseOperator();

}

public static string[] RandomizeStrings(string[] arr)

{

List<KeyValuePair<int, string>> list = new List<KeyValuePair<int, string>>();

// Add all strings from array

// Add new random int each time

foreach (string s in arr)

{

list.Add(new KeyValuePair<int, string>(\_random.Next(), s));

}

// Sort the list by the random number

var sorted = from item in list

orderby item.Key

select item;

// Allocate new string array

string[] result = new string[arr.Length];

// Copy values to array

int index = 0;

foreach (KeyValuePair<int, string> pair in sorted)

{

result[index] = pair.Value;

index++;

}

// Return copied array

return result;

}

public void AnimColorBACKGROUND\_BACK(bool isGoodAnswer)

{

StartCoroutine(AnimColorBACKGROUND\_BACK\_Corout(isGoodAnswer));

}

IEnumerator AnimColorBACKGROUND\_BACK\_Corout(bool isGoodAnswer)

{

Color c = FAIL\_COLOR;

var time = 0.3f;

var originalTime = time;

if (isGoodAnswer)

c = GOOD\_COLOR;

while (time > 0.0f)

{

time -= Time.deltaTime;

BACKGROUND\_BACK.color = Color.Lerp(NORMAL\_COLOR, c, time / originalTime);

yield return 0;

}

}

// <summary>

/// If using Very Simple Leaderboard by App Advisory, report the score : http://u3d.as/qxf

/// </summary>

void ReportScoreToLeaderboard(int p)

{

#if APPADVISORY\_LEADERBOARD

LeaderboardManager.ReportScore(p);

#else

print("Get very simple leaderboard to use it : http://u3d.as/qxf");

#endif

}

/// <summary>

/// If using Very Simple Ads by App Advisory, show an interstitial if number of play > numberOfPlayToShowInterstitial: http://u3d.as/oWD

/// </summary>

public void ShowAds()

{

int count = PlayerPrefs.GetInt("GAMEOVER\_COUNT", 0);

count++;

PlayerPrefs.SetInt("GAMEOVER\_COUNT", count);

PlayerPrefs.Save();

#if APPADVISORY\_ADS

if(count > numberOfPlayToShowInterstitial)

{

#if UNITY\_EDITOR

print("count = " + count + " > numberOfPlayToShowINterstitial = " + numberOfPlayToShowInterstitial);

#endif

if(AdsManager.instance.IsReadyInterstitial())

{

#if UNITY\_EDITOR

print("AdsManager.instance.IsReadyInterstitial() == true ----> SO ====> set count = 0 AND show interstial");

#endif

PlayerPrefs.SetInt("GAMEOVER\_COUNT",0);

AdsManager.instance.ShowInterstitial();

}

else

{

#if UNITY\_EDITOR

print("AdsManager.instance.IsReadyInterstitial() == false");

#endif

}

}

else

{

PlayerPrefs.SetInt("GAMEOVER\_COUNT", count);

}

PlayerPrefs.Save();

#else

if (count >= numberOfPlayToShowInterstitial)

{

Debug.LogWarning("To show ads, please have a look to Very Simple Ad on the Asset Store, or go to this link: " + VerySimpleAdsURL);

Debug.LogWarning("Very Simple Ad is already implemented in this asset");

Debug.LogWarning("Just import the package and you are ready to use it and monetize your game!");

Debug.LogWarning("Very Simple Ad : " + VerySimpleAdsURL);

PlayerPrefs.SetInt("GAMEOVER\_COUNT", 0);

}

else

{

PlayerPrefs.SetInt("GAMEOVER\_COUNT", count);

}

PlayerPrefs.Save();

#endif

}

}

}

using UnityEngine;

using UnityEngine.UI;

using System.Collections;

namespace AppAdvisory.MathGame

{

public class MenuGameOver : MonoBehaviour

{

public Text level;

public Text score;

public Text bestScore;

public GameObject newBestScoreLabel;

void OnEnable(){

level.text = ScoreManager.GetLastLevel ().ToString ();

score.text = ScoreManager.GetLastScore ().ToString ();

bestScore.text = ScoreManager.GetBestScore ().ToString ();

bool isNewBest = ScoreManager.GetLastScoreIsBest ();

if (isNewBest) {

newBestScoreLabel.SetActive (true);

} else {

newBestScoreLabel.SetActive (false);

}

}

}

}

using UnityEngine;

using UnityEngine.UI;

using System.Collections;

namespace AppAdvisory.MathGame

{

public class MenuLogic : MonoBehaviour

{

public GameObject FirstTimeMenu;

public GameObject GameOverMenu;

public Transform Title;

public Text M;

public Text A;

public Text T;

public Text H;

public Text G;

public Text A\_;

public Text M\_;

public Text E;

bool firstTime;

void Awake()

{

firstTime = true;

}

public void OnEnable()

{

foreach (Transform t in Title)

{

t.localScale = Vector3.one;

}

FirstTimeMenu.SetActive (firstTime);

GameOverMenu.SetActive (!firstTime);

if (!firstTime)

{

M.text = "G";

A.text = "A";

T.text = "M";

H.text = "E";

G.text = "O";

A\_.text = "V";

M\_.text = "E";

E.text = "R";

}

firstTime = false;

}

public void OnDisable()

{

foreach (Transform t in Title)

{

t.localScale = Vector3.one;

}

}

public void CloseGame()

{

}

}

}

using UnityEngine;

using UnityEngine.UI;

using System;

using System.Linq;

using System.Collections;

using System.Collections.Generic;

namespace MenuBarouch

{

public class MenuManager : MonoBehaviour

{

public GameObject MENU;

public GameObject GAME;

public GameObject SETTING;

public Image BACKGROUND\_BACK;

public Color NORMAL\_COLOR;

void Awake()

{

MENU.SetActive (true);

GAME.SetActive (false);

SETTING.SetActive (false);

}

//open the game

public void GoToGame()

{

float time = 0.2f;

GoOut (MENU,time,0);

GoIn (GAME, time, time);

}

//open the menu

public void GoToMenu()

{

float time = 0.2f;

GoOut (GAME,time,0);

GoIn (MENU, time, time);

}

//open the setting menu

public void OpenSettings()

{

float time = 0.2f;

GoOut (MENU,time,0);

GoIn (SETTING, time, time);

}

//close the setting menu

public void CloseSettings()

{

float time = 0.2f;

GoOut (SETTING,time,0);

GoIn (MENU, time, time);

}

void Update()

{

BACKGROUND\_BACK.color = Color.Lerp(BACKGROUND\_BACK.color, NORMAL\_COLOR,Time.time);

}

//animation scale from 1 to 0

public void GoOut(GameObject obj, float time, float delay)

{

obj.transform.localScale = Vector3.one;

StartCoroutine (GoInOrOutCorout (obj, 0, time, delay, () => {

obj.transform.localScale = Vector3.zero;

obj.SetActive(false);

}));

}

//animation scale from 0 to 1

public void GoIn(GameObject obj, float time, float delay){

obj.transform.localScale = Vector3.zero;

StartCoroutine (GoInOrOutCorout (obj, 1, time, delay, () => {

obj.transform.localScale = Vector3.one;

obj.SetActive(true);

}));

}

//do the animation scale

IEnumerator GoInOrOutCorout(GameObject obj, float scale, float time, float delay, Action callback)

{

obj.SetActive(true);

yield return new WaitForSeconds (delay);

var originalScale = obj.transform.localScale;

var targetScale = Vector3.one \* scale;

var originalTime = time;

while (time > 0.0f)

{

time -= Time.deltaTime;

obj.transform.localScale = Vector3.Lerp(targetScale, originalScale, time / originalTime);

yield return 0;

}

if (callback != null)

callback ();

}

}

}

using UnityEngine;

using System;

using System.Collections;

using UnityEngine.UI;

namespace AppAdvisory.MathGame

{

public class RateUsManager : MonoBehaviour

{

public int NumberOfLevelPlayedToShowRateUs = 30;

public string iOSURL = "itms://itunes.apple.com/us/app/apple-store/id963781532?mt=8";

public string ANDROIDURL = "http://app-advisory.com";

public Button btnYes;

public Button btnLater;

public Button btnNever;

public CanvasGroup popupCanvasGroup;

void Awake()

{

popupCanvasGroup.alpha = 0;

popupCanvasGroup.gameObject.SetActive(false);

}

void OnEnable()

{

GameManager.OnGameOver += CheckIfPromptRateDialogue;

}

void OnDisable()

{

GameManager.OnGameOver -= CheckIfPromptRateDialogue;

}

void AddButtonListeners()

{

btnYes.onClick.AddListener(OnClickedYes);

btnLater.onClick.AddListener(OnClickedLater);

btnNever.onClick.AddListener(OnClickedNever);

}

void RemoveButtonListener()

{

btnYes.onClick.RemoveListener(OnClickedYes);

btnLater.onClick.RemoveListener(OnClickedLater);

btnNever.onClick.RemoveListener(OnClickedNever);

}

void OnClickedYes()

{

#if UNITY\_IPHONE

Application.OpenURL(iOSURL);

#endif

#if UNITY\_ANDROID

Application.OpenURL(ANDROIDURL);

#endif

PlayerPrefs.SetInt("NUMOFLEVELPLAYED",-1);

PlayerPrefs.Save();

HidePopup();

}

void OnClickedLater()

{

PlayerPrefs.SetInt("NUMOFLEVELPLAYED",0);

PlayerPrefs.Save();

HidePopup();

}

void OnClickedNever()

{

PlayerPrefs.SetInt("NUMOFLEVELPLAYED",-1);

PlayerPrefs.Save();

HidePopup();

}

void CheckIfPromptRateDialogue()

{

int count = PlayerPrefs.GetInt("NUMOFLEVELPLAYED",0);

if(count == -1)

return;

count ++;

if(count > NumberOfLevelPlayedToShowRateUs)

{

PromptPopup();

}

else

{

PlayerPrefs.SetInt("NUMOFLEVELPLAYED",count);

}

PlayerPrefs.Save();

}

public void PromptPopup()

{

popupCanvasGroup.alpha = 0;

popupCanvasGroup.gameObject.SetActive(true);

StartCoroutine(DoLerpAlpha(popupCanvasGroup, 0, 1, 1, () => {

AddButtonListeners();

}));

}

void HidePopup()

{

StartCoroutine(DoLerpAlpha(popupCanvasGroup, 1, 0, 1, () => {

popupCanvasGroup.gameObject.SetActive(false);

RemoveButtonListener();

}));

}

public IEnumerator DoLerpAlpha(CanvasGroup c, float from, float to, float time, Action callback)

{

float timer = 0;

c.alpha = from;

while (timer <= time)

{

timer += Time.deltaTime;

c.alpha = Mathf.Lerp(from, to, timer / time);

yield return null;

}

c.alpha = to;

if (callback != null)

callback ();

}

}

}

using UnityEngine;

using System.Collections;

namespace AppAdvisory.MathGame

{

public class ScoreManager

{

public static void SaveScore(int lastScore, int level)

{

PlayerPrefs.SetInt ("LAST\_SCORE", lastScore);

PlayerPrefs.SetInt ("LAST\_LEVEL", level);

int best = GetBestScore ();

if(lastScore > best)

PlayerPrefs.SetInt ("LAST\_SCORE\_IS\_NEW\_BEST", 1);

else

PlayerPrefs.SetInt ("LAST\_SCORE\_IS\_NEW\_BEST", 0);

if(lastScore > best)

PlayerPrefs.SetInt ("BEST\_SCORE", lastScore);

PlayerPrefs.Save ();

}

public static int GetLastScore()

{

return PlayerPrefs.GetInt ("LAST\_SCORE");

}

public static int GetLastLevel()

{

return PlayerPrefs.GetInt ("LAST\_LEVEL");

}

public static bool GetLastScoreIsBest()

{

int temp = PlayerPrefs.GetInt ("LAST\_SCORE\_IS\_NEW\_BEST");

if (temp == 1) {

return true;

}

return false;

}

public static int GetBestScore()

{

return PlayerPrefs.GetInt ("BEST\_SCORE");

}

}

}

using UnityEngine;

using UnityEngine.UI;

using System.Collections;

namespace AppAdvisory.MathGame

{

public class SliderFillArea : MonoBehaviour

{

public RectTransform rect;

void OnEnable(){

rect.anchoredPosition = Vector2.zero;

}

}

}

using UnityEngine;

using UnityEngine.UI;

using System.Collections;

using System.Collections.Generic;

namespace AppAdvisory.MathGame

{

public static class Util

{

private static System.Random Random = new System.Random();

public static float RandomValue()

{

return (float)Random.NextDouble();

}

public static float RandomRange(float min, float max)

{

return min + ((float)Random.NextDouble() \* (max - min));

}

public static int RandomRange(int min, int max)

{

return Random.Next(min, max);

}

}

}

using System;

using System.Collections;

using System.Runtime.CompilerServices;

using UnityEngine.Bindings;

using UnityEngine.Internal;

using UnityEngine.Scripting;

namespace UnityEngine;

//

// 摘要:

// MonoBehaviour is the base class from which every Unity script derives.

[RequiredByNativeCode]

[ExtensionOfNativeClass]

[NativeHeader("Runtime/Mono/MonoBehaviour.h")]

[NativeHeader("Runtime/Scripting/DelayedCallUtility.h")]

public class MonoBehaviour : Behaviour

{

//

// 摘要:

// Disabling this lets you skip the GUI layout phase.

public extern bool useGUILayout

{

[MethodImpl(MethodImplOptions.InternalCall)]

get;

[MethodImpl(MethodImplOptions.InternalCall)]

set;

}

//

// 摘要:

// Allow a specific instance of a MonoBehaviour to run in edit mode (only available

// in the editor).

public extern bool runInEditMode

{

[MethodImpl(MethodImplOptions.InternalCall)]

get;

[MethodImpl(MethodImplOptions.InternalCall)]

set;

}

internal extern bool allowPrefabModeInPlayMode

{

[MethodImpl(MethodImplOptions.InternalCall)]

get;

}

public MonoBehaviour()

{

ConstructorCheck(this);

}

//

// 摘要:

// Is any invoke pending on this MonoBehaviour?

public bool IsInvoking()

{

return Internal\_IsInvokingAll(this);

}

//

// 摘要:

// Cancels all Invoke calls on this MonoBehaviour.

public void CancelInvoke()

{

Internal\_CancelInvokeAll(this);

}

//

// 摘要:

// Invokes the method methodName in time seconds.

//

// 参数:

// methodName:

//

// time:

public void Invoke(string methodName, float time)

{

InvokeDelayed(this, methodName, time, 0f);

}

//

// 摘要:

// Invokes the method methodName in time seconds, then repeatedly every repeatRate

// seconds.

//

// 参数:

// methodName:

//

// time:

//

// repeatRate:

public void InvokeRepeating(string methodName, float time, float repeatRate)

{

if (repeatRate <= 1E-05f && repeatRate != 0f)

{

throw new UnityException("Invoke repeat rate has to be larger than 0.00001F)");

}

InvokeDelayed(this, methodName, time, repeatRate);

}

//

// 摘要:

// Cancels all Invoke calls with name methodName on this behaviour.

//

// 参数:

// methodName:

public void CancelInvoke(string methodName)

{

CancelInvoke(this, methodName);

}

//

// 摘要:

// Is any invoke on methodName pending?

//

// 参数:

// methodName:

public bool IsInvoking(string methodName)

{

return IsInvoking(this, methodName);

}

//

// 摘要:

// Starts a coroutine named methodName.

//

// 参数:

// methodName:

//

// value:

[ExcludeFromDocs]

public Coroutine StartCoroutine(string methodName)

{

object value = null;

return StartCoroutine(methodName, value);

}

//

// 摘要:

// Starts a coroutine named methodName.

//

// 参数:

// methodName:

//

// value:

public Coroutine StartCoroutine(string methodName, [DefaultValue("null")] object value)

{

if (string.IsNullOrEmpty(methodName))

{

throw new NullReferenceException("methodName is null or empty");

}

if (!IsObjectMonoBehaviour(this))

{

throw new ArgumentException("Coroutines can only be stopped on a MonoBehaviour");

}

return StartCoroutineManaged(methodName, value);

}

//

// 摘要:

// Starts a Coroutine.

//

// 参数:

// routine:

public Coroutine StartCoroutine(IEnumerator routine)

{

if (routine == null)

{

throw new NullReferenceException("routine is null");

}

if (!IsObjectMonoBehaviour(this))

{

throw new ArgumentException("Coroutines can only be stopped on a MonoBehaviour");

}

return StartCoroutineManaged2(routine);

}

[Obsolete("StartCoroutine\_Auto has been deprecated. Use StartCoroutine instead (UnityUpgradable) -> StartCoroutine([mscorlib] System.Collections.IEnumerator)", false)]

public Coroutine StartCoroutine\_Auto(IEnumerator routine)

{

return StartCoroutine(routine);

}

//

// 摘要:

// Stops the first coroutine named methodName, or the coroutine stored in routine

// running on this behaviour.

//

// 参数:

// methodName:

// Name of coroutine.

//

// routine:

// Name of the function in code, including coroutines.

public void StopCoroutine(IEnumerator routine)

{

if (routine == null)

{

throw new NullReferenceException("routine is null");

}

if (!IsObjectMonoBehaviour(this))

{

throw new ArgumentException("Coroutines can only be stopped on a MonoBehaviour");

}

StopCoroutineFromEnumeratorManaged(routine);

}

//

// 摘要:

// Stops the first coroutine named methodName, or the coroutine stored in routine

// running on this behaviour.

//

// 参数:

// methodName:

// Name of coroutine.

//

// routine:

// Name of the function in code, including coroutines.

public void StopCoroutine(Coroutine routine)

{

if (routine == null)

{

throw new NullReferenceException("routine is null");

}

if (!IsObjectMonoBehaviour(this))

{

throw new ArgumentException("Coroutines can only be stopped on a MonoBehaviour");

}

StopCoroutineManaged(routine);

}

//

// 摘要:

// Stops the first coroutine named methodName, or the coroutine stored in routine

// running on this behaviour.

//

// 参数:

// methodName:

// Name of coroutine.

//

// routine:

// Name of the function in code, including coroutines.

[MethodImpl(MethodImplOptions.InternalCall)]

public extern void StopCoroutine(string methodName);

//

// 摘要:

// Stops all coroutines running on this behaviour.

[MethodImpl(MethodImplOptions.InternalCall)]

public extern void StopAllCoroutines();

//

// 摘要:

// Logs message to the Unity Console (identical to Debug.Log).

//

// 参数:

// message:

public static void print(object message)

{

Debug.Log(message);

}

[MethodImpl(MethodImplOptions.InternalCall)]

[NativeMethod(IsThreadSafe = true)]

private static extern void ConstructorCheck([Writable] Object self);

[MethodImpl(MethodImplOptions.InternalCall)]

[FreeFunction("CancelInvoke")]

private static extern void Internal\_CancelInvokeAll([NotNull("NullExceptionObject")] MonoBehaviour self);

[MethodImpl(MethodImplOptions.InternalCall)]

[FreeFunction("IsInvoking")]

private static extern bool Internal\_IsInvokingAll([NotNull("NullExceptionObject")] MonoBehaviour self);

[MethodImpl(MethodImplOptions.InternalCall)]

[FreeFunction]

private static extern void InvokeDelayed([NotNull("NullExceptionObject")] MonoBehaviour self, string methodName, float time, float repeatRate);

[MethodImpl(MethodImplOptions.InternalCall)]

[FreeFunction]

private static extern void CancelInvoke([NotNull("NullExceptionObject")] MonoBehaviour self, string methodName);

[MethodImpl(MethodImplOptions.InternalCall)]

[FreeFunction]

private static extern bool IsInvoking([NotNull("NullExceptionObject")] MonoBehaviour self, string methodName);

[MethodImpl(MethodImplOptions.InternalCall)]

[FreeFunction]

private static extern bool IsObjectMonoBehaviour([NotNull("NullExceptionObject")] Object obj);

[MethodImpl(MethodImplOptions.InternalCall)]

private extern Coroutine StartCoroutineManaged(string methodName, object value);

[MethodImpl(MethodImplOptions.InternalCall)]

private extern Coroutine StartCoroutineManaged2(IEnumerator enumerator);

[MethodImpl(MethodImplOptions.InternalCall)]

private extern void StopCoroutineManaged(Coroutine routine);

[MethodImpl(MethodImplOptions.InternalCall)]

private extern void StopCoroutineFromEnumeratorManaged(IEnumerator routine);

[MethodImpl(MethodImplOptions.InternalCall)]

internal extern string GetScriptClassName();

}

using UnityEngine;

using UnityEngine.UI;

using System.Collections;

using MenuBarouch;

namespace AppAdvisory.MathGame

{

public class ButtonHelper : MonoBehaviour

{

MenuManager \_menuManager;

public MenuManager menuManager

{

get

{

if (\_menuManager == null)

\_menuManager = FindObjectOfType<MenuManager> ();

return \_menuManager;

}

}

GameManager \_gameManager;

public GameManager gameManager

{

get

{

if (\_gameManager == null)

\_gameManager = FindObjectOfType<GameManager> ();

return \_gameManager;

}

}

virtual public void OnClicked(){}

void OnEnable()

{

GetComponent<Button> ().onClick.AddListener (OnClicked);

}

void OnDisable()

{

RemoveListener();

}

public void RemoveListener()

{

GetComponent<Button> ().onClick.RemoveListener(OnClicked);

}

}

}

using System;

using System.Runtime.CompilerServices;

using System.Runtime.InteropServices;

using System.Security;

using UnityEngine.Bindings;

using UnityEngine.Internal;

using UnityEngine.Scripting;

using UnityEngineInternal;

namespace UnityEngine;

//

// 摘要:

// Base class for all objects Unity can reference.

[StructLayout(LayoutKind.Sequential)]

[RequiredByNativeCode(GenerateProxy = true)]

[NativeHeader("Runtime/GameCode/CloneObject.h")]

[NativeHeader("Runtime/SceneManager/SceneManager.h")]

[NativeHeader("Runtime/Export/Scripting/UnityEngineObject.bindings.h")]

public class Object

{

private IntPtr m\_CachedPtr;

private int m\_InstanceID;

private string m\_UnityRuntimeErrorString;

internal static int OffsetOfInstanceIDInCPlusPlusObject = -1;

private const string objectIsNullMessage = "The Object you want to instantiate is null.";

private const string cloneDestroyedMessage = "Instantiate failed because the clone was destroyed during creation. This can happen if DestroyImmediate is called in MonoBehaviour.Awake.";

//

// 摘要:

// The name of the object.

public string name

{

get

{

return GetName(this);

}

set

{

SetName(this, value);

}

}

//

// 摘要:

// Should the object be hidden, saved with the Scene or modifiable by the user?

public extern HideFlags hideFlags

{

[MethodImpl(MethodImplOptions.InternalCall)]

get;

[MethodImpl(MethodImplOptions.InternalCall)]

set;

}

//

// 摘要:

// Gets the instance ID of the object.

//

// 返回结果:

// Returns the instance ID of the object. When used to call the origin object, this

// method returns a positive value. When used to call the instance object, this

// method returns a negative value.

[SecuritySafeCritical]

public int GetInstanceID()

{

EnsureRunningOnMainThread();

return m\_InstanceID;

}

public override int GetHashCode()

{

return m\_InstanceID;

}

public override bool Equals(object other)

{

Object @object = other as Object;

if (@object == null && other != null && !(other is Object))

{

return false;

}

return CompareBaseObjects(this, @object);

}

public static implicit operator bool(Object exists)

{

return !CompareBaseObjects(exists, null);

}

private static bool CompareBaseObjects(Object lhs, Object rhs)

{

bool flag = (object)lhs == null;

bool flag2 = (object)rhs == null;

if (flag2 && flag)

{

return true;

}

if (flag2)

{

return !IsNativeObjectAlive(lhs);

}

if (flag)

{

return !IsNativeObjectAlive(rhs);

}

return lhs.m\_InstanceID == rhs.m\_InstanceID;

}

private void EnsureRunningOnMainThread()

{

if (!CurrentThreadIsMainThread())

{

throw new InvalidOperationException("EnsureRunningOnMainThread can only be called from the main thread");

}

}

private static bool IsNativeObjectAlive(Object o)

{

if (o.GetCachedPtr() != IntPtr.Zero)

{

return true;

}

if (o is MonoBehaviour || o is ScriptableObject)

{

return false;

}

return DoesObjectWithInstanceIDExist(o.GetInstanceID());

}

private IntPtr GetCachedPtr()

{

return m\_CachedPtr;

}

//

// 摘要:

// Clones the object original and returns the clone.

//

// 参数:

// original:

// An existing object that you want to make a copy of.

//

// position:

// Position for the new object.

//

// rotation:

// Orientation of the new object.

//

// parent:

// Parent that will be assigned to the new object.

//

// instantiateInWorldSpace:

// When you assign a parent Object, pass true to position the new object directly

// in world space. Pass false to set the Object’s position relative to its new parent.

//

//

// 返回结果:

// The instantiated clone.

[TypeInferenceRule(TypeInferenceRules.TypeOfFirstArgument)]

public static Object Instantiate(Object original, Vector3 position, Quaternion rotation)

{

CheckNullArgument(original, "The Object you want to instantiate is null.");

if (original is ScriptableObject)

{

throw new ArgumentException("Cannot instantiate a ScriptableObject with a position and rotation");

}

Object @object = Internal\_InstantiateSingle(original, position, rotation);

if (@object == null)

{

throw new UnityException("Instantiate failed because the clone was destroyed during creation. This can happen if DestroyImmediate is called in MonoBehaviour.Awake.");

}

return @object;

}

//

// 摘要:

// Clones the object original and returns the clone.

//

// 参数:

// original:

// An existing object that you want to make a copy of.

//

// position:

// Position for the new object.

//

// rotation:

// Orientation of the new object.

//

// parent:

// Parent that will be assigned to the new object.

//

// instantiateInWorldSpace:

// When you assign a parent Object, pass true to position the new object directly

// in world space. Pass false to set the Object’s position relative to its new parent.

//

//

// 返回结果:

// The instantiated clone.

[TypeInferenceRule(TypeInferenceRules.TypeOfFirstArgument)]

public static Object Instantiate(Object original, Vector3 position, Quaternion rotation, Transform parent)

{

if (parent == null)

{

return Instantiate(original, position, rotation);

}

CheckNullArgument(original, "The Object you want to instantiate is null.");

Object @object = Internal\_InstantiateSingleWithParent(original, parent, position, rotation);

if (@object == null)

{

throw new UnityException("Instantiate failed because the clone was destroyed during creation. This can happen if DestroyImmediate is called in MonoBehaviour.Awake.");

}

return @object;

}

//

// 摘要:

// Clones the object original and returns the clone.

//

// 参数:

// original:

// An existing object that you want to make a copy of.

//

// position:

// Position for the new object.

//

// rotation:

// Orientation of the new object.

//

// parent:

// Parent that will be assigned to the new object.

//

// instantiateInWorldSpace:

// When you assign a parent Object, pass true to position the new object directly

// in world space. Pass false to set the Object’s position relative to its new parent.

//

//

// 返回结果:

// The instantiated clone.

[TypeInferenceRule(TypeInferenceRules.TypeOfFirstArgument)]

public static Object Instantiate(Object original)

{

CheckNullArgument(original, "The Object you want to instantiate is null.");

Object @object = Internal\_CloneSingle(original);

if (@object == null)

{

throw new UnityException("Instantiate failed because the clone was destroyed during creation. This can happen if DestroyImmediate is called in MonoBehaviour.Awake.");

}

return @object;

}

//

// 摘要:

// Clones the object original and returns the clone.

//

// 参数:

// original:

// An existing object that you want to make a copy of.

//

// position:

// Position for the new object.

//

// rotation:

// Orientation of the new object.

//

// parent:

// Parent that will be assigned to the new object.

//

// instantiateInWorldSpace:

// When you assign a parent Object, pass true to position the new object directly

// in world space. Pass false to set the Object’s position relative to its new parent.

//

//

// 返回结果:

// The instantiated clone.

[TypeInferenceRule(TypeInferenceRules.TypeOfFirstArgument)]

public static Object Instantiate(Object original, Transform parent)

{

return Instantiate(original, parent, instantiateInWorldSpace: false);

}

//

// 摘要:

// Clones the object original and returns the clone.

//

// 参数:

// original:

// An existing object that you want to make a copy of.

//

// position:

// Position for the new object.

//

// rotation:

// Orientation of the new object.

//

// parent:

// Parent that will be assigned to the new object.

//

// instantiateInWorldSpace:

// When you assign a parent Object, pass true to position the new object directly

// in world space. Pass false to set the Object’s position relative to its new parent.

//

//

// 返回结果:

// The instantiated clone.

[TypeInferenceRule(TypeInferenceRules.TypeOfFirstArgument)]

public static Object Instantiate(Object original, Transform parent, bool instantiateInWorldSpace)

{

if (parent == null)

{

return Instantiate(original);

}

CheckNullArgument(original, "The Object you want to instantiate is null.");

Object @object = Internal\_CloneSingleWithParent(original, parent, instantiateInWorldSpace);

if (@object == null)

{

throw new UnityException("Instantiate failed because the clone was destroyed during creation. This can happen if DestroyImmediate is called in MonoBehaviour.Awake.");

}

return @object;

}

public static T Instantiate<T>(T original) where T : Object

{

CheckNullArgument(original, "The Object you want to instantiate is null.");

T val = (T)Internal\_CloneSingle(original);

if ((Object)val == (Object)null)

{

throw new UnityException("Instantiate failed because the clone was destroyed during creation. This can happen if DestroyImmediate is called in MonoBehaviour.Awake.");

}

return val;

}

public static T Instantiate<T>(T original, Vector3 position, Quaternion rotation) where T : Object

{

return (T)Instantiate((Object)original, position, rotation);

}

public static T Instantiate<T>(T original, Vector3 position, Quaternion rotation, Transform parent) where T : Object

{

return (T)Instantiate((Object)original, position, rotation, parent);

}

public static T Instantiate<T>(T original, Transform parent) where T : Object

{

return Instantiate(original, parent, worldPositionStays: false);

}

public static T Instantiate<T>(T original, Transform parent, bool worldPositionStays) where T : Object

{

return (T)Instantiate((Object)original, parent, worldPositionStays);

}

//

// 摘要:

// Removes a GameObject, component or asset.

//

// 参数:

// obj:

// The object to destroy.

//

// t:

// The optional amount of time to delay before destroying the object.

[MethodImpl(MethodImplOptions.InternalCall)]

[NativeMethod(Name = "Scripting::DestroyObjectFromScripting", IsFreeFunction = true, ThrowsException = true)]

public static extern void Destroy(Object obj, [DefaultValue("0.0F")] float t);

//

// 摘要:

// Removes a GameObject, component or asset.

//

// 参数:

// obj:

// The object to destroy.

//

// t:

// The optional amount of time to delay before destroying the object.

[ExcludeFromDocs]

public static void Destroy(Object obj)

{

float t = 0f;

Destroy(obj, t);

}

//

// 摘要:

// Destroys the object obj immediately. You are strongly recommended to use Destroy

// instead.

//

// 参数:

// obj:

// Object to be destroyed.

//

// allowDestroyingAssets:

// Set to true to allow assets to be destroyed.

[MethodImpl(MethodImplOptions.InternalCall)]

[NativeMethod(Name = "Scripting::DestroyObjectFromScriptingImmediate", IsFreeFunction = true, ThrowsException = true)]

public static extern void DestroyImmediate(Object obj, [DefaultValue("false")] bool allowDestroyingAssets);

//

// 摘要:

// Destroys the object obj immediately. You are strongly recommended to use Destroy

// instead.

//

// 参数:

// obj:

// Object to be destroyed.

//

// allowDestroyingAssets:

// Set to true to allow assets to be destroyed.

[ExcludeFromDocs]

public static void DestroyImmediate(Object obj)

{

bool allowDestroyingAssets = false;

DestroyImmediate(obj, allowDestroyingAssets);

}

//

// 摘要:

// Gets a list of all loaded objects of Type type.

//

// 参数:

// type:

// The type of object to find.

//

// includeInactive:

// If true, components attached to inactive GameObjects are also included.

//

// 返回结果:

// The array of objects found matching the type specified.

public static Object[] FindObjectsOfType(Type type)

{

return FindObjectsOfType(type, includeInactive: false);

}

//

// 摘要:

// Gets a list of all loaded objects of Type type.

//

// 参数:

// type:

// The type of object to find.

//

// includeInactive:

// If true, components attached to inactive GameObjects are also included.

//

// 返回结果:

// The array of objects found matching the type specified.

[MethodImpl(MethodImplOptions.InternalCall)]

[FreeFunction("UnityEngineObjectBindings::FindObjectsOfType")]

[TypeInferenceRule(TypeInferenceRules.ArrayOfTypeReferencedByFirstArgument)]

public static extern Object[] FindObjectsOfType(Type type, bool includeInactive);

//

// 摘要:

// Retrieves a list of all loaded objects of Type type.

//

// 参数:

// type:

// The type of object to find.

//

// findObjectsInactive:

// Whether to include components attached to inactive GameObjects. If you don't

// specify this parameter, this function doesn't include inactive objects in the

// results.

//

// sortMode:

// Whether and how to sort the returned array. Not sorting the array makes this

// function run significantly faster.

//

// 返回结果:

// The array of objects found matching the type specified.

public static Object[] FindObjectsByType(Type type, FindObjectsSortMode sortMode)

{

return FindObjectsByType(type, FindObjectsInactive.Exclude, sortMode);

}

//

// 摘要:

// Retrieves a list of all loaded objects of Type type.

//

// 参数:

// type:

// The type of object to find.

//

// findObjectsInactive:

// Whether to include components attached to inactive GameObjects. If you don't

// specify this parameter, this function doesn't include inactive objects in the

// results.

//

// sortMode:

// Whether and how to sort the returned array. Not sorting the array makes this

// function run significantly faster.

//

// 返回结果:

// The array of objects found matching the type specified.

[MethodImpl(MethodImplOptions.InternalCall)]

[TypeInferenceRule(TypeInferenceRules.ArrayOfTypeReferencedByFirstArgument)]

[FreeFunction("UnityEngineObjectBindings::FindObjectsByType")]

public static extern Object[] FindObjectsByType(Type type, FindObjectsInactive findObjectsInactive, FindObjectsSortMode sortMode);

//

// 摘要:

// Do not destroy the target Object when loading a new Scene.

//

// 参数:

// target:

// An Object not destroyed on Scene change.

[MethodImpl(MethodImplOptions.InternalCall)]

[FreeFunction("GetSceneManager().DontDestroyOnLoad", ThrowsException = true)]

public static extern void DontDestroyOnLoad([NotNull("NullExceptionObject")] Object target);

[Obsolete("use Object.Destroy instead.")]

public static void DestroyObject(Object obj, [DefaultValue("0.0F")] float t)

{

Destroy(obj, t);

}

[Obsolete("use Object.Destroy instead.")]

[ExcludeFromDocs]

public static void DestroyObject(Object obj)

{

float t = 0f;

Destroy(obj, t);

}

[Obsolete("warning use Object.FindObjectsByType instead.")]

public static Object[] FindSceneObjectsOfType(Type type)

{

return FindObjectsOfType(type);

}

//

// 摘要:

// Returns a list of all active and inactive loaded objects of Type type, including

// assets.

//

// 参数:

// type:

// The type of object or asset to find.

//

// 返回结果:

// The array of objects and assets found matching the type specified.

[MethodImpl(MethodImplOptions.InternalCall)]

[Obsolete("use Resources.FindObjectsOfTypeAll instead.")]

[FreeFunction("UnityEngineObjectBindings::FindObjectsOfTypeIncludingAssets")]

public static extern Object[] FindObjectsOfTypeIncludingAssets(Type type);

public static T[] FindObjectsOfType<T>() where T : Object

{

return Resources.ConvertObjects<T>(FindObjectsOfType(typeof(T), includeInactive: false));

}

public static T[] FindObjectsByType<T>(FindObjectsSortMode sortMode) where T : Object

{

return Resources.ConvertObjects<T>(FindObjectsByType(typeof(T), FindObjectsInactive.Exclude, sortMode));

}

public static T[] FindObjectsOfType<T>(bool includeInactive) where T : Object

{

return Resources.ConvertObjects<T>(FindObjectsOfType(typeof(T), includeInactive));

}

public static T[] FindObjectsByType<T>(FindObjectsInactive findObjectsInactive, FindObjectsSortMode sortMode) where T : Object

{

return Resources.ConvertObjects<T>(FindObjectsByType(typeof(T), findObjectsInactive, sortMode));

}

public static T FindObjectOfType<T>() where T : Object

{

return (T)FindObjectOfType(typeof(T), includeInactive: false);

}

public static T FindObjectOfType<T>(bool includeInactive) where T : Object

{

return (T)FindObjectOfType(typeof(T), includeInactive);

}

public static T FindFirstObjectByType<T>() where T : Object

{

return (T)FindFirstObjectByType(typeof(T), FindObjectsInactive.Exclude);

}

public static T FindAnyObjectByType<T>() where T : Object

{

return (T)FindAnyObjectByType(typeof(T), FindObjectsInactive.Exclude);

}

public static T FindFirstObjectByType<T>(FindObjectsInactive findObjectsInactive) where T : Object

{

return (T)FindFirstObjectByType(typeof(T), findObjectsInactive);

}

public static T FindAnyObjectByType<T>(FindObjectsInactive findObjectsInactive) where T : Object

{

return (T)FindAnyObjectByType(typeof(T), findObjectsInactive);

}

//

// 摘要:

// Returns a list of all active and inactive loaded objects of Type type.

//

// 参数:

// type:

// The type of object to find.

//

// 返回结果:

// The array of objects found matching the type specified.

[Obsolete("Please use Resources.FindObjectsOfTypeAll instead")]

public static Object[] FindObjectsOfTypeAll(Type type)

{

return Resources.FindObjectsOfTypeAll(type);

}

private static void CheckNullArgument(object arg, string message)

{

if (arg == null)

{

throw new ArgumentException(message);

}

}

//

// 摘要:

// Returns the first active loaded object of Type type.

//

// 参数:

// type:

// The type of object to find.

//

// includeInactive:

//

// 返回结果:

// Object The first active loaded object that matches the specified type. It returns

// null if no Object matches the type.

[TypeInferenceRule(TypeInferenceRules.TypeReferencedByFirstArgument)]

public static Object FindObjectOfType(Type type)

{

Object[] array = FindObjectsOfType(type, includeInactive: false);

if (array.Length != 0)

{

return array[0];

}

return null;

}

//

// 摘要:

// Retrieves the first active loaded object of Type type.

//

// 参数:

// type:

// The type of object to find.

//

// findObjectsInactive:

// Whether to include components attached to inactive GameObjects. If you don't

// specify this parameter, this function doesn't include inactive objects in the

// results.

//

// 返回结果:

// Returns the first active loaded object that matches the specified type. If no

// object matches the specified type, returns null.

public static Object FindFirstObjectByType(Type type)

{

Object[] array = FindObjectsByType(type, FindObjectsInactive.Exclude, FindObjectsSortMode.InstanceID);

return (array.Length != 0) ? array[0] : null;

}

//

// 摘要:

// Retrieves any active loaded object of Type type.

//

// 参数:

// type:

// The type of object to find.

//

// findObjectsInactive:

// Whether to include components attached to inactive GameObjects. If you don't

// specify this parameter, this function doesn't include inactive objects in the

// results.

//

// 返回结果:

// Returns an arbitrary active loaded object that matches the specified type. If

// no object matches the specified type, returns null.

public static Object FindAnyObjectByType(Type type)

{

Object[] array = FindObjectsByType(type, FindObjectsInactive.Exclude, FindObjectsSortMode.None);

return (array.Length != 0) ? array[0] : null;

}

//

// 摘要:

// Returns the first active loaded object of Type type.

//

// 参数:

// type:

// The type of object to find.

//

// includeInactive:

//

// 返回结果:

// Object The first active loaded object that matches the specified type. It returns

// null if no Object matches the type.

[TypeInferenceRule(TypeInferenceRules.TypeReferencedByFirstArgument)]

public static Object FindObjectOfType(Type type, bool includeInactive)

{

Object[] array = FindObjectsOfType(type, includeInactive);

if (array.Length != 0)

{

return array[0];

}

return null;

}

//

// 摘要:

// Retrieves the first active loaded object of Type type.

//

// 参数:

// type:

// The type of object to find.

//

// findObjectsInactive:

// Whether to include components attached to inactive GameObjects. If you don't

// specify this parameter, this function doesn't include inactive objects in the

// results.

//

// 返回结果:

// Returns the first active loaded object that matches the specified type. If no

// object matches the specified type, returns null.

public static Object FindFirstObjectByType(Type type, FindObjectsInactive findObjectsInactive)

{

Object[] array = FindObjectsByType(type, findObjectsInactive, FindObjectsSortMode.InstanceID);

return (array.Length != 0) ? array[0] : null;

}

//

// 摘要:

// Retrieves any active loaded object of Type type.

//

// 参数:

// type:

// The type of object to find.

//

// findObjectsInactive:

// Whether to include components attached to inactive GameObjects. If you don't

// specify this parameter, this function doesn't include inactive objects in the

// results.

//

// 返回结果:

// Returns an arbitrary active loaded object that matches the specified type. If

// no object matches the specified type, returns null.

public static Object FindAnyObjectByType(Type type, FindObjectsInactive findObjectsInactive)

{

Object[] array = FindObjectsByType(type, findObjectsInactive, FindObjectsSortMode.None);

return (array.Length != 0) ? array[0] : null;

}

//

// 摘要:

// Returns the name of the object.

//

// 返回结果:

// The name returned by ToString.

public override string ToString()

{

return ToString(this);

}

public static bool operator ==(Object x, Object y)

{

return CompareBaseObjects(x, y);

}

public static bool operator !=(Object x, Object y)

{

return !CompareBaseObjects(x, y);

}

[MethodImpl(MethodImplOptions.InternalCall)]

[NativeMethod(Name = "Object::GetOffsetOfInstanceIdMember", IsFreeFunction = true, IsThreadSafe = true)]

private static extern int GetOffsetOfInstanceIDInCPlusPlusObject();

[MethodImpl(MethodImplOptions.InternalCall)]

[NativeMethod(Name = "CurrentThreadIsMainThread", IsFreeFunction = true, IsThreadSafe = true)]

private static extern bool CurrentThreadIsMainThread();

[MethodImpl(MethodImplOptions.InternalCall)]

[NativeMethod(Name = "CloneObject", IsFreeFunction = true, ThrowsException = true)]

private static extern Object Internal\_CloneSingle([NotNull("NullExceptionObject")] Object data);

[MethodImpl(MethodImplOptions.InternalCall)]

[FreeFunction("CloneObject")]

private static extern Object Internal\_CloneSingleWithParent([NotNull("NullExceptionObject")] Object data, [NotNull("NullExceptionObject")] Transform parent, bool worldPositionStays);

[FreeFunction("InstantiateObject")]

private static Object Internal\_InstantiateSingle([NotNull("NullExceptionObject")] Object data, Vector3 pos, Quaternion rot)

{

return Internal\_InstantiateSingle\_Injected(data, ref pos, ref rot);

}

[FreeFunction("InstantiateObject")]

private static Object Internal\_InstantiateSingleWithParent([NotNull("NullExceptionObject")] Object data, [NotNull("NullExceptionObject")] Transform parent, Vector3 pos, Quaternion rot)

{

return Internal\_InstantiateSingleWithParent\_Injected(data, parent, ref pos, ref rot);

}

[MethodImpl(MethodImplOptions.InternalCall)]

[FreeFunction("UnityEngineObjectBindings::ToString")]

private static extern string ToString(Object obj);

[MethodImpl(MethodImplOptions.InternalCall)]

[FreeFunction("UnityEngineObjectBindings::GetName")]

private static extern string GetName([NotNull("NullExceptionObject")] Object obj);

[MethodImpl(MethodImplOptions.InternalCall)]

[FreeFunction("UnityEngineObjectBindings::IsPersistent")]

internal static extern bool IsPersistent([NotNull("NullExceptionObject")] Object obj);

[MethodImpl(MethodImplOptions.InternalCall)]

[FreeFunction("UnityEngineObjectBindings::SetName")]

private static extern void SetName([NotNull("NullExceptionObject")] Object obj, string name);

[MethodImpl(MethodImplOptions.InternalCall)]

[NativeMethod(Name = "UnityEngineObjectBindings::DoesObjectWithInstanceIDExist", IsFreeFunction = true, IsThreadSafe = true)]

internal static extern bool DoesObjectWithInstanceIDExist(int instanceID);

[MethodImpl(MethodImplOptions.InternalCall)]

[FreeFunction("UnityEngineObjectBindings::FindObjectFromInstanceID")]

[VisibleToOtherModules]

internal static extern Object FindObjectFromInstanceID(int instanceID);

[MethodImpl(MethodImplOptions.InternalCall)]

[FreeFunction("UnityEngineObjectBindings::ForceLoadFromInstanceID")]

[VisibleToOtherModules]

internal static extern Object ForceLoadFromInstanceID(int instanceID);

[MethodImpl(MethodImplOptions.InternalCall)]

private static extern Object Internal\_InstantiateSingle\_Injected(Object data, ref Vector3 pos, ref Quaternion rot);

[MethodImpl(MethodImplOptions.InternalCall)]

private static extern Object Internal\_InstantiateSingleWithParent\_Injected(Object data, Transform parent, ref Vector3 pos, ref Quaternion rot);

}

namespace System.Runtime.CompilerServices

{

[AttributeUsage(AttributeTargets.Constructor | AttributeTargets.Method, Inherited = false)]

public sealed class MethodImplAttribute : Attribute

{

public MethodCodeType MethodCodeType;

public MethodImplAttribute();

public MethodImplAttribute(short value);

public MethodImplAttribute(MethodImplOptions methodImplOptions);

public MethodImplOptions Value { get; }

}

using System.Runtime.CompilerServices;

using UnityEngine.Bindings;

using UnityEngine.Scripting;

namespace UnityEngine;

//

// 摘要:

// Position, size, anchor and pivot information for a rectangle.

[NativeHeader("Runtime/Transform/RectTransform.h")]

[NativeClass("UI::RectTransform")]

public sealed class RectTransform : Transform

{

//

// 摘要:

// Enum used to specify one edge of a rectangle.

public enum Edge

{

//

// 摘要:

// The left edge.

Left,

//

// 摘要:

// The right edge.

Right,

//

// 摘要:

// The top edge.

Top,

//

// 摘要:

// The bottom edge.

Bottom

}

//

// 摘要:

// An axis that can be horizontal or vertical.

public enum Axis

{

//

// 摘要:

// Horizontal.

Horizontal,

//

// 摘要:

// Vertical.

Vertical

}

//

// 摘要:

// Delegate used for the reapplyDrivenProperties event.

//

// 参数:

// driven:

public delegate void ReapplyDrivenProperties(RectTransform driven);

//

// 摘要:

// The calculated rectangle in the local space of the Transform.

public Rect rect

{

get

{

get\_rect\_Injected(out var ret);

return ret;

}

}

//

// 摘要:

// The normalized position in the parent RectTransform that the lower left corner

// is anchored to.

public Vector2 anchorMin

{

get

{

get\_anchorMin\_Injected(out var ret);

return ret;

}

set

{

set\_anchorMin\_Injected(ref value);

}

}

//

// 摘要:

// The normalized position in the parent RectTransform that the upper right corner

// is anchored to.

public Vector2 anchorMax

{

get

{

get\_anchorMax\_Injected(out var ret);

return ret;

}

set

{

set\_anchorMax\_Injected(ref value);

}

}

//

// 摘要:

// The position of the pivot of this RectTransform relative to the anchor reference

// point.

public Vector2 anchoredPosition

{

get

{

get\_anchoredPosition\_Injected(out var ret);

return ret;

}

set

{

set\_anchoredPosition\_Injected(ref value);

}

}

//

// 摘要:

// The size of this RectTransform relative to the distances between the anchors.

public Vector2 sizeDelta

{

get

{

get\_sizeDelta\_Injected(out var ret);

return ret;

}

set

{

set\_sizeDelta\_Injected(ref value);

}

}

//

// 摘要:

// The normalized position in this RectTransform that it rotates around.

public Vector2 pivot

{

get

{

get\_pivot\_Injected(out var ret);

return ret;

}

set

{

set\_pivot\_Injected(ref value);

}

}

//

// 摘要:

// The 3D position of the pivot of this RectTransform relative to the anchor reference

// point.

public Vector3 anchoredPosition3D

{

get

{

Vector2 vector = anchoredPosition;

return new Vector3(vector.x, vector.y, base.localPosition.z);

}

set

{

anchoredPosition = new Vector2(value.x, value.y);

Vector3 vector = base.localPosition;

vector.z = value.z;

base.localPosition = vector;

}

}

//

// 摘要:

// The offset of the lower left corner of the rectangle relative to the lower left

// anchor.

public Vector2 offsetMin

{

get

{

return anchoredPosition - Vector2.Scale(sizeDelta, pivot);

}

set

{

Vector2 vector = value - (anchoredPosition - Vector2.Scale(sizeDelta, pivot));

sizeDelta -= vector;

anchoredPosition += Vector2.Scale(vector, Vector2.one - pivot);

}

}

//

// 摘要:

// The offset of the upper right corner of the rectangle relative to the upper right

// anchor.

public Vector2 offsetMax

{

get

{

return anchoredPosition + Vector2.Scale(sizeDelta, Vector2.one - pivot);

}

set

{

Vector2 vector = value - (anchoredPosition + Vector2.Scale(sizeDelta, Vector2.one - pivot));

sizeDelta += vector;

anchoredPosition += Vector2.Scale(vector, pivot);

}

}

//

// 摘要:

// The object that is driving the values of this RectTransform. Value is null if

// not driven.

public extern Object drivenByObject

{

[MethodImpl(MethodImplOptions.InternalCall)]

get;

[MethodImpl(MethodImplOptions.InternalCall)]

internal set;

}

internal extern DrivenTransformProperties drivenProperties

{

[MethodImpl(MethodImplOptions.InternalCall)]

get;

[MethodImpl(MethodImplOptions.InternalCall)]

set;

}

public static event ReapplyDrivenProperties reapplyDrivenProperties;

//

// 摘要:

// Force the recalculation of RectTransforms internal data.

[MethodImpl(MethodImplOptions.InternalCall)]

[NativeMethod("UpdateIfTransformDispatchIsDirty")]

public extern void ForceUpdateRectTransforms();

//

// 摘要:

// Get the corners of the calculated rectangle in the local space of its Transform.

//

//

// 参数:

// fourCornersArray:

// The array that corners are filled into.

public void GetLocalCorners(Vector3[] fourCornersArray)

{

if (fourCornersArray == null || fourCornersArray.Length < 4)

{

Debug.LogError("Calling GetLocalCorners with an array that is null or has less than 4 elements.");

return;

}

Rect rect = this.rect;

float x = rect.x;

float y = rect.y;

float xMax = rect.xMax;

float yMax = rect.yMax;

fourCornersArray[0] = new Vector3(x, y, 0f);

fourCornersArray[1] = new Vector3(x, yMax, 0f);

fourCornersArray[2] = new Vector3(xMax, yMax, 0f);

fourCornersArray[3] = new Vector3(xMax, y, 0f);

}

//

// 摘要:

// Get the corners of the calculated rectangle in world space.

//

// 参数:

// fourCornersArray:

// The array that corners are filled into.

public void GetWorldCorners(Vector3[] fourCornersArray)

{

if (fourCornersArray == null || fourCornersArray.Length < 4)

{

Debug.LogError("Calling GetWorldCorners with an array that is null or has less than 4 elements.");

return;

}

GetLocalCorners(fourCornersArray);

Matrix4x4 matrix4x = base.transform.localToWorldMatrix;

for (int i = 0; i < 4; i++)

{

fourCornersArray[i] = matrix4x.MultiplyPoint(fourCornersArray[i]);

}

}

public void SetInsetAndSizeFromParentEdge(Edge edge, float inset, float size)

{

int index = ((edge == Edge.Top || edge == Edge.Bottom) ? 1 : 0);

bool flag = edge == Edge.Top || edge == Edge.Right;

float value = (flag ? 1 : 0);

Vector2 vector = anchorMin;

vector[index] = value;

anchorMin = vector;

vector = anchorMax;

vector[index] = value;

anchorMax = vector;

Vector2 vector2 = sizeDelta;

vector2[index] = size;

sizeDelta = vector2;

Vector2 vector3 = anchoredPosition;

vector3[index] = (flag ? (0f - inset - size \* (1f - pivot[index])) : (inset + size \* pivot[index]));

anchoredPosition = vector3;

}

public void SetSizeWithCurrentAnchors(Axis axis, float size)

{

Vector2 vector = sizeDelta;

vector[(int)axis] = size - GetParentSize()[(int)axis] \* (anchorMax[(int)axis] - anchorMin[(int)axis]);

sizeDelta = vector;

}

[RequiredByNativeCode]

internal static void SendReapplyDrivenProperties(RectTransform driven)

{

RectTransform.reapplyDrivenProperties?.Invoke(driven);

}

internal Rect GetRectInParentSpace()

{

Rect result = rect;

Vector2 vector = offsetMin + Vector2.Scale(pivot, result.size);

if ((bool)base.transform.parent)

{

RectTransform component = base.transform.parent.GetComponent<RectTransform>();

if ((bool)component)

{

vector += Vector2.Scale(anchorMin, component.rect.size);

}

}

result.x += vector.x;

result.y += vector.y;

return result;

}

private Vector2 GetParentSize()

{

RectTransform rectTransform = base.parent as RectTransform;

if (!rectTransform)

{

return Vector2.zero;

}

return rectTransform.rect.size;

}

[MethodImpl(MethodImplOptions.InternalCall)]

[SpecialName]

private extern void get\_rect\_Injected(out Rect ret);

[MethodImpl(MethodImplOptions.InternalCall)]

[SpecialName]

private extern void get\_anchorMin\_Injected(out Vector2 ret);

[MethodImpl(MethodImplOptions.InternalCall)]

[SpecialName]

private extern void set\_anchorMin\_Injected(ref Vector2 value);

[MethodImpl(MethodImplOptions.InternalCall)]

[SpecialName]

private extern void get\_anchorMax\_Injected(out Vector2 ret);

[MethodImpl(MethodImplOptions.InternalCall)]

[SpecialName]

private extern void set\_anchorMax\_Injected(ref Vector2 value);

[MethodImpl(MethodImplOptions.InternalCall)]

[SpecialName]

private extern void get\_anchoredPosition\_Injected(out Vector2 ret);

[MethodImpl(MethodImplOptions.InternalCall)]

[SpecialName]

private extern void set\_anchoredPosition\_Injected(ref Vector2 value);

[MethodImpl(MethodImplOptions.InternalCall)]

[SpecialName]

private extern void get\_sizeDelta\_Injected(out Vector2 ret);

[MethodImpl(MethodImplOptions.InternalCall)]

[SpecialName]

private extern void set\_sizeDelta\_Injected(ref Vector2 value);

[MethodImpl(MethodImplOptions.InternalCall)]

[SpecialName]

private extern void get\_pivot\_Injected(out Vector2 ret);

[MethodImpl(MethodImplOptions.InternalCall)]

[SpecialName]

private extern void set\_pivot\_Injected(ref Vector2 value);

}

using System;

using System.Collections;

using UnityEngine.Events;

using UnityEngine.EventSystems;

using UnityEngine.Serialization;

namespace UnityEngine.UI

{

/// <summary>

/// A standard button that sends an event when clicked.

/// </summary>

[AddComponentMenu("UI/Button", 30)]

public class Button : Selectable, IPointerClickHandler, ISubmitHandler

{

[Serializable]

/// <summary>

/// Function definition for a button click event.

/// </summary>

public class ButtonClickedEvent : UnityEvent {}

// Event delegates triggered on click.

[FormerlySerializedAs("onClick")]

[SerializeField]

private ButtonClickedEvent m\_OnClick = new ButtonClickedEvent();

protected Button()

{}

/// <summary>

/// UnityEvent that is triggered when the button is pressed.

/// Note: Triggered on MouseUp after MouseDown on the same object.

/// </summary>

///<example>

///<code>

/// <![CDATA[

/// using UnityEngine;

/// using UnityEngine.UI;

/// using System.Collections;

///

/// public class ClickExample : MonoBehaviour

/// {

/// public Button yourButton;

///

/// void Start()

/// {

/// Button btn = yourButton.GetComponent<Button>();

/// btn.onClick.AddListener(TaskOnClick);

/// }

///

/// void TaskOnClick()

/// {

/// Debug.Log("You have clicked the button!");

/// }

/// }

/// ]]>

///</code>

///</example>

public ButtonClickedEvent onClick

{

get { return m\_OnClick; }

set { m\_OnClick = value; }

}

private void Press()

{

if (!IsActive() || !IsInteractable())

return;

UISystemProfilerApi.AddMarker("Button.onClick", this);

m\_OnClick.Invoke();

}

/// <summary>

/// Call all registered IPointerClickHandlers.

/// Register button presses using the IPointerClickHandler. You can also use it to tell what type of click happened (left, right etc.).

/// Make sure your Scene has an EventSystem.

/// </summary>

/// <param name="eventData">Pointer Data associated with the event. Typically by the event system.</param>

/// <example>

/// <code>

/// <![CDATA[

/// //Attatch this script to a Button GameObject

/// using UnityEngine;

/// using UnityEngine.EventSystems;

///

/// public class Example : MonoBehaviour, IPointerClickHandler

/// {

/// //Detect if a click occurs

/// public void OnPointerClick(PointerEventData pointerEventData)

/// {

/// //Use this to tell when the user right-clicks on the Button

/// if (pointerEventData.button == PointerEventData.InputButton.Right)

/// {

/// //Output to console the clicked GameObject's name and the following message. You can replace this with your own actions for when clicking the GameObject.

/// Debug.Log(name + " Game Object Right Clicked!");

/// }

///

/// //Use this to tell when the user left-clicks on the Button

/// if (pointerEventData.button == PointerEventData.InputButton.Left)

/// {

/// Debug.Log(name + " Game Object Left Clicked!");

/// }

/// }

/// }

/// ]]>

///</code>

/// </example>

public virtual void OnPointerClick(PointerEventData eventData)

{

if (eventData.button != PointerEventData.InputButton.Left)

return;

Press();

}

/// <summary>

/// Call all registered ISubmitHandler.

/// </summary>

/// <param name="eventData">Associated data with the event. Typically by the event system.</param>

/// <remarks>

/// This detects when a Button has been selected via a "submit" key you specify (default is the return key).

///

/// To change the submit key, either:

///

/// 1. Go to Edit->Project Settings->Input.

///

/// 2. Next, expand the Axes section and go to the Submit section if it exists.

///

/// 3. If Submit doesn’t exist, add 1 number to the Size field. This creates a new section at the bottom. Expand the new section and change the Name field to “Submit”.

///

/// 4. Change the Positive Button field to the key you want (e.g. space).

///

///

/// Or:

///

/// 1. Go to your EventSystem in your Project

///

/// 2. Go to the Inspector window and change the Submit Button field to one of the sections in the Input Manager (e.g. "Submit"), or create your own by naming it what you like, then following the next few steps.

///

/// 3. Go to Edit->Project Settings->Input to get to the Input Manager.

///

/// 4. Expand the Axes section in the Inspector window. Add 1 to the number in the Size field. This creates a new section at the bottom.

///

/// 5. Expand the new section and name it the same as the name you inserted in the Submit Button field in the EventSystem. Set the Positive Button field to the key you want (e.g. space)

/// </remarks>

public virtual void OnSubmit(BaseEventData eventData)

{

Press();

// if we get set disabled during the press

// don't run the coroutine.

if (!IsActive() || !IsInteractable())

return;

DoStateTransition(SelectionState.Pressed, false);

StartCoroutine(OnFinishSubmit());

}

private IEnumerator OnFinishSubmit()

{

var fadeTime = colors.fadeDuration;

var elapsedTime = 0f;

while (elapsedTime < fadeTime)

{

elapsedTime += Time.unscaledDeltaTime;

yield return null;

}

DoStateTransition(currentSelectionState, false);

}

}

}

using System;

using System.ComponentModel;

using System.Diagnostics;

using System.Runtime.CompilerServices;

using UnityEngine.Bindings;

using UnityEngine.Internal;

using UnityEngine.Scripting;

namespace UnityEngine;

//

// 摘要:

// Class containing methods to ease debugging while developing a game.

[NativeHeader("Runtime/Export/Debug/Debug.bindings.h")]

public class Debug

{

internal static readonly ILogger s\_DefaultLogger = new Logger(new DebugLogHandler());

internal static ILogger s\_Logger = new Logger(new DebugLogHandler());

//

// 摘要:

// Get default debug logger.

public static ILogger unityLogger => s\_Logger;

//

// 摘要:

// Reports whether the development console is visible. The development console cannot

// be made to appear using:

public static extern bool developerConsoleVisible

{

[MethodImpl(MethodImplOptions.InternalCall)]

get;

[MethodImpl(MethodImplOptions.InternalCall)]

set;

}

//

// 摘要:

// In the Build Settings dialog there is a check box called "Development Build".

[StaticAccessor("GetBuildSettings()", StaticAccessorType.Dot)]

[NativeProperty(TargetType = TargetType.Field)]

public static extern bool isDebugBuild

{

[MethodImpl(MethodImplOptions.InternalCall)]

get;

}

[NativeThrows]

internal static extern DiagnosticSwitch[] diagnosticSwitches

{

[MethodImpl(MethodImplOptions.InternalCall)]

get;

}

[EditorBrowsable(EditorBrowsableState.Never)]

[Obsolete("Debug.logger is obsolete. Please use Debug.unityLogger instead (UnityUpgradable) -> unityLogger")]

public static ILogger logger => s\_Logger;

//

// 摘要:

// Draws a line between specified start and end points.

//

// 参数:

// start:

// Point in world space where the line should start.

//

// end:

// Point in world space where the line should end.

//

// color:

// Color of the line.

//

// duration:

// How long the line should be visible for.

//

// depthTest:

// Determines whether objects closer to the camera obscure the line.

[ExcludeFromDocs]

public static void DrawLine(Vector3 start, Vector3 end, Color color, float duration)

{

bool depthTest = true;

DrawLine(start, end, color, duration, depthTest);

}

//

// 摘要:

// Draws a line between specified start and end points.

//

// 参数:

// start:

// Point in world space where the line should start.

//

// end:

// Point in world space where the line should end.

//

// color:

// Color of the line.

//

// duration:

// How long the line should be visible for.

//

// depthTest:

// Determines whether objects closer to the camera obscure the line.

[ExcludeFromDocs]

public static void DrawLine(Vector3 start, Vector3 end, Color color)

{

bool depthTest = true;

float duration = 0f;

DrawLine(start, end, color, duration, depthTest);

}

//

// 摘要:

// Draws a line between specified start and end points.

//

// 参数:

// start:

// Point in world space where the line should start.

//

// end:

// Point in world space where the line should end.

//

// color:

// Color of the line.

//

// duration:

// How long the line should be visible for.

//

// depthTest:

// Determines whether objects closer to the camera obscure the line.

[ExcludeFromDocs]

public static void DrawLine(Vector3 start, Vector3 end)

{

bool depthTest = true;

float duration = 0f;

Color white = Color.white;

DrawLine(start, end, white, duration, depthTest);

}

//

// 摘要:

// Draws a line between specified start and end points.

//

// 参数:

// start:

// Point in world space where the line should start.

//

// end:

// Point in world space where the line should end.

//

// color:

// Color of the line.

//

// duration:

// How long the line should be visible for.

//

// depthTest:

// Determines whether objects closer to the camera obscure the line.

[FreeFunction("DebugDrawLine", IsThreadSafe = true)]

public static void DrawLine(Vector3 start, Vector3 end, [UnityEngine.Internal.DefaultValue("Color.white")] Color color, [UnityEngine.Internal.DefaultValue("0.0f")] float duration, [UnityEngine.Internal.DefaultValue("true")] bool depthTest)

{

DrawLine\_Injected(ref start, ref end, ref color, duration, depthTest);

}

//

// 摘要:

// Draws a line from start to start + dir in world coordinates.

//

// 参数:

// start:

// Point in world space where the ray should start.

//

// dir:

// Direction and length of the ray.

//

// color:

// Color of the drawn line.

//

// duration:

// How long the line will be visible for (in seconds).

//

// depthTest:

// Determines whether objects closer to the camera obscure the line.

[ExcludeFromDocs]

public static void DrawRay(Vector3 start, Vector3 dir, Color color, float duration)

{

bool depthTest = true;

DrawRay(start, dir, color, duration, depthTest);

}

//

// 摘要:

// Draws a line from start to start + dir in world coordinates.

//

// 参数:

// start:

// Point in world space where the ray should start.

//

// dir:

// Direction and length of the ray.

//

// color:

// Color of the drawn line.

//

// duration:

// How long the line will be visible for (in seconds).

//

// depthTest:

// Determines whether objects closer to the camera obscure the line.

[ExcludeFromDocs]

public static void DrawRay(Vector3 start, Vector3 dir, Color color)

{

bool depthTest = true;

float duration = 0f;

DrawRay(start, dir, color, duration, depthTest);

}

//

// 摘要:

// Draws a line from start to start + dir in world coordinates.

//

// 参数:

// start:

// Point in world space where the ray should start.

//

// dir:

// Direction and length of the ray.

//

// color:

// Color of the drawn line.

//

// duration:

// How long the line will be visible for (in seconds).

//

// depthTest:

// Determines whether objects closer to the camera obscure the line.

[ExcludeFromDocs]

public static void DrawRay(Vector3 start, Vector3 dir)

{

bool depthTest = true;

float duration = 0f;

Color white = Color.white;

DrawRay(start, dir, white, duration, depthTest);

}

//

// 摘要:

// Draws a line from start to start + dir in world coordinates.

//

// 参数:

// start:

// Point in world space where the ray should start.

//

// dir:

// Direction and length of the ray.

//

// color:

// Color of the drawn line.

//

// duration:

// How long the line will be visible for (in seconds).

//

// depthTest:

// Determines whether objects closer to the camera obscure the line.

public static void DrawRay(Vector3 start, Vector3 dir, [UnityEngine.Internal.DefaultValue("Color.white")] Color color, [UnityEngine.Internal.DefaultValue("0.0f")] float duration, [UnityEngine.Internal.DefaultValue("true")] bool depthTest)

{

DrawLine(start, start + dir, color, duration, depthTest);

}

//

// 摘要:

// Pauses the editor.

[MethodImpl(MethodImplOptions.InternalCall)]

[FreeFunction("PauseEditor")]

public static extern void Break();

[MethodImpl(MethodImplOptions.InternalCall)]

public static extern void DebugBreak();

//

// 摘要:

// Populate an unmanaged buffer with the current managed call stack as a sequence

// of UTF-8 bytes, without allocating GC memory. Returns the number of bytes written

// into the buffer.

//

// 参数:

// buffer:

// Target buffer to receive the callstack text

//

// bufferMax:

// Max number of bytes to write

//

// projectFolder:

// Project folder path, to clean up path names

[MethodImpl(MethodImplOptions.InternalCall)]

[ThreadSafe]

public unsafe static extern int ExtractStackTraceNoAlloc(byte\* buffer, int bufferMax, string projectFolder);

//

// 摘要:

// Logs a message to the Unity Console.

//

// 参数:

// message:

// String or object to be converted to string representation for display.

//

// context:

// Object to which the message applies.

public static void Log(object message)

{

unityLogger.Log(LogType.Log, message);

}

//

// 摘要:

// Logs a message to the Unity Console.

//

// 参数:

// message:

// String or object to be converted to string representation for display.

//

// context:

// Object to which the message applies.

public static void Log(object message, Object context)

{

unityLogger.Log(LogType.Log, message, context);

}

//

// 摘要:

// Logs a formatted message to the Unity Console.

//

// 参数:

// format:

// A composite format string.

//

// args:

// Format arguments.

//

// context:

// Object to which the message applies.

//

// logType:

// Type of message e.g. warn or error etc.

//

// logOptions:

// Option flags to treat the log message special.

public static void LogFormat(string format, params object[] args)

{

unityLogger.LogFormat(LogType.Log, format, args);

}

//

// 摘要:

// Logs a formatted message to the Unity Console.

//

// 参数:

// format:

// A composite format string.

//

// args:

// Format arguments.

//

// context:

// Object to which the message applies.

//

// logType:

// Type of message e.g. warn or error etc.

//

// logOptions:

// Option flags to treat the log message special.

public static void LogFormat(Object context, string format, params object[] args)

{

unityLogger.LogFormat(LogType.Log, context, format, args);

}

//

// 摘要:

// Logs a formatted message to the Unity Console.

//

// 参数:

// format:

// A composite format string.

//

// args:

// Format arguments.

//

// context:

// Object to which the message applies.

//

// logType:

// Type of message e.g. warn or error etc.

//

// logOptions:

// Option flags to treat the log message special.

public static void LogFormat(LogType logType, LogOption logOptions, Object context, string format, params object[] args)

{

if (!(unityLogger.logHandler is DebugLogHandler debugLogHandler))

{

unityLogger.LogFormat(logType, context, format, args);

}

else if (unityLogger.IsLogTypeAllowed(logType))

{

debugLogHandler.LogFormat(logType, logOptions, context, format, args);

}

}

//

// 摘要:

// A variant of Debug.Log that logs an error message to the console.

//

// 参数:

// message:

// String or object to be converted to string representation for display.

//

// context:

// Object to which the message applies.

public static void LogError(object message)

{

unityLogger.Log(LogType.Error, message);

}

//

// 摘要:

// A variant of Debug.Log that logs an error message to the console.

//

// 参数:

// message:

// String or object to be converted to string representation for display.

//

// context:

// Object to which the message applies.

public static void LogError(object message, Object context)

{

unityLogger.Log(LogType.Error, message, context);

}

//

// 摘要:

// Logs a formatted error message to the Unity console.

//

// 参数:

// format:

// A composite format string.

//

// args:

// Format arguments.

//

// context:

// Object to which the message applies.

public static void LogErrorFormat(string format, params object[] args)

{

unityLogger.LogFormat(LogType.Error, format, args);

}

//

// 摘要:

// Logs a formatted error message to the Unity console.

//

// 参数:

// format:

// A composite format string.

//

// args:

// Format arguments.

//

// context:

// Object to which the message applies.

public static void LogErrorFormat(Object context, string format, params object[] args)

{

unityLogger.LogFormat(LogType.Error, context, format, args);

}

internal static void LogError(string message, string fileName, int lineNumber, int columnNumber)

{

LogCompilerError(message, fileName, lineNumber, columnNumber);

}

internal static void LogWarning(string message, string fileName, int lineNumber, int columnNumber)

{

LogCompilerWarning(message, fileName, lineNumber, columnNumber);

}

internal static void LogInfo(string message, string fileName, int lineNumber, int columnNumber)

{

LogInformation(message, fileName, lineNumber, columnNumber);

}

[MethodImpl(MethodImplOptions.InternalCall)]

internal static extern void LogCompilerMessage(string message, string fileName, int lineNumber, int columnNumber, bool forEditor, bool isError, int identifier, int instanceId);

[MethodImpl(MethodImplOptions.InternalCall)]

private static extern void LogCompilerWarning(string message, string fileName, int lineNumber, int columnNumber);

[MethodImpl(MethodImplOptions.InternalCall)]

private static extern void LogCompilerError(string message, string fileName, int lineNumber, int columnNumber);

[MethodImpl(MethodImplOptions.InternalCall)]

private static extern void LogInformation(string message, string fileName, int lineNumber, int columnNumber);

//

// 摘要:

// Clears errors from the developer console.

[MethodImpl(MethodImplOptions.InternalCall)]

public static extern void ClearDeveloperConsole();

//

// 摘要:

// A variant of Debug.Log that logs an error message to the console.

//

// 参数:

// context:

// Object to which the message applies.

//

// exception:

// Runtime Exception.

public static void LogException(Exception exception)

{

unityLogger.LogException(exception, null);

}

//

// 摘要:

// A variant of Debug.Log that logs an error message to the console.

//

// 参数:

// context:

// Object to which the message applies.

//

// exception:

// Runtime Exception.

public static void LogException(Exception exception, Object context)

{

unityLogger.LogException(exception, context);

}

[MethodImpl(MethodImplOptions.InternalCall)]

internal static extern void LogPlayerBuildError(string message, string file, int line, int column);

//

// 摘要:

// A variant of Debug.Log that logs a warning message to the console.

//

// 参数:

// message:

// String or object to be converted to string representation for display.

//

// context:

// Object to which the message applies.

public static void LogWarning(object message)

{

unityLogger.Log(LogType.Warning, message);

}

//

// 摘要:

// A variant of Debug.Log that logs a warning message to the console.

//

// 参数:

// message:

// String or object to be converted to string representation for display.

//

// context:

// Object to which the message applies.

public static void LogWarning(object message, Object context)

{

unityLogger.Log(LogType.Warning, message, context);

}

//

// 摘要:

// Logs a formatted warning message to the Unity Console.

//

// 参数:

// format:

// A composite format string.

//

// args:

// Format arguments.

//

// context:

// Object to which the message applies.

public static void LogWarningFormat(string format, params object[] args)

{

unityLogger.LogFormat(LogType.Warning, format, args);

}

//

// 摘要:

// Logs a formatted warning message to the Unity Console.

//

// 参数:

// format:

// A composite format string.

//

// args:

// Format arguments.

//

// context:

// Object to which the message applies.

public static void LogWarningFormat(Object context, string format, params object[] args)

{

unityLogger.LogFormat(LogType.Warning, context, format, args);

}

//

// 摘要:

// Assert a condition and logs an error message to the Unity console on failure.

//

//

// 参数:

// condition:

// Condition you expect to be true.

//

// context:

// Object to which the message applies.

//

// message:

// String or object to be converted to string representation for display.

[Conditional("UNITY\_ASSERTIONS")]

public static void Assert(bool condition)

{

if (!condition)

{

unityLogger.Log(LogType.Assert, "Assertion failed");

}

}

//

// 摘要:

// Assert a condition and logs an error message to the Unity console on failure.

//

//

// 参数:

// condition:

// Condition you expect to be true.

//

// context:

// Object to which the message applies.

//

// message:

// String or object to be converted to string representation for display.

[Conditional("UNITY\_ASSERTIONS")]

public static void Assert(bool condition, Object context)

{

if (!condition)

{

unityLogger.Log(LogType.Assert, (object)"Assertion failed", context);

}

}

//

// 摘要:

// Assert a condition and logs an error message to the Unity console on failure.

//

//

// 参数:

// condition:

// Condition you expect to be true.

//

// context:

// Object to which the message applies.

//

// message:

// String or object to be converted to string representation for display.

[Conditional("UNITY\_ASSERTIONS")]

public static void Assert(bool condition, object message)

{

if (!condition)

{

unityLogger.Log(LogType.Assert, message);

}

}

[Conditional("UNITY\_ASSERTIONS")]

public static void Assert(bool condition, string message)

{

if (!condition)

{

unityLogger.Log(LogType.Assert, message);

}

}

//

// 摘要:

// Assert a condition and logs an error message to the Unity console on failure.

//

//

// 参数:

// condition:

// Condition you expect to be true.

//

// context:

// Object to which the message applies.

//

// message:

// String or object to be converted to string representation for display.

[Conditional("UNITY\_ASSERTIONS")]

public static void Assert(bool condition, object message, Object context)

{

if (!condition)

{

unityLogger.Log(LogType.Assert, message, context);

}

}

[Conditional("UNITY\_ASSERTIONS")]

public static void Assert(bool condition, string message, Object context)

{

if (!condition)

{

unityLogger.Log(LogType.Assert, (object)message, context);

}

}

//

// 摘要:

// Assert a condition and logs a formatted error message to the Unity console on

// failure.

//

// 参数:

// condition:

// Condition you expect to be true.

//

// format:

// A composite format string.

//

// args:

// Format arguments.

//

// context:

// Object to which the message applies.

[Conditional("UNITY\_ASSERTIONS")]

public static void AssertFormat(bool condition, string format, params object[] args)

{

if (!condition)

{

unityLogger.LogFormat(LogType.Assert, format, args);

}

}

//

// 摘要:

// Assert a condition and logs a formatted error message to the Unity console on

// failure.

//

// 参数:

// condition:

// Condition you expect to be true.

//

// format:

// A composite format string.

//

// args:

// Format arguments.

//

// context:

// Object to which the message applies.

[Conditional("UNITY\_ASSERTIONS")]

public static void AssertFormat(bool condition, Object context, string format, params object[] args)

{

if (!condition)

{

unityLogger.LogFormat(LogType.Assert, context, format, args);

}

}

//

// 摘要:

// A variant of Debug.Log that logs an assertion message to the console.

//

// 参数:

// message:

// String or object to be converted to string representation for display.

//

// context:

// Object to which the message applies.

[Conditional("UNITY\_ASSERTIONS")]

public static void LogAssertion(object message)

{

unityLogger.Log(LogType.Assert, message);

}

//

// 摘要:

// A variant of Debug.Log that logs an assertion message to the console.

//

// 参数:

// message:

// String or object to be converted to string representation for display.

//

// context:

// Object to which the message applies.

[Conditional("UNITY\_ASSERTIONS")]

public static void LogAssertion(object message, Object context)

{

unityLogger.Log(LogType.Assert, message, context);

}

//

// 摘要:

// Logs a formatted assertion message to the Unity console.

//

// 参数:

// format:

// A composite format string.

//

// args:

// Format arguments.

//

// context:

// Object to which the message applies.

[Conditional("UNITY\_ASSERTIONS")]

public static void LogAssertionFormat(string format, params object[] args)

{

unityLogger.LogFormat(LogType.Assert, format, args);

}

//

// 摘要:

// Logs a formatted assertion message to the Unity console.

//

// 参数:

// format:

// A composite format string.

//

// args:

// Format arguments.

//

// context:

// Object to which the message applies.

[Conditional("UNITY\_ASSERTIONS")]

public static void LogAssertionFormat(Object context, string format, params object[] args)

{

unityLogger.LogFormat(LogType.Assert, context, format, args);

}

[MethodImpl(MethodImplOptions.InternalCall)]

[FreeFunction("DeveloperConsole\_OpenConsoleFile")]

internal static extern void OpenConsoleFile();

internal static DiagnosticSwitch GetDiagnosticSwitch(string name)

{

DiagnosticSwitch[] array = diagnosticSwitches;

foreach (DiagnosticSwitch diagnosticSwitch in array)

{

if (diagnosticSwitch.name == name)

{

return diagnosticSwitch;

}

}

throw new ArgumentException("Could not find DiagnosticSwitch named " + name);

}

[RequiredByNativeCode]

internal static bool CallOverridenDebugHandler(Exception exception, Object obj)

{

if (unityLogger.logHandler is DebugLogHandler)

{

return false;

}

try

{

unityLogger.LogException(exception, obj);

}

catch (Exception arg)

{

s\_DefaultLogger.LogError($"Invalid exception thrown from custom {unityLogger.logHandler.GetType()}.LogException(). Message: {arg}", obj);

return false;

}

return true;

}

[RequiredByNativeCode]

internal static bool IsLoggingEnabled()

{

if (unityLogger.logHandler is DebugLogHandler)

{

return unityLogger.logEnabled;

}

return s\_DefaultLogger.logEnabled;

}

[MethodImpl(MethodImplOptions.InternalCall)]

internal static extern void LogSticky(int identifier, LogType logType, LogOption logOptions, string message, Object context = null);

[MethodImpl(MethodImplOptions.InternalCall)]

internal static extern void RemoveLogEntriesByIdentifier(int identifier);

[Obsolete("Assert(bool, string, params object[]) is obsolete. Use AssertFormat(bool, string, params object[]) (UnityUpgradable) -> AssertFormat(\*)", true)]

[EditorBrowsable(EditorBrowsableState.Never)]

[Conditional("UNITY\_ASSERTIONS")]

public static void Assert(bool condition, string format, params object[] args)

{

if (!condition)

{

unityLogger.LogFormat(LogType.Assert, format, args);

}

}

[MethodImpl(MethodImplOptions.InternalCall)]

private static extern void DrawLine\_Injected(ref Vector3 start, ref Vector3 end, [UnityEngine.Internal.DefaultValue("Color.white")] ref Color color, [UnityEngine.Internal.DefaultValue("0.0f")] float duration, [UnityEngine.Internal.DefaultValue("true")] bool depthTest);

}