Technical Solution - Contents

[BinarySearch.cs 2](#_Toc130588543)

[characterchecking.cs 3](#_Toc130588544)

[CircularQueue.cs 3](#_Toc130588545)

[gravity.cs 5](#_Toc130588546)

[hashing.cs 6](#_Toc130588547)

[LinkedList.cs 7](#_Toc130588548)

[MergeSort.cs 9](#_Toc130588549)

[projectile.cs 10](#_Toc130588550)

[Stack.cs 11](#_Toc130588551)

[DifficultyPanel.cs 12](#_Toc130588552)

[FirstTimeMan.cs 13](#_Toc130588553)

[circular.cs 13](#_Toc130588554)

[Clock.cs 15](#_Toc130588555)

[escapedcontroller.cs 16](#_Toc130588556)

[exitTrigger.cs 17](#_Toc130588557)

[gameOver.cs 18](#_Toc130588558)

[mainChar.cs 19](#_Toc130588559)

[MazeManager.cs 19](#_Toc130588560)

[minimap.cs 29](#_Toc130588561)

[onLoadMaze.cs 30](#_Toc130588562)

[open\_gate.cs 30](#_Toc130588563)

[passwordPanel.cs 31](#_Toc130588564)

[pause.cs 32](#_Toc130588565)

[PerlinNoiseMap.cs 33](#_Toc130588566)

[PickUp.cs 34](#_Toc130588567)

[PickUpSelectionSlot.cs 35](#_Toc130588568)

[CharacterDataBase.cs 35](#_Toc130588569)

[characterSpecification.cs 36](#_Toc130588570)

[charcard.cs 36](#_Toc130588571)

[Inventory.cs 36](#_Toc130588572)

[item.cs 38](#_Toc130588573)

[itemsObject.cs 39](#_Toc130588574)

[Slot.cs 39](#_Toc130588575)

[inte.cs 40](#_Toc130588576)

[goto.cs 44](#_Toc130588577)

[Maze.cs 45](#_Toc130588578)

[Mazecell.cs 47](#_Toc130588579)

[MazeGeneration.cs 47](#_Toc130588580)

[SingleMazeGeneration.cs 50](#_Toc130588581)

[WallCell.cs 52](#_Toc130588582)

[BigGreiver.cs 52](#_Toc130588583)

[blade.cs 52](#_Toc130588584)

[MonsterBaseClass.cs 53](#_Toc130588585)

[SilverFish.cs 56](#_Toc130588586)

[SmallGreiver.cs 56](#_Toc130588587)

[movement.cs 56](#_Toc130588588)

[newTerrainGravity.cs 58](#_Toc130588589)

[PlayerSetup.cs 60](#_Toc130588590)

[switchcam.cs 61](#_Toc130588591)

[toucharea.cs 62](#_Toc130588592)

[turnaround.cs 63](#_Toc130588593)

[continuePanel.cs 64](#_Toc130588594)

[fadeOutImage.cs 64](#_Toc130588595)

[LeaderBoarddata.cs 65](#_Toc130588596)

[leaderslot.cs 67](#_Toc130588597)

[loginState.cs 68](#_Toc130588598)

[MainMenuManager.cs 71](#_Toc130588599)

[onLoadMainMenu.cs 74](#_Toc130588600)

[menuBaseClass.cs 75](#_Toc130588601)

[QuitMenuPanel.cs 75](#_Toc130588602)

[SlotSelectionPanel.cs 76](#_Toc130588603)

[AudioSettingPanel.cs 78](#_Toc130588604)

[ControlSettingPanel.cs 78](#_Toc130588605)

[GraphicsSettingPanel.cs 78](#_Toc130588606)

[Setting.cs 78](#_Toc130588607)

[SettingManager.cs 81](#_Toc130588608)

[VideoSettingPanel.cs 83](#_Toc130588609)

[LoadingCircle.cs 83](#_Toc130588610)

[SplashScreen.cs 83](#_Toc130588611)

[Fading.cs 84](#_Toc130588612)

[health.cs 85](#_Toc130588613)

[LoadingPanel.cs 87](#_Toc130588614)

[objectcloner.cs 89](#_Toc130588615)

[PopupPanel.cs 90](#_Toc130588616)

[Singleton.cs 90](#_Toc130588617)

[CsharpREST.cs 94](#_Toc130588618)

[db.cs 95](#_Toc130588619)

[UnityREST.cs 97](#_Toc130588620)

[Date.php 98](#_Toc130588621)

[Index.php 98](#_Toc130588622)

[Index.js 99](#_Toc130588623)

[Cross.html 101](#_Toc130588624)

[Done.html 105](#_Toc130588625)

[Pivot.html 105](#_Toc130588626)

[Register.html 109](#_Toc130588627)

[Register.js 113](#_Toc130588628)

Algorithm

# BinarySearch.cs

using System.Collections;  
using System.Collections.Generic;  
using UnityEngine;  
using System;  
  
public class BinarySearch : MonoBehaviour  
{

int Search(int[] list, int find, int min, int max) //need orderedlist // need hashed value  
 {  
 if (max >= min) //check if the list is not empty   
 {

//find out the index of mid value  
 int mid = (new Func<int>(() => Mathf.FloorToInt((min + max) / 2)))();

//return found data  
 if (find == list[mid]) return mid;

//first half of the list  
 else if (find < list[mid]) return Search(list, find, min, mid - 1);

//second half of the list  
 else return Search(list, find, mid + 1, max); //discard half of the list  
 }  
 else return -1;// exit if not found in whole list  
 }  
}

# characterchecking.cs

using System.Collections;  
using System.Collections.Generic;  
using UnityEngine;  
  
public static class Checking{  
 public static bool check(string password)   
 {

//remove unwant character before the string  
 password = password.Trim();

//loop through the string  
 foreach (char a in password){   
 switch ((int)a)  
 {  
 case int i when i >= 48 && i <= 57: //numbers  
 break;  
 case int i when i >= 65 && i <= 90: //UpperCase   
 break;  
 case int i when i >= 97 && i <= 122: //lowerCase  
 break;  
 default: //other invalid characters  
 return true;  
 }  
 }  
 return false; //no invalid character is found  
 }   
}

# CircularQueue.cs

using System.Collections;  
using System.Collections.Generic;  
using System.Threading.Tasks;  
using UnityEngine;  
  
public class CQueue  
{  
 int[] \_queue;  
 bool \_circular = false;  
 int \_frontofqueue = -1, \_rearofqueue = -1, \_numberofitem;  
  
 public CQueue(bool circular, int numberofitem)

//constructor (define circular queue or linear queue) and the size of the queue  
 {  
 this.\_circular = circular;  
 this.\_numberofitem = numberofitem;  
 \_queue = new int[numberofitem];  
 }  
  
 public int[] Queue1 => \_queue;  
  
 public void Enqueue(int data)   
 {   
 if (\_circular == false || (((\_rearofqueue + 1) % \_numberofitem + 1) == \_frontofqueue))  
 // check if the list is full and circular mode == false  
 { }  
 else {  
 \_rearofqueue = (\_rearofqueue + 1) % \_numberofitem;  
 \_queue[\_rearofqueue] = data; //input data into the Queue  
 }  
 }  
 public int Dequeue()  
 {  
 if (\_circular == false || (((\_frontofqueue + 1) % \_numberofitem + 1) ==\_rearofqueue))

// check if empty and circular == true

return -1; //data not available  
 else {

//pointer has move to next position and return dequeued value  
 \_frontofqueue = (\_frontofqueue + 1) % \_numberofitem;  
 return \_queue[\_frontofqueue];  
 }  
 }  
 public int Peek()  
 {  
 if (\_rearofqueue == -1)

// Check if every item is dequeued and empty queue left behind

return -1;  
 else {  
 int frontofqueue = (\_frontofqueue + 1) % \_numberofitem;  
 return \_queue[frontofqueue];  
 }  
 }  
}

# gravity.cs

using System;  
using UnityEngine;  
  
 public class gravity : MonoBehaviour  
 {  
   
 //gravity  
 [SerializeField] protected bool \_grounded = false;  
 [SerializeField] protected float gravityacceleration;  
 [SerializeField] private float \_jumpgravity;  
 [SerializeField] private float \_gravityspeed;  
  
 protected Vector3 original;  
 [SerializeField] protected bool \_isUpJumping;  
 public bool isGrounded {  
 get => \_grounded;  
 set => \_grounded = value;  
 }  
 public float gravityspeed {  
 get => \_gravityspeed;  
 set => \_gravityspeed = value;  
 }  
  
 public void Update()  
 {  
 original = this.transform.position;  
 }  
  
 public float jumpgravity {  
 get => \_jumpgravity;  
 set => \_jumpgravity = value;  
 }  
 public float gravitySetting(){  
 if(original.y < 0.03f && gravityacceleration<=0) {

//check if player at ground level  
 \_grounded = true;  
 gravityacceleration = 0;  
 }else{

//set the object is not on the ground  
 \_grounded = false;  
 if(gravityacceleration < 0){ //check if player jumping upward or downward

//use general gravity  
 \_isUpJumping = false;   
 gravityacceleration -= \_gravityspeed\*Time.deltaTime;  
 }else{

//use jump gravity  
 gravityacceleration -= \_jumpgravity\*Time.deltaTime;  
 }   
 }return gravityacceleration\*Time.deltaTime;  
 }  
 }

# hashing.cs

using System;  
using System.Collections.Generic;  
using UnityEngine;  
  
namespace algorithm  
{  
 public class hashing : MonoBehaviour  
 {  
 public int Convert(string key) {

//convert key value into integer hash value  
 int hashvalue = 0;  
 int count = 7;  
 foreach (var ch in key)

//loop through each character  
 {  
 hashvalue += (int)((int)ch \* Mathf.Pow(10, count));  
 count -= 1;  
 }  
 return hashvalue/key.Length;

//reduce the size of the hash and check the length of the key  
 }  
 public void HashTable(string[] list)   
 {  
 int count = 0;  
 Dictionary<int, object> table = new Dictionary<int, object>();  
 for (int i = 0; i < list.Length; i++)  
 {

//loop through the list to put in dictionary based hash table  
 int hashvalue = Convert(list[count]);  
 if (!table.TryAdd(hashvalue, list[count]))  
 {  
 //linear probing  
 while (!table.TryAdd(hashvalue,list[count])) hashvalue++;

//chaining  
 string temp = (string)table[hashvalue]; lList list1;  
 if (table[hashvalue].GetType() != typeof(string))  
 {  
 list1 = new lList(list.Length);  
 list1.AddHead(temp);  
 }else list1 = (lList)table[hashvalue];  
 list1.AddFoot(list[count]);  
 } count++;  
 }  
 }  
 }  
}

# LinkedList.cs

using System.Collections;  
using System.Collections.Generic;  
using UnityEngine;  
using Microsoft.CSharp;  
  
public class lList{

object[,] dataNmemoryloc;  
 int Headloc = -1;  
   
 public lList(int num) {

//contrustor, set head pointer and   
 dataNmemoryloc = new object[num,4];  
 for (int i = 0; i < num; i++) { dataNmemoryloc[i, 0] = true; }  
 }  
 public void AddHead(object data)  
 {  
 int temp = Headloc;  
 Headloc = AddValue(data);  
 if (Headloc == -1) { dataNmemoryloc[Headloc, 3] = null; }  
 else{ dataNmemoryloc[Headloc, 3] = temp; }  
 }  
 int? GetNextPointer(int num) { return (int?)dataNmemoryloc[num, 3]; }  
 dynamic GetBlockData(int num) { return (dynamic)dataNmemoryloc[num, 1]; }  
 public void AddFoot(object data) {  
 int newloc = AddValue(data);  
 int pointer = Headloc;  
 if (dataNmemoryloc[pointer, 3] != null) pointer = (int)GetNextPointer(pointer);   
 dataNmemoryloc[pointer,3] = newloc;// was last one  
 dataNmemoryloc[newloc, 3] = null; //is last one  
 }  
 int? Search(object datatofind, int loc,bool firstTime) //use base case control recursion but one return needed?  
 { if (firstTime) { loc = (int)Headloc; firstTime = false; }  
 object data = GetBlockData(loc);  
 if (data == null) { return -1;  
 }else if (data != datatofind) {  
 loc = (int)GetNextPointer(loc);  
 Search(datatofind, loc, firstTime);  
 return null;  
 }  
 else return loc;  
 }  
  
 List<dynamic> Traversal(List<dynamic> list, int loc, bool firstTime = true)  
 {  
 if (firstTime) { list = new List<dynamic>(); loc = Headloc; firstTime = false; }  
 object data = GetBlockData(loc);  
 if (data != null)  
 {  
 list.Add(data);  
 Traversal(list, (int)GetNextPointer(loc), false);  
 return null;  
 }else { return list; }  
 }  
 int? SearchPriority(int priority, int loc = -1, bool firstTime= true) {  
 if(firstTime) { loc = (int)Headloc; firstTime = false; }  
 if(priority <= (int)dataNmemoryloc[loc, 2])  
 {//repeat  
 loc = (int)dataNmemoryloc[loc, 2];  
 SearchPriority(priority, loc, firstTime);  
 return null;  
 }else { return loc; }  
 }  
 void AddWithPriority(object inputdata, int priority)  
 {  
 int temp = (int)dataNmemoryloc[(int)SearchPriority(priority), 2];   
 int newloc = AddValue(inputdata, priority);  
 dataNmemoryloc[temp, 2] = newloc;  
 dataNmemoryloc[newloc, 2] = temp;  
 }  
 int AddValue(object data, int priority = 5)  
 {  
 List<int> heap = new List<int>();  
 for (int i = 0; i < dataNmemoryloc.GetLength(0); i++) {  
 if ((bool)dataNmemoryloc[i, 0]) { heap.Add(i); } }  
 int newloc = Random.Range(0,heap.Count);  
 dataNmemoryloc[newloc, 0] = false;  
 dataNmemoryloc[newloc, 1] = (dynamic)data;  
 dataNmemoryloc[newloc, 2] = priority;  
 return newloc;  
 }  
}

# MergeSort.cs

using System.Collections;  
using System.Collections.Generic;  
using UnityEngine;  
  
public class MergeSortClass: MonoBehaviour{  
 public Leaderboard[] MergeSort(Leaderboard[] list, int sortData)  
 {int mid = list.Length / 2;  
  
 if (list.Length <= 1)  
 {  
 return list;  
 }  
 Leaderboard[] left = new Leaderboard[mid];  
 Leaderboard[] right = new Leaderboard[list.Length - mid];  
  
 for (int i = 0; i < left.Length; i++)  
 {  
 left[i] = list[i];  
 }  
  
 for (int i = 0; i < right.Length; i++)  
 {  
 right[i] = list[i + mid];  
 }  
  
 left = MergeSort(left, sortData);  
 right = MergeSort(right, sortData);  
  
 return Merge(left, right,sortData);  
 }  
 Leaderboard[] Merge(Leaderboard[] left, Leaderboard[] right, int dataToSort)  
 {  
 Leaderboard[] result = new Leaderboard[left.Length + right.Length];  
 int leftIndex = 0;  
 int rightIndex = 0;  
 int resultIndex = 0;  
  
 while (leftIndex < left.Length && rightIndex < right.Length)  
 {  
 int comparsion;  
 switch (dataToSort)  
 {  
 case 1:  
 comparsion = left[leftIndex].Level.CompareTo(right[rightIndex].Level);  
 break;  
 case 2:  
 comparsion = left[leftIndex].Score.CompareTo(right[rightIndex].Score);  
 break;  
 case 3:  
 comparsion = left[leftIndex].Since.CompareTo(right[rightIndex].Since);  
 break;  
 default:  
 comparsion = left[leftIndex].Username.CompareTo(right[rightIndex].Username);  
 break;  
 }  
 if (comparsion< 0)  
 {  
 result[resultIndex] = left[leftIndex];  
 leftIndex++;  
 }  
 else  
 {  
 result[resultIndex] = right[rightIndex];  
 rightIndex++;  
 }  
 resultIndex++;  
 }  
  
 while (leftIndex < left.Length)  
 {  
 result[resultIndex] = left[leftIndex];  
 leftIndex++;  
 resultIndex++;  
 }  
  
 while (rightIndex < right.Length)  
 {  
 result[resultIndex] = right[rightIndex];  
 rightIndex++;  
 resultIndex++;  
 }  
  
 return result;  
 }  
}

# projectile.cs

ï»¿using UnityEngine;  
  
namespace algorithm  
{  
 public class projectile : gravity  
 {  
 private float yacc;  
 public float speed = 20f;  
 public Vector3 gravity = Physics.gravity;  
  
 private Vector3 velocity = Vector3.zero;  
  
 void Update()  
 {  
 velocity += gravity \* Time.deltaTime;  
 transform.position += velocity \* Time.deltaTime + transform.forward \* (speed \* Time.deltaTime);  
 }  
 public void Shoot(GameObject prefab, Transform trans, float force)  
 {  
 GameObject arrow = Instantiate(prefab,trans.position+ Vector3.up\*1.5f+ Vector3.right\*0.3f,trans.rotation,GameObject.Find("objs").transform);  
 arrow.tag = "Axe";  
 projectile script = arrow.AddComponent<projectile>();  
 }  
 }  
}

# Stack.cs

using System.Collections;  
using System.Collections.Generic;  
using UnityEngine;  
  
public class Stack  
{  
 object[] \_stack\_arr;  
 private int \_pointer;  
 public int pointer => \_pointer;  
 public Stack(int \_numberofsize){  
 \_pointer = -1;  
 \_stack\_arr = new object[\_numberofsize];  
 }  
 public void Push(object data) {  
 if (pointer < \_stack\_arr.Length - 1) {  
 //if pointer 38 < (40-1 = 39)  
 \_pointer = pointer + 1;  
 \_stack\_arr[pointer] = data;  
 }  
 }  
  
 public bool Contains(object data)  
 {  
 foreach (var VARIABLE in \_stack\_arr)  
 {  
 if (VARIABLE == data) return true;  
 }return false;  
 }  
 public object Pop() {  
 if (pointer >= 0){  
 \_pointer = pointer - 1;  
 return \_stack\_arr[pointer+1];  
 }  
 else return null;  
 }  
 public object Peek(){  
 if (pointer == -1) return null;   
 else return \_stack\_arr[pointer];  
 }  
}

First Time

# DifficultyPanel.cs

using System;  
using System.Collections.Generic;  
using System.Threading.Tasks;  
using Newtonsoft.Json;  
using UnityEngine;  
using UnityEngine.SceneManagement;  
using UnityEngine.UI;  
  
public class DifficultyPanel : MenuPanels  
{  
 private Slider diffi;  
  
 public override void Start()  
 { diffi = FindObjectOfType<Slider>();  
 base.Start();  
 }  
 public void load()  
 {  
 Task.Run((() => Singleton.Localdb.Query("ALTER TABLE GAMEDATA ADD difficulty INTEGER",$"Save{Singleton.Instance.LoadNumber}.sqlite3"))).Wait();  
 Task.Run((() => Singleton.Localdb.Query($"INSERT INTO GAMEDATA(difficulty) VALUES ({diffi.value})",$"Save{Singleton.Instance.LoadNumber}.sqlite3"))).Wait();  
 SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);  
 }  
   
 // Update is called once per frame  
 void Update()  
 {  
 Singleton.Instance.Difficulty = (int)diffi.value;  
 }  
}

# FirstTimeMan.cs

using System;  
using System.Collections;  
using System.Collections.Generic;  
using UnityEngine.SceneManagement;  
using UnityEngine;  
  
public class FirstTimeMan : MonoBehaviour  
{  
 [SerializeField]private GameObject loader;  
 [SerializeField]private Animator \_animator;  
 // Start is called before the first fram  
 void Awake()  
 {  
 Singleton.Instance.init();  
 }  
  
 private void Update()  
 {  
 AnimatorStateInfo stateInfo = \_animator.GetCurrentAnimatorStateInfo(0);  
 if (stateInfo.normalizedTime >= 0.818f )//&& !\_animator.IsInTransition(0)  
 {  
 LoadScene();  
 }  
 }  
  
 public void LoadScene()  
 {  
 loader.SetActive(true);  
   
 }  
}

MazeGame

# circular.cs

ï»¿using UnityEngine;  
  
namespace MazeScreen  
{  
 public class circular : MonoBehaviour  
 {  
 private Vector2 center;  
 [SerializeField] private float radius;  
 [SerializeField] private Vector2 startingPoint;  
 [SerializeField] private float meterWidth;  
 [SerializeField] private float repeat,angle;  
   
 [SerializeField] private GameObject objectToRotate;  
 private Vector3 rotationPoint;  
 private float rotationSpeed;  
 private void Start()  
 {  
 MazeManager maze1 =FindObjectOfType<MazeManager>();  
 maze1.enabled = true;  
 objectToRotate = GameObject.Find("blockingWall");  
 ObjectCloner oc = new ObjectCloner();  
 float length = Vector2.Distance(startingPoint, center);  
 float circumference = 2 \* Mathf.PI \* radius;  
 float arcLength = meterWidth / length \* circumference;  
 float centralAngle = arcLength / circumference \* 360;  
 GameObject wall = Resources.Load<GameObject>("Maze/wall");  
 for (int i = 0; i < repeat; i++)  
 {  
 startingPoint =RotatePointAroundPivot(startingPoint, center, Quaternion.Euler(0, 0, centralAngle));  
   
 GameObject instantiateWall = Instantiate(wall, GameObject.Find("outerWall").transform);  
 instantiateWall.name = $"outerWall{i}";  
 instantiateWall.transform.localScale = new Vector3(20, 15, 1);  
 instantiateWall.transform.position = new Vector3(startingPoint.x,0,startingPoint.y);  
 instantiateWall.transform.localRotation = Quaternion.Euler(new Vector3(0,180-i\*angle,0));  
  
 }  
  
 for (int i = 0; i < 3; i++)  
 {  
 objectToRotate = oc.CloneObject(objectToRotate, Vector3.zero);  
 objectToRotate.transform.RotateAround(rotationPoint, Vector3.up, 90f);  
 objectToRotate.name = $"blockingWall{i}";  
 }  
 }  
  
 private Vector2 RotatePointAroundPivot(Vector2 point, Vector2 pivot, Quaternion rotation)  
 {  
 Vector3 rotatedPoint = rotation \* (new Vector3(point.x, point.y, 0) - new Vector3(pivot.x, pivot.y, 0)) + new Vector3(pivot.x, pivot.y, 0);  
 return new Vector2(rotatedPoint.x, rotatedPoint.y);  
 }  
 }  
   
}

# Clock.cs

using System.Collections;  
using System.Collections.Generic;  
using TMPro;  
using UnityEngine;  
using UnityEngine.Serialization;  
  
public class Clock : MonoBehaviour  
{  
 private Light light;  
 Vector3 rotate = Vector3.zero;  
 public float time;  
 [SerializeField] private TMP\_Text text;  
 public int date;  
  
 public float Time1 => time;  
  
 void Start()  
 {  
 date = 0;  
 light = FindObjectOfType<Light>();  
 }  
  
 // Update is called once per frame  
 void FixedUpdate()  
 {  
 Singleton.Instance.Time1 = time;  
 time += Time.deltaTime /2.5f; //add 1 every second  
 if (time % 24 < 0.05f) // Check if time is a multiple of 24  
 {  
 FindObjectOfType<MazeManager>().openGate();  
 date += 1;  
 date %= 2;  
 if (date == 1)  
 {  
 FindObjectOfType<MazeManager>().spawnMonster();  
 }  
 }  
 text.text = $"{(int)time%24}:{Mathf.FloorToInt(60f \* (time - (int)time))}"; //clamp between 0 to 24  
 rotate.x = (time % 24f \* 360f / 24f);   
 light.transform.rotation = Quaternion.Euler(rotate);  
 }

# escapedcontroller.cs

using System.IO;  
using System.Threading.Tasks;  
using TMPro;  
using UnityEngine;  
using UnityEngine.UI;  
  
public class escapedcontroller : MonoBehaviour  
{  
 [SerializeField] private TMP\_Text time, monster, difficulties;  
 [SerializeField] private GameObject gameOverPanel, PassedPanel;  
 [SerializeField] private GameObject mainPanel, loginPanel;  
 [SerializeField]private Button CloseButton;  
 private string username;  
  
 private void Start()  
 {  
 var SaveNumber = Singleton.Instance.LoadNumber;  
 File.ReadAllText(Application.persistentDataPath + "/continue.txt");  
 var result = Task.Run(() =>   
 Singleton.Localdb.Query("SELECT time, monsterKilled FROM GAMEDATA",  
 $"Save{Singleton.Instance.LoadNumber}.sqlite3")).Result.Split(";");  
 time.text = result[0];  
 monster.text = result[1];  
  
 difficulties.text = Singleton.Instance.Difficulty.ToString();  
 Singleton.Instance.init();  
 if (Singleton.Instance.GameOver)  
 {  
 gameOverPanel.SetActive(true);  
 PassedPanel.SetActive(false);  
 }  
 else  
 {  
 PassedPanel.SetActive(true);  
 gameOverPanel.SetActive(false);  
 }  
 }  
  
 private void Update()  
 {  
 if (Singleton.Instance.Loggedin)  
 CloseButton.interactable = true;  
 else CloseButton.interactable = false;  
 }  
  
 public void Upload()  
 {  
 var result = Task.Run(() =>Singleton.CsharpAPI.GetData(true,  
 "INSERT INTO `Progress`( `playerID`, `levelID`, `score`, `time`) VALUES (?,?,?,?)",  
 $"{Singleton.Instance.playerID},{difficulties.text},{monster.text},{time.text}")).Result;  
 Back();  
 Task.Run(() =>   
 Singleton.Localdb.Query("DROP TABLE GAMEDATA",  
 $"Save{Singleton.Instance.LoadNumber}.sqlite3")).Wait();  
 File.Delete(Application.persistentDataPath + $"/Save{Singleton.Instance.LoadNumber}.sqlite3");  
 File.Delete(Application.persistentDataPath + "/continue.txt");  
  
 }  
 public void ContinueButton()  
 {  
 if (Singleton.Instance.Loggedin)  
 {  
 Destroy(loginPanel);  
 Upload();  
 }  
 else Destroy(mainPanel);  
 }  
   
  
 void Back()  
 {  
 Singleton.Instance.init();  
 Singleton.LoadScreenclass.LoadScreen(true, false, 1, false, true,false);  
 }  
}

# exitTrigger.cs

using System.Collections;  
using System.Collections.Generic;  
using UnityEngine;  
  
public class exitTrigger : MonoBehaviour  
{  
 // Start is called before the first frame update  
 [SerializeField] private GameObject exit,passwordPanel;  
  
 private void OnTriggerEnter(Collider col)  
 {  
 passwordPanel = GameObject.Find("password");  
 passwordPanel.GetComponent<Canvas>().enabled = true;  
 }  
  
 public void DestroyWall()  
 {  
 int angle = FindObjectOfType<MazeManager>().Angle;  
 float deleteWall = Mathf.Floor((360f - angle) / 360f \* 176f);  
 for (int i = -3; i < 4; i++)  
 {  
 Destroy(GameObject.Find($"outerWall{(deleteWall + i)}"), 0);  
 }  
 Destroy(passwordPanel);  
 Destroy(gameObject);  
   
 }  
}

# gameOver.cs

using System;  
using System.Collections;  
using System.Collections.Generic;  
using System.Threading.Tasks;  
using UnityEngine;  
using UnityEngine.SceneManagement;  
  
public class gameOver : MonoBehaviour  
{  
 private void OnTriggerEnter(Collider other)  
 {  
 Singleton.Instance.init();  
 Singleton.Instance.GameOver = false;  
 var time = FindObjectOfType<Clock>().Time1;  
 var mons = FindObjectOfType<MazeManager>().MonsterKilled;  
 Task.Run(() =>   
 Singleton.Localdb.Query($"UPDATE GAMEDATA SET time = {time}, monsterKilled = {mons}",  
 $"Save{Singleton.Instance.LoadNumber}.sqlite3")).Wait();  
 Singleton.LoadScreenclass.LoadScreen(true, false, SceneManager.GetActiveScene().buildIndex + 1, true, true);  
 }  
}

# mainChar.cs

ï»¿using UnityEngine;  
  
namespace MazeScreen  
{  
 public class mainChar : MonoBehaviour  
 {  
 private int currentSlot;  
 }  
}

# MazeManager.cs

using System;  
using System.Collections.Generic;  
using System.IO;  
using System.Threading.Tasks;  
using TMPro;  
using Newtonsoft.Json;  
using UnityEngine;  
using UnityEngine.UI;  
using Random = UnityEngine.Random;  
using System.Linq;  
using algorithm;  
using UnityEngine.SceneManagement;  
using UnityEngine.Serialization;  
  
public class MazeManager : MonoBehaviour  
{  
 [Space(20)]  
   
 [Header("UI")]  
 private GameObject characterUI,empty;  
 [SerializeField] private GameObject characterSelectionUI,mainchar;  
   
 private List<Camera> cams = new List<Camera>();  
 private GameObject player;  
 private characterSpecification[] character;  
 private int currentselection = 0;  
 private int currentAttackSelection = 0;  
 [SerializeField] private Button selectcharButton;  
 private Inventory inven;  
 private int \_angle, \_monsterKilled = 0;  
 private CQueue open\_gate\_order;  
 private bool opennow;  
 private Vector3 playerPos;  
 private Setting \_setting;  
 private string \_password;  
 private Sprite[] sprites;  
 [SerializeField] private Sprite sprite1, sprite2;  
 [SerializeField]private Image attackSelection;  
 [SerializeField] private TMP\_Text startAt, NextGate,NextMonster,counting, monsterPrompt;  
  
 public int Angle  
 {  
 get => \_angle;  
 }  
  
 public Setting Setting1  
 {  
 get => \_setting;  
 set => \_setting = value;  
 }  
  
 public int MonsterKilled  
 {  
 get => \_monsterKilled;  
 set => \_monsterKilled = value;  
 }  
  
 public string Password  
 {  
 get => \_password;  
 set => \_password = value;  
 }  
  
 public void Dead()  
 {  
 Singleton.Instance.init();  
 Singleton.Instance.GameOver = true;  
 // Singleton.LoadScreenclass.LoadScreen(true, false, SceneManager.GetActiveScene().buildIndex + 1, true, true);  
 SceneManager.LoadScene(4);  
 Destroy(FindObjectOfType<playerhealth>());  
 }  
   
 public void spawnMonster()  
 {  
   
 float x = Random.Range(0f, 40f) - 20f, z = Random.Range(0f, 40f) - 20f;int mons = Random.Range(0,1);  
 GameObject[] monster = { Resources.Load<GameObject>("Tiny"), Resources.Load<GameObject>("Big") };  
 Instantiate(monster[mons],player.transform.position + new Vector3(x, 0, z),player.transform.rotation, GameObject.Find("monsterSpawner").gameObject.transform);  
 }  
 private void Awake()  
 {  
 empty = new GameObject();  
 characterUI = Resources.Load<GameObject>("UI/charactercard");  
 selectcharButton.onClick.AddListener(SelectedCharacter);  
 \_setting = FindObjectOfType<Setting>();  
 monsterPrompt.enabled = false;  
 }  
  
 public void MonsterPrompt(bool show)  
 {  
 if (show) monsterPrompt.enabled = true;  
 else monsterPrompt.enabled = false;  
 }  
 private void Start()  
 {  
 Singleton.Instance.init();  
 character = GetComponent<CharacterDataBase>().characters;  
  
 if (Singleton.Instance.FirstTime) FirstTime();// load default  
 else{//load setting  
 var result = Task.Run(() =>  
 Singleton.Localdb.Query("SELECT characterSelection, difficulty, monsterKilled,time,playerX,playerY,playerZ FROM GAMEDATA",  
 $"Save{Singleton.Instance.LoadNumber}.sqlite3")).Result.Split(new char[] { ',', ';' });  
 Destroy(characterSelectionUI);  
 try{playerPos = new Vector3(int.Parse(result[4]), int.Parse(result[5]), int.Parse(result[6]));}catch{}  
  
 FindObjectOfType<Clock>().time = float.Parse(result[3]);  
 \_monsterKilled = int.Parse(result[2]);  
 currentselection = int.Parse(result[0]);  
 Singleton.Instance.Difficulty = int.Parse(result[1]);  
 StartGame();  
 \_setting.LoadSetting(); // setting change apply  
 }  
 UpdateScore();  
 Singleton.HealthClass.maxHealth = 100f;  
 Singleton.HealthClass.chipspeed = 1.2f;  
 }  
  
 private void ShowCharacterPopup(int index)  
 {  
 characterSpecification details = character[index];  
 Singleton.Instance.ShowDetail(details.charName,  
 $"{details.charDescription}\n speed: {details.speed}\n Inventory:{details.InventorySize}",  
 details.charImage);   
 }  
  
 private void Selection(int index) => currentselection = index;  
  
 public void SelectedCharacter()  
 {  
 if(currentselection < 0) Singleton.Instance.ShowError("No selected character");  
 else  
 {  
 Task.Run(() =>  
 {  
 Singleton.Localdb.Query($"UPDATE GAMEDATA SET characterSelection = {currentselection};",  
 $"Save{Singleton.Instance.LoadNumber}.sqlite3");  
 }).Wait();  
 Destroy(characterSelectionUI);  
 StartGame();  
 }  
 }  
  
  
  
 private void Maze()  
 {  
 int[] mazeX = new int[8] { 50, 50, 50, 0, -50, -50, -50, 0 };  
 int[] mazeY = new int[8] { 50, 0, -50, -50, -50, 0, 50, 50 };  
 int[] mazeZ = new int[8] { 90, 90, 180, 180, 270, 270, 0, 0 };  
 GameObject mazeEmpty;  
 Transform ObjParent = GameObject.Find("objs").transform;  
 for (int i = 0; i < 8; i++)  
 {  
 mazeEmpty = Instantiate(empty);  
 mazeEmpty.name = "maze" + i;  
 mazeEmpty.transform.parent = ObjParent;  
 mazeEmpty.transform.position = new Vector3(mazeX[i], 0, mazeY[i]);  
  
 Maze mazeInstance = mazeEmpty.AddComponent<Maze>();  
 mazeInstance.sizeX = Singleton.Instance.Difficulty;  
 mazeInstance.sizeZ = Singleton.Instance.Difficulty;  
 mazeInstance.cellPrefab = Resources.Load<Mazecell>("Maze/floor");  
 mazeInstance.wallPrefab = Resources.Load<WallCell>("Maze/wall");  
 MazeGeneration gen;  
 mazeInstance.Generate();  
  
 if (Singleton.Instance.FirstTime)  
 {  
 gen = mazeEmpty.AddComponent<MazeGeneration>();  
 gen.myParent = mazeEmpty.transform;  
 gen.Wall = mazeInstance.walls;  
 gen.Cell = mazeInstance.cells;  
 try {  
 gen.Run();  
 Task.Run(() => Singleton.Localdb.Query($"ALTER TABLE GAMEDATA ADD Maze{i} TEXT",  
 $"Save{Singleton.Instance.LoadNumber}.sqlite3")).Wait();  
 }catch (Exception e) { Debug.Log(e); }  
  
 string walltemp = JsonConvert.SerializeObject(gen.Wall);  
 // Debug.Log(gen.Wall.Count +$"gen{i}");  
 Task.Run(() => Singleton.Localdb.Query($"UPDATE GAMEDATA SET Maze{i} = '{walltemp}'",  
 $"Save{Singleton.Instance.LoadNumber}.sqlite3")).Wait();  
 }  
 List<string> resultwall = JsonConvert.DeserializeObject<List<string>>(Task.Run(() =>  
 Singleton.Localdb.Query($"SELECT Maze{i} FROM GAMEDATA",  
 $"Save{Singleton.Instance.LoadNumber}.sqlite3")).Result.Split(';')[0]);  
 List<string> wall = new List<string>();  
 for (int x = 0; x < Singleton.Instance.Difficulty; x++)  
 {  
 for (int z = 0; z < Singleton.Instance.Difficulty; z++)  
 {  
 wall.Add(String.Format("Wall {0}-{1}-V", x, z));  
 wall.Add(String.Format("Wall {0}-{1}-H", x, z));  
 }  
 }  
  
 wall = wall.Except(resultwall).ToList();  
 foreach (var VARIABLE in wall)  
 {  
 try {  
 Destroy(mazeEmpty.transform.Find(VARIABLE).gameObject);  
 }  
 catch (Exception e) { }  
 }  
 mazeInstance.transform.localScale = new Vector3(50f / mazeInstance.sizeX, 1f, 50f / mazeInstance.sizeZ);  
 GameObject gate = Instantiate(Resources.Load<GameObject>("Maze/Gate"), mazeEmpty.transform);  
 // Debug.Log("gate?");Debug.Log(Singleton.Instance.FirstTime);  
 gate.transform.localScale = new Vector3(mazeInstance.sizeX / 50f, 1f, mazeInstance.sizeZ / 50f);  
 gate.transform.localPosition = new Vector3(mazeInstance.sizeX, 0, -mazeInstance.sizeX);  
 mazeEmpty.transform.rotation = Quaternion.Euler(0, mazeZ[i], 0);  
 }  
 open\_gate\_order = new CQueue(true, 8);  
 \_password = Task.Run(() => Singleton.Localdb.Query("SELECT orderOpening FROM GAMEDATA"  
 , $"Save{Singleton.Instance.LoadNumber}.sqlite3")).Result.Split(new char[] { ';', ',' })[0];  
 foreach (var VARIABLE in \_password)  
 {  
 open\_gate\_order.Enqueue(VARIABLE - 48);  
 }  
  
 startAt.text = open\_gate\_order.Peek().ToString();  
  
 }  
  
 public void StartGame()  
 {  
   
 Maze();  
 Camera cam;  
 try{mainchar.transform.position = playerPos;}catch{}  
 player = Instantiate(character[currentselection].actualObject, mainchar.transform);  
 minimap mini = mainchar.AddComponent<minimap>();   
 SpriteRenderer minimapPlayerLoc = Instantiate(empty,player.transform).AddComponent<SpriteRenderer>();  
 minimapPlayerLoc.sprite = Resources.Load<Sprite>("Sprite/Star");  
 minimapPlayerLoc.transform.position = new Vector3(0, 6.7f, 0);  
 minimapPlayerLoc.transform.rotation = Quaternion.Euler(new Vector3(90f,0,0));  
 minimapPlayerLoc.transform.localScale = new Vector3(0.5f, 0.5f, 0.1f);  
 minimapPlayerLoc.color = Color.cyan;  
 minimapPlayerLoc.gameObject.layer = LayerMask.NameToLayer("Minimap");  
 mini.Setup();  
   
 Rigidbody rb = FindObjectOfType<Rigidbody>();  
 if(rb == null) rb = mainchar.AddComponent<Rigidbody>();  
 rb.mass = 50f; rb.drag = 10f;  
 rb.angularDrag = 10f; rb.useGravity = true;  
 rb.collisionDetectionMode = CollisionDetectionMode.ContinuousDynamic;  
 rb.constraints = RigidbodyConstraints.FreezePositionY | RigidbodyConstraints.FreezeRotationX |  
 RigidbodyConstraints.FreezeRotationZ;  
 string[] camName = new[] { "FPP", "TPP" };  
 Vector3[] camLocation = new[] { new Vector3(0, 1.6f,0), new Vector3(0.46f, 1.88f,-1.5f) };  
 Vector3[] camRotation = new[] { new Vector3(0, 0, 0), new Vector3(16.1f, 0, 0) };  
 for (int i = 0; i < 2; i++)  
 {  
 cam = Instantiate(empty, player.transform).AddComponent<Camera>();  
 cam.name = camName[i];  
 cam.cullingMask = ~(1 << LayerMask.NameToLayer("Minimap"));  
 cam.transform.localPosition = camLocation[i];  
 cam.transform.localRotation = Quaternion.Euler(camRotation[i]);  
 cams.Add(cam);  
 }  
  
 movement movementScript = mainchar.AddComponent<movement>();  
 movementScript.walkspeed = 0.14f;  
 movementScript.jumpgravity = 9.8f;  
 movementScript.jumpheight = 5f;  
 movementScript.gravityspeed = 9.8f;  
 movementScript.crouchwalk = 0.24f;  
 movementScript.crouchheight = 0.5f;  
 turnaround turnScript = mainchar.AddComponent<turnaround>();  
 turnScript.cam = cams[0];  
 switchcam switchcamScript = gameObject.AddComponent<switchcam>();  
 switchcamScript.cams = cams.ToArray();  
 Button switchcamButton = GameObject.Find("SwitchCam").GetComponent<Button>();  
 switchcamButton.onClick.AddListener(switchcamScript.ChangeCamera);  
 //UI  
 Button jumpButton = GameObject.Find("Jump").GetComponent<Button>();  
 jumpButton.onClick.AddListener(movementScript.Jumpbutton);  
 Button crouchbutton = GameObject.Find("Crouch").GetComponent<Button>();  
 crouchbutton.onClick.AddListener(movementScript.crouch);  
 GameObject escapeGp = empty;  
 escapeGp = Instantiate(Resources.Load<GameObject>("Maze/escapeGp"));  
 \_angle = int.Parse(Task  
 .Run(() => Singleton.Localdb.Query("SELECT exit FROM GAMEDATA",  
 $"Save{Singleton.Instance.LoadNumber}.sqlite3")).Result.Split(new char[] { ',', ';' })[0]);  
 escapeGp.transform.localRotation = Quaternion.Euler( \_angle \* Vector3.up);  
 escapeGp.transform.parent = GameObject.Find("objs").transform;  
 \_setting.LoadSetting();  
 if(Singleton.Instance.FirstTime)  
 {  
 Singleton.Instance.init();  
 Singleton.Instance.ContinuePrompt = true;  
 Back();  
   
 }  
 openGate();  
 }  
  
 public void OpenPassword()  
 {  
   
 }  
 private void FirstTime()  
 {//exit  
 try  
 {  
 Task.Run(() => Singleton.Localdb.Query("ALTER TABLE GAMEDATA ADD exit INTEGER ", $"Save{Singleton.Instance.LoadNumber}.sqlite3")).Wait();  
 }  
 catch (Exception e) { }  
 int ranNumber = Random.Range(1, 360) ;  
 Task.Run(() => Singleton.Localdb.Query(  
 $@"UPDATE GAMEDATA SET exit= {ranNumber};", $"Save{Singleton.Instance.LoadNumber}.sqlite3")).Wait();  
 //selection Screen  
 GameObject showcase = GameObject.Find("charShowCase");  
 for (int i = 0; i < character.Length; i++)  
 {  
 characterSpecification details = character[i];  
 int dummyi = i;  
 GameObject tempChar = Instantiate(characterUI, showcase.transform);  
 tempChar.name = String.Format("Selector-{0}", details.actualObject.name);  
 TMP\_Text chartext = tempChar.GetComponentInChildren<TextMeshProUGUI>();  
 chartext.text = details.charName;  
 if (details.charImage != null)  
 {  
 Image showImage = tempChar.GetComponentInChildren<Image>();  
 showImage.overrideSprite = details.charImage;  
 }  
  
 charcard selection = tempChar.GetComponentInChildren<charcard>();  
 selection.popup.onClick.AddListener(() => ShowCharacterPopup(dummyi));  
 selection.selection.onClick.AddListener((() => Selection(dummyi)));  
 }  
 int[] seqence = {0,1, 2, 3, 4, 5, 6, 7}; // Replace with your own seqence  
   
 for (int i = seqence.Length - 1; i > 0; i--)  
 {  
 int j = Random.Range(0, i + 1);  
 (seqence[i], seqence[j]) = (seqence[j], seqence[i]);  
 }  
 Task.Run(() => Singleton.Localdb.Query($"ALTER TABLE GAMEDATA ADD orderOpening TEXT",  
 $"Save{Singleton.Instance.LoadNumber}.sqlite3")).Wait();  
 string uploadData= "";  
 foreach (var VARIABLE in seqence)  
 {  
 uploadData += VARIABLE.ToString();  
 }  
 // Debug.Log(uploadData);  
 Task.Run(()=> Singleton.Localdb.Query($"UPDATE GAMEDATA SET orderOpening = '{uploadData}'",  
 $"Save{Singleton.Instance.LoadNumber}.sqlite3")).Wait();  
 Task.Run(() => Singleton.Localdb.Query($"ALTER TABLE GAMEDATA ADD monsterKilled INTEGER ",  
 $"Save{Singleton.Instance.LoadNumber}.sqlite3")).Wait();  
 }  
  
 public void SaveLevel()  
 {  
 Task.Run(()=> Singleton.Localdb.Query($"UPDATE GAMEDATA SET monsterKilled = '{\_monsterKilled}'",  
 $"Save{Singleton.Instance.LoadNumber}.sqlite3")).Wait();  
 Vector3 tempPos = player.transform.position;  
 Task.Run(()=> Singleton.Localdb.Query(@$"UPDATE GAMEDATA SET playerX = {tempPos.x},  
 playerY = {tempPos.y},playerZ= {tempPos.z}",  
 $"Save{Singleton.Instance.LoadNumber}.sqlite3")).Wait();  
 try{Task.Run(() => Singleton.Localdb.Query($"ALTER TABLE GAMEDATA ADD time FLOAT ",  
 $"Save{Singleton.Instance.LoadNumber}.sqlite3")).Wait();}catch{}  
  
 float time = FindObjectOfType<Clock>().Time1;  
 Task.Run(() => Singleton.Localdb.Query($"UPDATE GAMEDATA SET time = {time} ",  
 $"Save{Singleton.Instance.LoadNumber}.sqlite3")).Wait();  
 }  
   
 public void Attack()  
 {  
 projectile projectile\_script = FindObjectOfType<projectile>();  
 switch (currentAttackSelection)  
 {  
 case 0:  
 projectile\_script.Shoot(Resources.Load<GameObject>("Item/attach/Axe\_03"),player.transform,10);  
 break;  
 case 1:  
 Debug.Log(player.transform.rotation.z);  
 Instantiate(Resources.Load<GameObject>("Item/attach/Pitchfork\_01"),player.transform.position + new Vector3(0.4f,0.5f,0),  
 Quaternion.Euler(Quaternion.ToEulerAngles(player.transform.rotation)+ new Vector3(90, 0, 0)),GameObject.Find("objs").transform).tag="Trap";  
 break;  
 }  
 }  
  
 public void changeAttack(int index)  
 {sprites = new[] { sprite1, sprite2 };attackSelection.sprite = sprites[currentAttackSelection];  
 currentAttackSelection = index;  
 }  
  
 public void heal()  
 {  
 Singleton.HealthClass.changeHP(10);  
 }  
public void openGate()  
 {  
 open\_gate[] instances = FindObjectsOfType<open\_gate>();  
 foreach (var VARIABLE in instances)  
 {  
 VARIABLE.Open = false;  
 }  
  
 int gateNumber = open\_gate\_order.Peek();  
 open\_gate ins = GameObject.Find("maze" + gateNumber).GetComponentInChildren<open\_gate>();  
   
 ins.Open = true;  
 open\_gate\_order.Enqueue(open\_gate\_order.Dequeue());  
 NextGate.text = open\_gate\_order.Peek().ToString();  
 }  
public void Back()  
{  
 SaveLevel();  
 SceneManager.LoadScene(1);  
}  
public void UpdateScore()  
{  
 if (counting.text != \_monsterKilled.ToString())  
 {  
 counting.text = \_monsterKilled.ToString();  
 Task.Run(()=> Singleton.Localdb.Query($"UPDATE GAMEDATA SET monsterKilled = '{\_monsterKilled}'",  
 $"Save{Singleton.Instance.LoadNumber}.sqlite3"));  
 }  
  
}  
}

# minimap.cs

using System.Collections;  
using System.Collections.Generic;  
using UnityEngine;  
using UnityEngine.UI;  
  
public class minimap : MonoBehaviour  
{  
 Camera minimap\_cam;  
 RenderTexture texture;  
 RawImage img;  
  
 public Camera MinimapCam  
 {  
 get => minimap\_cam;  
 set => minimap\_cam = value;  
 }  
  
 public void Setup()  
 {  
 GameObject empty = new GameObject();  
 minimap\_cam = Instantiate(empty, this.transform).AddComponent<Camera>();  
 minimap\_cam.name = "Minicam";  
   
 minimap\_cam.transform.localPosition = new Vector3(0, 20f, 0);  
 minimap\_cam.transform.localRotation = Quaternion.Euler(90,0,0);  
 minimap\_cam.orthographic = true;  
 minimap\_cam.cullingMask = (1 << LayerMask.NameToLayer("Minimap"))|(1<<LayerMask.NameToLayer("Ground"));  
 texture = new RenderTexture(446,233,16,RenderTextureFormat.ARGB32);  
 minimap\_cam.targetTexture = texture;  
 img = GameObject.Find("Minimap").GetComponent<RawImage>();  
 img.texture = texture;  
 }  
}

# onLoadMaze.cs

using System;  
using System.Collections;  
using System.Collections.Generic;  
using MenuScreen;  
using UnityEngine;  
using UnityEngine.EventSystems;  
  
public class onLoadMaze : MonoBehaviour  
{  
 void Awake()  
 {  
 GameObject singleton\_ins = Singleton.Instance.gameObject;  
 singleton\_ins.GetComponent<Singleton>().init();  
 try  
 {  
 singleton\_ins.AddComponent<EventSystem>();  
 singleton\_ins.AddComponent<StandaloneInputModule>();  
 }  
 catch (Exception e)  
 {  
 Console.WriteLine(e);  
 throw;  
 }  
 }  
}

# open\_gate.cs

using System;  
using System.Collections;  
using System.Collections.Generic;  
using UnityEngine;  
  
public class open\_gate : MonoBehaviour  
{  
 [SerializeField] private Transform m\_left, m\_right, back;  
 [SerializeField] private bool open;  
 [SerializeField] private int Speed;  
 [SerializeField] private Vector3 m\_vecRight\_OpenPos, m\_vecRight\_ClosePos, m\_vecLeft\_OpenPos, m\_vecLeft\_ClosePos;  
  
 public bool Open  
 {  
 get => open;  
 set => open = value;  
 }  
  
 private void Update()  
 {  
 if (open)  
 {  
 m\_right.localPosition = Vector3.Lerp(m\_right.localPosition, m\_vecRight\_OpenPos,  
 Time.deltaTime \* Speed);  
 m\_left.localPosition =  
 Vector3.Lerp(m\_left.localPosition, m\_vecLeft\_OpenPos, Time.deltaTime \* Speed);  
 back.gameObject.SetActive(false);  
 }  
 else  
 {  
 m\_right.localPosition = Vector3.Lerp(m\_right.localPosition, m\_vecRight\_ClosePos,  
 Time.deltaTime \* Speed);  
 m\_left.localPosition =  
 Vector3.Lerp(m\_left.localPosition, m\_vecLeft\_ClosePos, Time.deltaTime \* Speed);  
 back.gameObject.SetActive(true);  
 }  
 }  
}

# passwordPanel.cs

using System;  
using System.Collections;  
using System.Collections.Generic;  
using TMPro;  
using UnityEngine;  
  
public class passwordPanel : MonoBehaviour  
{  
 [SerializeField]private TMP\_InputField inputext;  
 private string password;  
 private bool done = false;  
 private void Start()  
 {  
 password = FindObjectOfType<MazeManager>().Password;  
   
 }  
 private void Update()  
 {  
 // Debug.Log(password);  
 if (inputext.text == password)  
 {  
 FindObjectOfType<exitTrigger>().DestroyWall();  
 }   
 }  
}

# pause.cs

using UnityEngine;  
  
public class pause : MenuPanels  
{  
 private LeaderBoarddata leader;  
 private void Awake()  
 {  
 \_menu = gameObject;  
 leader = FindObjectOfType<LeaderBoarddata>();  
 }  
  
 public override void Start()  
 {  
   
 base.Start();  
 }  
  
 public void timeChange(float time)  
 {  
 Time.timeScale = time;  
 }  
  
 public void Open()  
 {  
 timeChange(0);  
 ShowHide(true);  
 }  
  
 public void Close()  
 {  
 timeChange(1);  
 ShowHide(false);  
 }  
 public void SettingPause()  
 {  
 FindObjectOfType<MazeManager>().Setting1.gameObject.SetActive(true);  
 }  
  
 public void LeaderboardPause()  
 {  
 leader.ShowHide(true);  
 }  
  
 public void Back()  
 {  
 FindObjectOfType<MazeManager>().Back();  
 }  
  
 // Update is called once per frame  
 void Update()  
 {  
   
 }  
}

# PerlinNoiseMap.cs

ï»¿using UnityEngine;  
  
namespace MazeScreen  
{  
 public class PerlinNoiseMap : MonoBehaviour  
 {  
 public int width;  
 public int height;  
 public float scale;  
 public float persistence;  
 public float lacunarity;  
 public int octaves;  
  
 public float[,] noiseMap;  
  
 void Start()  
 {  
 noiseMap = new float[width, height];  
 generatePerlinNoiseMap();  
 }  
  
 void generatePerlinNoiseMap()  
 {  
 for (int x = 0; x < width; x++)  
 {  
 for (int y = 0; y < height; y++)  
 {  
 float amplitude = 1;  
 float frequency = 1;  
 float noiseHeight = 0;  
  
 for (int i = 0; i < octaves; i++)  
 {  
 float xCoord = (float)x / scale \* frequency;  
 float yCoord = (float)y / scale \* frequency;  
  
 float perlinValue = Mathf.PerlinNoise(xCoord, yCoord);  
 noiseHeight += perlinValue \* amplitude;  
  
 amplitude \*= persistence;  
 frequency \*= lacunarity;  
 }  
  
 noiseMap[x, y] = noiseHeight;  
 }  
 }  
 }  
 }  
 }

# PickUp.cs

using System;  
using System.Collections;  
using System.Collections.Generic;  
using MazeScreen.movement;  
using UnityEngine;  
using UnityEngine.UI;  
  
public class PickUp : MenuPanels  
{  
 public List<item> items;  
 private PickUpSelectionSlot \_selectionSlot;  
 public override void Start()  
 {  
   
 \_selectionSlot = GetComponent<PickUpSelectionSlot>();  
 Button toggler = GameObject.Find("PickUpToggle").GetComponent<Button>();  
 toggler.onClick.AddListener(Toggle);  
 base.Start();  
 }  
 public void Update()  
 {  
 if (items.Count != 0)  
 ShowHide(true);else  
 ShowHide(false);  
 }  
 public void Toggle()  
 {  
 \_selectionSlot.Toggle();  
 }   
}

# PickUpSelectionSlot.cs

ï»¿using UnityEngine;  
  
namespace MazeScreen.movement  
{  
 public class PickUpSelectionSlot : MenuPanels  
 {  
 public override void Start()  
 {  
 ShowHide(true);  
 }  
  
 public void Toggle()  
 {  
 if (\_menu) ShowHide(false);  
 else ShowHide(true);  
 }  
 }  
}

MazeGame > Inventory

# CharacterDataBase.cs

ï»¿using System;  
using UnityEngine;  
using UnityEngine.UI;  
  
public class CharacterDataBase : MonoBehaviour  
{  
 public characterSpecification[] characters;  
  
 public GameObject GetGameObject(int index)  
 {  
 return characters[index].actualObject;  
 }  
  
  
}

# characterSpecification.cs

ï»¿using System;  
using UnityEngine;  
using UnityEngine.UI;  
[Serializable]  
public class characterSpecification  
{   
 public Sprite charImage;  
 public string charName, charDescription;  
 public float speed;  
 public int InventorySize;  
 public GameObject actualObject;  
}

# charcard.cs

ï»¿using UnityEngine;  
using UnityEngine.UI;  
  
 public class charcard : MonoBehaviour  
 {  
 public Button popup, selection;  
 }

# Inventory.cs

using System;  
using System.Collections;  
using System.Collections.Generic;  
using TMPro;  
using UnityEngine;  
  
public class Inventory : MenuPanels  
{  
 public struct InventoyItem {  
 public item itemDetail;  
 public int quantity;  
 }  
   
 private InventoyItem[] inventoryItems;  
 [SerializeField] private Slot SlotUI;  
 private int maxsize;  
  
 public InventoyItem[] InventoryItems  
 {  
 get => inventoryItems;  
 set => inventoryItems = value;  
 }  
  
 public override void Start() { }  
 public void Setup(int size)  
 {  
 maxsize = size;  
 GameObject slotsParent = GameObject.Find("InventorySlots");  
 for (int i = 0; i < size; i++)  
 {  
 Slot tempslot = Instantiate(SlotUI, slotsParent.transform);  
 if(i < 2)  
 {  
 tempslot.frame.color = Color.cyan;  
 tempslot.mainSlot = true;  
 }  
 tempslot.gameObject.name = $"SlotN{i}";  
 tempslot.stacksizeUI.color = Color.yellow;  
 tempslot.stacksizeUI.text = "x0";  
 }  
 ShowHide(false);  
 }  
 void pickupitem(item itemdetail)  
 {  
 int itemSlotLoc;  
 bool stack = false;   
 InventoyItem item = new InventoyItem();  
 foreach (var VARIABLE in inventoryItems) {  
 if(VARIABLE.itemDetail == itemdetail)  
 {  
 stack = true;  
 item = VARIABLE;  
 }  
 }  
  
 if (stack)  
 {  
 if (item.quantity >= itemdetail.maxStack)  
 {  
   
 }  
 else  
 {  
 //input  
 }  
 }  
 else  
 {  
 if (inventoryItems.Length >= maxsize)  
 {  
 //reject  
 }else  
 {  
 //input  
 }  
 }  
   
   
 }  
 void removeitem(string s, int quantity){  
 for (int i = 0; i < inventoryItems.Length; i++)  
 {  
 // if(jerk[i].object1.name)  
 }  
 }  
   
   
}

# item.cs

ï»¿using UnityEngine;  
using UnityEngine.UI;  
  
[System.Serializable]  
public class item : MonoBehaviour  
{  
 public int maxStack ;  
 public Image itemImage;  
 private GameObject obj;  
 public string itemName ,itemDescription;  
 public float Weight, damage;  
 public bool Weapon, throwable, shootable;  
 private PickUp pickupScript;  
   
 [SerializeField] private float triggerradius = 1f;  
  
 public bool Shootable  
 {  
 get => shootable;  
 set => shootable = value;  
 }  
  
 public bool Throwable  
 {  
 get => throwable;  
 set => throwable = value;  
 }  
  
 public GameObject Obj  
 {  
 get => obj;  
 set => obj = value;  
 }  
  
 private void OnEnable()  
 {  
 pickupScript = FindObjectOfType<PickUp>();  
 SphereCollider trigger = gameObject.AddComponent<SphereCollider>();  
 trigger.isTrigger = true;  
 trigger.radius = triggerradius;  
 if (throwable)  
 {  
 // AxeThrow throw  
 }  
 }  
  
 private void OnTriggerEnter(Collider other)  
 {  
 pickupScript.items.Add(gameObject.GetComponent<item>());  
 }  
   
  
 private void OnTriggerExit(Collider other)  
 {  
 pickupScript.items.Remove(gameObject.GetComponent<item>());  
 }  
}

# itemsObject.cs

ï»¿using System;  
using UnityEngine;  
  
namespace MazeScreen.Inventory  
{  
 public class itemsObject : MonoBehaviour  
 {  
 }  
}

# Slot.cs

ï»¿using System;  
using TMPro;  
using UnityEngine;  
using UnityEngine.UI;  
  
public class Slot : MonoBehaviour  
{  
 public bool mainSlot;  
 public item itemdetail;  
 public int stacksize;  
 public TMP\_Text stacksizeUI;  
 public Image itemImage, frame;  
}

MazeGame > Maze

# inte.cs

using System;  
using System.Collections.Generic;  
using System.Linq;  
using UnityEngine;  
  
public class djk : MonoBehaviour  
{  
 public List<String> wall;  
 private Mazecell[,] cell;  
 public List<Mazecell> unvisitedcell;  
  
 //nodex y visited cost previous  
 public void djkinit(int startx, int starty, int endx, int endy, List<String> wall,Mazecell[,] cell)  
 {  
 this.cell = cell;  
 this.wall = wall;  
 // unvisitedcell = new List<Mazecell>();  
 for (int x = 0; x < cell.GetLength(0); x++)  
 {  
 for (int y = 0; y < cell.GetLength(1); y++)  
 {  
 bool xWall = false, yWall = false;  
 if (wall.Contains($"Wall {x}-{y}-V"))  
 AvaliableCell(x, y, true);  
 if (wall.Contains($"Wall {x}-{y}-H"))  
 AvaliableCell(x, y, false);  
 if (x == 0) Sidebar(true, true,x,y);  
 if(y==0) Sidebar(false,false,x,y);  
 if(x==cell.GetLength(0)-1) Sidebar(true, false,x,y);  
 if(y==cell.GetLength(1)-1) Sidebar(false, true,x,y);  
 }  
 }  
 foreach (var IndividualCell in cell)  
 {  
 List<int> djkdirXList = new List<int>(), djkdirYList = new List<int>();  
 foreach (var VARIABLE in IndividualCell.djkdirX)  
 {  
 if (VARIABLE != 5)  
 djkdirXList.Add(VARIABLE);  
 }  
 foreach (var VARIABLE in IndividualCell.djkdirY)  
 {  
 if (VARIABLE != 5)  
 djkdirYList.Add(VARIABLE);  
 }  
  
 IndividualCell.djkdirX = djkdirXList.ToArray();  
 IndividualCell.djkdirY = djkdirYList.ToArray();  
 }  
  
 unvisitedcell = new List<Mazecell>();  
 this.cell[startx, starty].djkmincost = 0;  
 Search(startx, starty,endx, endy);  
 }  
  
 void Search(int currX, int currY, int endX, int endY)  
 {  
 Mazecell current = cell[currX, currY];  
 Stack unvisited = new Stack(100),unvistedLoop = new Stack(1000);  
 if (currX == endX && currY == endY)return;  
  
 if (current.djkvisited) return;  
 current.djkvisited = true;  
 if (current.djkdirX.Any()) {  
 Vector2 temp;  
 for (int i = 0; i < current.djkdirX.Length; i++)  
 {  
 temp = new Vector2(currX + current.djkdirX[i],  
 currY + current.djkdirY[i]);  
 try{if (!unvisited.Contains(temp) && cell[(int)temp.x,(int)temp.y].djkvisited == false)  
 unvisited.Push(temp);}catch{}  
 }  
 unvistedLoop = unvisited;  
 }  
 if (unvisited.Peek() != null)  
 {  
 calcost(currX,currY, (Vector2)unvisited.Pop());//maybe underflow  
 }  
  
 Vector2 next;  
 if(unvisited.Peek() != null){next = (Vector2)unvistedLoop.Pop();  
 Search((int)next.x,(int)next.y,endX, endY);}  
 }  
   
   
  
 void calcost(int currentx, int currenty,Vector2 cellPosition)  
 {  
 Debug.Log(currentx +""+ currenty +""+cellPosition);  
 try  
 {  
 Mazecell neighbour = cell[currentx + (int)cellPosition.x, currenty + (int)cellPosition.y],  
 current = cell[currentx, currentx];  
 float cost = current.djkmincost + obcheck((int)cellPosition.x, (int)cellPosition.y);  
 if (neighbour.djkmincost > cost)  
 {  
 neighbour.djkmincost = cost;  
 neighbour.djkprevious = new Vector2(currentx, currenty);  
 }  
 }catch { }  
 }  
  
 int obcheck(int xPos, int yPos)  
 {  
 switch ((xPos, yPos))  
 {  
 case (1, 1):  
 case (1, -1):  
 case (-1, -1):  
 case (-1, 1):  
 return 14;  
 case (0, -1):  
 case (1, 0):  
 case (-1, 0):  
 case (0, 1):  
 return 10;  
 default:  
 return 0;  
 }  
 }  
   
   
   
   
   
   
 void Sidebar(bool isVertical, bool isLeftorTop, int xPos, int yPos)  
 {  
 switch ((isVertical, isLeftorTop))  
 {  
 case (true, true) when isVertical == isLeftorTop:  
 cell[xPos, yPos].djkdirX[4] = 5; cell[xPos, yPos].djkdirY[4] = 5;  
 cell[xPos, yPos].djkdirX[5] = 5; cell[xPos, yPos].djkdirY[5] = 5;  
 cell[xPos, yPos].djkdirX[6] = 5; cell[xPos, yPos].djkdirY[6] = 5;  
 break;  
 case (false, false) when isVertical == isLeftorTop:  
 cell[xPos, yPos].djkdirX[2] = 5; cell[xPos, yPos].djkdirY[2] = 5;  
 cell[xPos, yPos].djkdirX[3] = 5; cell[xPos, yPos].djkdirY[3] = 5;  
 cell[xPos, yPos].djkdirX[4] = 5; cell[xPos, yPos].djkdirY[4] = 5;  
 break;  
 case (false, true):  
 cell[xPos, yPos].djkdirX[6] = 5; cell[xPos, yPos].djkdirY[6] = 5;  
 cell[xPos, yPos].djkdirX[7] = 5; cell[xPos, yPos].djkdirY[7] = 5;  
 cell[xPos, yPos].djkdirX[0] = 5; cell[xPos, yPos].djkdirY[0] = 5;  
 break;  
 case (true, false):   
 cell[xPos, yPos].djkdirX[0] = 5; cell[xPos, yPos].djkdirY[0] = 5;  
 cell[xPos, yPos].djkdirX[1] = 5; cell[xPos, yPos].djkdirY[1] = 5;  
 cell[xPos, yPos].djkdirX[2] = 5; cell[xPos, yPos].djkdirY[2] = 5;  
 break;  
 default:  
 Debug.Log("One or both measurements are not valid.");  
 break;  
 }  
 }  
 void AvaliableCell( int xPos, int yPos, bool isVertical)  
 {  
 if (isVertical)  
 {  
 cell[xPos, yPos].djkdirX[6] = 5; cell[xPos, yPos].djkdirY[6] = 5;  
 cell[xPos, yPos].djkdirX[7] = 5; cell[xPos, yPos].djkdirY[7] = 5;  
 cell[xPos, yPos].djkdirX[0] = 5; cell[xPos, yPos].djkdirY[0] = 5;  
 try{cell[xPos, yPos+1].djkdirX[2] = 5; cell[xPos, yPos+1].djkdirY[2] = 5;  
 cell[xPos, yPos+1].djkdirX[3] = 5; cell[xPos, yPos+1].djkdirY[3] = 5;  
 cell[xPos, yPos+1].djkdirX[4] = 5; cell[xPos, yPos+1].djkdirY[4] = 5;}catch{ }  
 try{cell[xPos-1, yPos+1].djkdirX[2] = 5; cell[xPos-1, yPos+1].djkdirY[2] = 5;}catch{ }  
 try{cell[xPos-1, yPos].djkdirX[0] = 5; cell[xPos-1, yPos].djkdirY[0] = 5;}catch{ }  
 try{cell[xPos+1, yPos+1].djkdirX[4] = 5; cell[xPos+1, yPos+1].djkdirY[4] = 5;}catch{ }  
 try{cell[xPos+1, yPos].djkdirX[6] = 5; cell[xPos+1, yPos].djkdirY[6] = 5;}catch{ }  
 }else if (!isVertical)  
 {  
 cell[xPos, yPos].djkdirX[0] = 5; cell[xPos, yPos].djkdirY[0] = 5;  
 cell[xPos, yPos].djkdirX[1] = 5; cell[xPos, yPos].djkdirY[1] = 5;  
 cell[xPos, yPos].djkdirX[2] = 5; cell[xPos, yPos].djkdirY[2] = 5;  
 try{cell[xPos+1, yPos].djkdirX[4] = 5; cell[xPos+1, yPos].djkdirY[4] = 5;  
 cell[xPos+1, yPos].djkdirX[5] = 5; cell[xPos+1, yPos].djkdirY[5] = 5;  
 cell[xPos+1, yPos].djkdirX[6] = 5; cell[xPos+1, yPos].djkdirY[6] = 5;}catch{ }  
 try{cell[xPos, yPos-1].djkdirX[0] = 5; cell[xPos, yPos-1].djkdirY[0] = 5;}catch{ }  
 try{cell[xPos+1, yPos-1].djkdirX[6] = 5; cell[xPos+1, yPos-1].djkdirY[6] = 5;}catch{ }  
 try{cell[xPos, yPos+1].djkdirX[2] = 5; cell[xPos, yPos+1].djkdirY[2] = 5;}catch{ }  
 try{cell[xPos+1, yPos+1].djkdirX[4] = 5; cell[xPos+1, yPos+1].djkdirY[4] = 5;}catch{ }  
 }  
 }  
}

# goto.cs

using System.Collections;  
using System.Collections.Generic;  
using UnityEngine;  
  
public class returndijsktra : MonoBehaviour  
{  
   
 bool shouldmove;  
 Vector2 pos;  
 float speed;  
 GameObject hello;  
  
 public goto12(Vector2 \_pos, float \_speed, GameObject \_hello){  
 hello = \_hello;  
 pos = \_pos;  
 speed = \_speed;  
 Debug.Log("setup complete");  
 }  
 void Update(){  
 Move();  
 }  
  
 void Move(){  
 Transform hell = hello.transform;  
 Vector3 gotopos = new Vector3(pos.x, 0, pos.y);  
 if(hell.position == gotopos){  
 Destroy(this);  
 }  
 hell.LookAt(hello.transform, gotopos);  
 Vector3 trans = hell.TransformDirection(gotopos);  
 hell.position = new Vector3(hell.position.x + trans.x, hell.position.y, hell.position.z + trans.z);  
 }  
}

# Maze.cs

using System.Collections;  
using System.Collections.Generic;  
using UnityEngine;  
using System;  
using System.Linq;  
  
public class Maze : MonoBehaviour  
{  
 private int \_sizeX;  
 private int \_sizeZ;  
 private Mazecell \_cellPrefab;  
 private Mazecell[,] \_cells; //need to access by others  
 private List<String> \_walls;  
 private WallCell \_wallPrefab;  
 // public float generationStepDelay = 0;  
 public int sizeX  
 {  
 get => \_sizeX;  
 set => \_sizeX = value;  
 }  
 public int sizeZ  
 {  
 get => \_sizeZ;  
 set => \_sizeZ = value;  
 }  
 public Mazecell cellPrefab  
 {  
 get => \_cellPrefab;  
 set => \_cellPrefab = value;  
 }  
 public WallCell wallPrefab  
 {  
 get => \_wallPrefab;  
 set => \_wallPrefab = value;  
 }  
  
 public List<string> walls  
 {  
 get => \_walls;  
 set => \_walls = value;  
 }  
  
 public Mazecell[,] cells  
 {  
 get => \_cells;  
 set => \_cells = value;  
 }  
  
 public void Generate (bool generate = true) {  
 // WaitForSeconds delay = new WaitForSeconds(generationStepDelay);  
 \_walls = new List<String>();  
 \_cells = new Mazecell[\_sizeX, \_sizeZ];  
 for (int x = 0; x < \_sizeX; x++) {  
 for (int z = 0; z < \_sizeZ; z++) {  
 // yield return delay;  
 CreateCell(x, z, generate);  
 }  
 }  
 }  
  
 private void CreateCell (int x, int z, bool generate)  
 {  
 Vector3 scale= \_cellPrefab.transform.localScale;  
 if (x >= 0 && z >= 0)  
 {  
 Mazecell newCell = Instantiate(\_cellPrefab) as Mazecell;  
 cells[x, z] = newCell;  
 newCell.name = $"Cell {x}-{z}";  
 newCell.transform.parent = transform;  
 // newCell.transform.localPosition = new Vector3(x\*scale.x - \_sizeX \* 0.5f \*scale.x + scale.x/2, 0f, z\*scale.z - \_sizeZ \* 0.5f\*scale.z + scale.z/2);  
 newCell.transform.localPosition = new Vector3(x \* scale.x + 0.5f \* scale.x, 0f, z \* scale.z + 0.5f \* scale.z);  
 if (z == 0) newCell.dir.Remove("left");  
 if (x == 0) newCell.dir.Remove("up");  
 if (z == \_sizeX - 1) newCell.dir.Remove("right");  
 if (x == \_sizeZ - 1) newCell.dir.Remove("down");  
 }  
  
 if (x != \_sizeX - 1)  
 {  
 WallCell newWall = Instantiate(\_wallPrefab) as WallCell;  
 newWall.name = String.Format("Wall {0}-{1}-H", x, z);  
 newWall.transform.parent = transform;  
 newWall.transform.localRotation = Quaternion.Euler(new Vector3(90, 90, 0));  
 newWall.transform.localPosition = new Vector3(x \* scale.x + scale.x, 0.5f, z \* scale.z + scale.z / 2);  
 walls.Add(newWall.name);  
 }  
  
 if (z != \_sizeZ - 1)  
 {  
 WallCell newWall1 = Instantiate(\_wallPrefab) as WallCell;  
 newWall1.name = String.Format("Wall {0}-{1}-V", x, z);  
 newWall1.transform.parent = transform;  
 newWall1.transform.localRotation = Quaternion.Euler(new Vector3(90, 0, 0));  
 newWall1.transform.localPosition = new Vector3(x \* scale.x + scale.x / 2, 0.5f, z \* scale.z + scale.z);  
 walls.Add(newWall1.name);  
} } }

# Mazecell.cs

using System.Collections;  
using System.Collections.Generic;  
using UnityEngine;  
  
public class Mazecell : MonoBehaviour{  
 public bool visited = false;  
 public List<string> dir = new List<string>{"up","down","left","right"};  
 // bool up, down, left, right, upleft, upright, botleft,botright;  
   
 //dijstrka  
 public float djkmincost = float.MaxValue;  
 public Vector2 djkprevious;  
 public bool djkvisited = false;  
 public int[] djkdirX = { 1, 1, 1, 0, -1, -1, -1, 0 };  
 public int[] djkdirY = { 1, 0, -1, -1, -1, 0, 1, 1 };  
   
  
 //Astar  
 public float ghost, hcost, fcost;  
 public Vector2 Aprevious;  
 public bool Avisited;  
 public bool[] adir = new bool[8]{ false,false,false,false,true,true,true,true};  
}

# MazeGeneration.cs

using System;  
using System.Collections;  
using System.Collections.Generic;  
using UnityEngine;  
using System.Linq;  
using Newtonsoft.Json;  
using Random = UnityEngine.Random;  
  
public class MazeGeneration : MonoBehaviour  
{  
 [SerializeField] private Transform \_myParent;  
 [SerializeField] private List<String> wall;  
 [SerializeField] private Mazecell[,] cell ;  
 private Stack stack1 = new(100);  
  
 public Transform myParent  
 {  
 get => \_myParent;  
 set => \_myParent = value;  
 }  
  
 public List<string> Wall  
 {  
 get => wall;  
 set => wall = value;  
 }  
  
 public Mazecell[,] Cell  
 {  
 get => cell;  
 set => cell = value;  
 }  
  
 private djk algo;  
 public void Run()  
 {  
 BuildMaze(0,0);  
 }  
 void BuildMaze(int x, int y){   
 Mazecell currentcell = cell[x,y];  
 currentcell.visited = true; //set the cell as visited  
 stack1.Push(currentcell);  
 while(stack1.pointer>=0){//runs until the stack is empty,all mazecell will be explored   
 currentcell = stack1.Pop() as Mazecell;  
 Checkdir(x,y);  
 if(currentcell.dir.Any()){  
 stack1.Push(currentcell);  
 int element = Random.Range(0, currentcell.dir.Count());  
 string now = currentcell.dir[element];  
 currentcell.dir.RemoveAt(element);  
 // Debug.Log(x+""+y);  
 try  
 {  
 switch (now) //change the record of list of wall  
 {  
 case "up":  
 wall.Remove(String.Format("Wall {0}-{1}-H", x - 1, y));  
 cell[x - 1, y].visited = true;  
  
 stack1.Push(cell[x - 1, y]);  
 x -= 1;  
 break;  
 case "down":  
 wall.Remove(System.String.Format("Wall {0}-{1}-H", x, y));  
 cell[x + 1, y].visited = true;  
  
 stack1.Push(cell[x + 1, y]);  
 x++;  
 break;  
 case "left":  
 wall.Remove(System.String.Format("Wall {0}-{1}-V", x, y - 1));  
 cell[x, y - 1].visited = true;  
  
 stack1.Push(cell[x, y - 1]);  
 y -= 1;  
 break;  
 case "right":  
 wall.Remove(System.String.Format("Wall {0}-{1}-V", x, y));  
 cell[x, y + 1].visited = true;  
  
 stack1.Push(cell[x, y + 1]);  
 y++;  
 break;  
 }  
 }  
 catch  
 {  
 stack1.Pop();  
 }  
 }  
 }   
 algo = new djk(0, 0, cell.GetLength(0)-1, cell.GetLength(1)-1,cell, wall);  
 }  
  
 private void Checkdir(int x, int y){ //check the direction available for a cell  
 try{if(cell[x-1,y].visited){  
 cell[x,y].dir.Remove("up");  
 }}catch{}  
 try{if(cell[x+1,y].visited){  
 cell[x,y].dir.Remove("down");  
 }}catch{}  
 try{if(cell[x,y-1].visited){  
 cell[x,y].dir.Remove("left");  
 }}catch{}  
 try{if(cell[x,y+1].visited){  
 cell[x,y].dir.Remove("right");  
 }}catch{}  
 }  
}

# SingleMazeGeneration.cs

using System.Collections;  
using System.Collections.Generic;  
using UnityEngine;  
using System.Linq;  
  
public class mazegeneration2 : MonoBehaviour  
{  
 Mazecell[,] cell;  
 Stack stack1 = new Stack(1000);  
 int startx,starty,endx,endy;  
   
 public mazegeneration2(Mazecell[,] cell1){ cell = cell1;   
 Run();   
 }  
 void Run(){  
 Debug.Log("runcount");  
 BuildMaze(0,0);  
 Notify();  
 }  
 void Notify(){  
   
 bool again = false;  
 foreach (var item in cell)  
 {  
 if (!item.visited) again = true;  
 }  
 if(again){  
 Run();  
 }  
 }  
 //it works but after reach dead-end it stop  
 int BuildMaze(int x, int y){   
 Mazecell currentcell = cell[x,y];  
 currentcell.visited = true;  
 checkdir(x,y);  
 if(currentcell.dir.Count()>0){  
 int element = Random.Range(0, currentcell.dir.Count());  
 string now = currentcell.dir[element];  
 currentcell.dir.RemoveAt(element);  
 switch(now){  
 case "up":  
 Destroy(GameObject.Find(System.String.Format("Wall {0}-{1}-H",x-1,y)));  
 return BuildMaze(x-1,y);  
 break;  
 case "down":  
 Destroy(GameObject.Find(System.String.Format("Wall {0}-{1}-H",x,y)));  
 return BuildMaze(x+1,y);  
 break;  
 case "left":  
 Destroy(GameObject.Find(System.String.Format("Wall {0}-{1}-V",x,y-1)));  
 return BuildMaze(x,y-1);  
 break;  
 case "right":  
 Destroy(GameObject.Find(System.String.Format("Wall {0}-{1}-V",x,y)));  
 return BuildMaze(x,y+1);  
 break;  
 default:  
 return 0;  
 }  
 }else{  
 return 0;  
 }  
 }  
 void checkdir(int x, int y){  
 try{if(cell[x-1,y].visited){  
 Debug.Log("remove up");  
 cell[x,y].dir.Remove("up");  
 }}catch{}  
 try{if(cell[x+1,y].visited){  
 cell[x,y].dir.Remove("down");  
 }}catch{}  
 try{if(cell[x,y-1].visited){  
 cell[x,y].dir.Remove("left");  
 }}catch{}  
 try{if(cell[x,y+1].visited){  
 cell[x,y].dir.Remove("right");  
 }}catch{}  
 }  
}

# WallCell.cs

using System.Collections;  
using System.Collections.Generic;  
using UnityEngine;  
  
public class WallCell : MonoBehaviour  
{  
}

MazeGame > Monster

# BigGreiver.cs

using System;  
using System.Collections;  
using System.Collections.Generic;  
using UnityEngine;  
  
public class BigGreiver : MonsterBehaviour  
{  
 public override void Update()  
 {  
 if (char1 == null)  
 {  
 try{char1 = FindObjectOfType<movement>().gameObject;}catch{ }  
 }  
 else  
 {  
 if(!isDead) Chase(char1.transform.position,2,15,1.53f,35f);  
 if(!isRunned) Dead();  
 }  
 }   
}

# blade.cs

using System;  
using System.Collections;  
using System.Collections.Generic;  
using UnityEngine;  
  
public class blade : MonoBehaviour  
{  
 [SerializeField] float rotateSpeed;  
 [SerializeField] float Scale;  
  
 // Start is called before the first frame update  
 void Start()  
 {  
 Vector3 or = transform.localScale;  
 transform.localScale = new Vector3(or.x,or.y,or.z\*Scale);  
 }  
  
 void Update()  
 {  
 transform.Rotate( new Vector3(0,rotateSpeed\*Time.deltaTime, 0));  
   
 }  
 private void OnCollisionEnter(Collision collision)  
 {  
 Singleton.HealthClass.changeHP(-10);  
 }  
}

# MonsterBaseClass.cs

ï»¿using System;  
using Unity.VisualScripting;  
using UnityEngine;  
  
public interface IMonsterConfig {  
 public float health { get; set; }  
 public Transform location { get; set; }  
 public void Look(Vector3 loc);  
 public void Chase(Vector3 loc, float MoveSpeed, float MoveDist, float AttackDist ,float LookDist);  
 public void Dead();  
}  
  
public abstract class MonsterBehaviour : MonoBehaviour, IMonsterConfig  
{  
   
 public float \_health;  
 private Transform \_location, \_monsterloc;  
 protected GameObject char1;  
 private Animator \_animator;  
 protected bool isDead = false;  
 protected bool isRunned = false;  
 private MazeManager \_ins;  
   
 public float health { get => \_health;  
 set => \_health = value; }  
 public Transform location { get => \_location;  
 set => \_location = value; }  
  
 public void Awake()  
 {  
 \_monsterloc = gameObject.transform;  
 \_health = 100f;  
 }  
 public void Start()  
 {  
 \_animator = GetComponentInChildren<Animator>();  
 \_ins = FindObjectOfType<MazeManager>();  
 // Debug.Log(char1);  
 }  
  
 public virtual void Update()  
 {  
 if (char1 == null)  
 {  
 char1 = FindObjectOfType<movement>().gameObject;  
 }  
 else  
 {  
 if(!isDead) Chase(char1.transform.position);  
 if(!isRunned) Dead();  
 }  
 }  
  
 public void Look(Vector3 loc)  
 {  
 float deltax = transform.position.x - loc.x;  
 float deltay = transform.position.z - loc.z;  
 float x = Mathf.Rad2Deg\* (Mathf.Atan(deltay/deltax));  
 Vector3 test;  
 if (deltax < 0 && deltay > 0) test = new Vector3(0, -(270+x), 0);  
 else if (deltax <0 &&deltay <0) test = new Vector3(0, 90-x, 0);  
 else if (deltax > 0 && deltay < 0) test = new Vector3(0, -90-x, 0);  
 else test = new Vector3(0, -90-x, 0);  
 transform.localEulerAngles = test;  
 }  
   
 public void Chase(Vector3 loc,float MoveSpeed = 2f, float MoveDist = 10f, float AttackDist = 1.3f,float LookDist = 25f)  
 {  
 // Vector3 trans = transform.TransformDirection(loc);  
 // transform.LookAt(char1.transform.position);  
 float dist = Vector3.Distance(transform.position,char1.transform.position);  
 if(dist <= LookDist)  
 {  
 Look(loc);  
 \_ins.MonsterPrompt(false);  
 if(dist >= AttackDist && dist <= MoveDist){  
 \_ins.MonsterPrompt(true);  
 \_animator.SetBool("walk", true);  
 transform.position += transform.forward \* (MoveSpeed \* Time.deltaTime);  
 }else if (dist <= AttackDist)  
 {  
 \_animator.SetBool("walk", false);  
 \_animator.SetTrigger("attack");  
 Singleton.HealthClass.changeHP(-1);  
 }  
 else  
 {  
 \_animator.SetBool("walk", false);  
 }  
 }  
 }  
  
 public void OnCollisionEnter(Collision collision)  
 {  
 // Debug.Log(collision.collider.tag.ToString());  
 if (collision.collider.tag == "Player")  
 {  
 \_animator.SetTrigger("attack");  
 }  
 else if (collision.collider.tag == "Trap")  
 {  
 GameObject headshot = Instantiate(Resources.Load<GameObject>("Blood\_Headshot Variant"), \_monsterloc);  
 Destroy(headshot, 3f);  
 \_animator.SetTrigger("hurt");  
 \_health -= 60f;  
 }  
 else if (collision.collider.tag == "Axe")  
 {  
 GameObject headshot = Instantiate(Resources.Load<GameObject>("Blood\_Headshot Variant"), \_monsterloc);  
 Destroy(headshot, 3f);  
 \_health -= 110f;  
 }  
 else  
 {  
// Debug.Log(collision.collider.name);  
 }  
  
 }  
 public void Dead()  
 {  
   
 if (\_health < 0)  
 {  
 \_animator.SetBool("walk", false);  
 \_animator.SetTrigger("dead");  
 isDead = true;  
 isRunned = true;  
 FindObjectOfType<MazeManager>().MonsterKilled++;  
 FindObjectOfType<MazeManager>().UpdateScore();  
 Destroy(gameObject.GetComponentInChildren<SpriteRenderer>().gameObject);  
 }  
 }  
}

# SilverFish.cs

using System.Collections;  
using System.Collections.Generic;  
using UnityEngine;  
  
public class SilverFish : MonsterBehaviour  
{  
  
 // Update is called once per frame  
}

# SmallGreiver.cs

using System.Collections;  
using System.Collections.Generic;  
using UnityEngine;  
  
public class SmallGreiver : MonsterBehaviour  
{  
   
}

MazeGame > Movement

# movement.cs

using System.Collections;  
using System.Collections.Generic;  
using UnityEngine;  
  
public class movement : gravity  
{  
 [SerializeField] private Joystick joystick;  
 [SerializeField] private Vector2 move;  
 [SerializeField] private float \_walkspeed;  
 //crouch  
 [SerializeField] private bool \_isCrouching;  
 [SerializeField] private float \_crouchwalk;  
 [SerializeField] private float \_crouchheight;  
 //jumping  
 [SerializeField] private float \_jumpheight;  
   
 public float walkspeed {  
 get => \_walkspeed;  
 set => \_walkspeed = value;  
 }  
 public bool isCrouching {  
 get => \_isCrouching;  
 set => \_isCrouching = value;  
 }  
 public float crouchwalk {  
 get => \_crouchwalk;  
 set => \_crouchwalk = value;  
 }  
 public float crouchheight {  
 get => \_crouchheight;  
 set => \_crouchheight = value;  
 }  
 public bool isUpJumping {  
 get => \_isUpJumping;  
 set => \_isUpJumping = value;  
 }  
 public float jumpheight {  
 get => \_jumpheight;  
 set => \_jumpheight = value;  
 }  
   
 void Start(){  
 joystick = FindObjectOfType<VariableJoystick>();  
 }  
 void Update() {  
 base.Update();  
 touchmovement();  
 }  
 private void touchmovement(){  
 float yacc = gravitySetting();  
 if(!\_isUpJumping){  
 float updatespeed;  
 if(\_isCrouching||!\_grounded) updatespeed = \_walkspeed \* \_crouchwalk;else updatespeed = \_walkspeed;  
 move = new Vector2(joystick.Horizontal \* updatespeed,joystick.Vertical \* updatespeed);   
 }else move = Vector2.zero;  
 Vector3 trans = transform.TransformDirection(new Vector3(move.x, 0 , move.y));  
 transform.position = new Vector3(trans.x+ original.x, original.y+ yacc, trans.z+ original.z);  
 }  
 public void crouch()  
 {  
 CapsuleCollider collider = this.gameObject.GetComponent<CapsuleCollider>();  
 if(\_grounded){  
 if(\_isCrouching) {  
 \_isCrouching = false;  
 transform.position = new Vector3(original.x, original.y + \_crouchheight, original.z);  
   
 collider.center = new Vector3(0, 0.91f, 0);  
 collider.height = 1.82f;  
 } else {  
 \_isCrouching = true;  
 transform.position = new Vector3(original.x, original.y - \_crouchheight, original.z);  
 collider.center = new Vector3(0, 1.26f, 0);  
 collider.height = 1.37f;  
 }   
 }  
 }  
 public void Jumpbutton(){  
 if(\_grounded){  
 \_isUpJumping = true;  
 gravityacceleration = \_jumpheight;  
 }  
 }  
}

# newTerrainGravity.cs

ï»¿using UnityEngine;  
using UnityEngine.TerrainUtils;  
  
public class TerrainFollower : MonoBehaviour  
{  
 public Terrain terrain;  
  
 private Vector3 terrainSize;  
 private Vector3 terrainPos;  
  
 void Start()  
 {  
 terrainSize = terrain.terrainData.size;  
 terrainPos = terrain.transform.position;  
 }  
  
 void Update()  
 {  
 float x = (transform.position.x - terrainPos.x) / terrainSize.x;  
 float z = (transform.position.z - terrainPos.z) / terrainSize.z;  
  
 float y = terrain.SampleHeight(transform.position);  
 // float slope = terrain.GetSteepness(x, z);  
 float slope = 6;  
 if (slope >= 30)  
 {  
 y = terrainPos.y + terrainSize.y;  
 }  
  
 transform.position = new Vector3(transform.position.x, y, transform.position.z);  
 }  
}  
//  
// using UnityEngine;  
//  
// public class CharacterController : MonoBehaviour  
// {  
// public float speed = 10.0f;  
// public float jumpForce = 10.0f;  
// public float raycastDistance = 1.0f;  
// public LayerMask groundLayer;  
//  
// private Rigidbody rb;  
// private Vector3 moveDirection;  
//  
// void Start()  
// {  
// rb = GetComponent<Rigidbody>();  
// }  
//  
// void Update()  
// {  
// moveDirection = new Vector3(Input.GetAxis("Horizontal"), 0, Input.GetAxis("Vertical"));  
// moveDirection = transform.TransformDirection(moveDirection);  
// moveDirection \*= speed \* Time.deltaTime;  
//  
// RaycastHit hit;  
// if (Physics.Raycast(transform.position, -Vector3.up, out hit, raycastDistance, groundLayer))  
// {  
// float y = hit.point.y;  
// transform.position = new Vector3(transform.position.x, y, transform.position.z);  
// }  
//  
// if (Input.GetKeyDown(KeyCode.Space) && IsGrounded())  
// {  
// rb.AddForce(Vector3.up \* jumpForce, ForceMode.Impulse);  
// }  
//  
// rb.MovePosition(transform.position + moveDirection);  
// }  
//  
// bool IsGrounded()  
// {  
// return Physics.Raycast(transform.position, -Vector3.up, raycastDistance, groundLayer);  
// }  
// }

# PlayerSetup.cs

using System.Collections;  
using System.Collections.Generic;  
using UnityEngine;  
  
public class PlayerSetup : MonoBehaviour  
{  
 float timer1;  
 // Start is called before the first frame update  
 void Start()  
 {  
   
 }  
  
 // Update is called once per frame  
 void Update()  
 {  
 timer1 += Time.deltaTime;  
 timer1 %= 10;  
 if(timer1 >5) {  
 Vector3 scale1 = new Vector3(5,5,5);  
 transform.localScale = scale1;}   
 else {  
 Vector3 scale1 = new Vector3(1,1,1);  
 transform.localScale = scale1;}  
 }  
}  
interface PlayerAction{  
 void PlayerMove(){}  
 void PlayerLook(){}  
}  
class playerset :PlayerAction{  
 public void PlayerMove(){  
  
 }  
}

# switchcam.cs

using System.Collections;  
using System.Collections.Generic;  
using UnityEngine;  
  
public class switchcam : MonoBehaviour  
{  
 // Start is called before the first frame update  
 void Start()  
 {  
   
 }  
  
 // Update is called once per frame  
 void Update()  
 {  
   
 }  
  
 private Camera[] \_cams;  
   
 public void ChangeCamera(){  
 turnaround turnScript = GameObject.FindObjectOfType<turnaround>();  
 if(\_cams[0].enabled){  
 \_cams[0].enabled = false;  
 \_cams[1].enabled = true;  
 }else{  
 \_cams[0].enabled = true;  
 \_cams[1].enabled = false;  
 }  
 turnScript.cam = cams[0];  
 }  
  
 public Camera[] cams  
 {  
 get => \_cams;  
 set => \_cams = value;  
 }  
}

# toucharea.cs

using System.Collections;  
using System.Collections.Generic;  
using UnityEngine;  
  
public class toucharea : MonoBehaviour  
{  
 // Start is called before the first frame update  
 GameObject toucharea1;  
   
 void Start()  
 {  
 toucharea1 = this.gameObject;  
 Debug.Log(toucharea1);  
 }  
 void Update(){  
   
 }  
 private void Raycast(){  
 Touch touch;  
 // if(Input.touchCount > 0){  
 // touch = Input.GetTouch(touchindex);  
 // Debug.DrawRay(Vector3.zero,touch.position);  
  
 // Ray ray = Camera.main.ScreenPointToRay(touch.position);   
 // RaycastHit hit;   
 // if (Physics.Raycast(ray, out hit)) {   
 // if (hit.transform.name == "Toucharea") { }   
 // }   
 // }  
 }  
 private int GetTouch(){  
 int lastIndex=0, lastFingerIndex =0;  
 for(int i = 0; i < Input.touches.Length; i++) {   
 if(Input.touches[i].phase == TouchPhase.Began)  
 {  
 lastFingerIndex = Input.touches[i].fingerId;  
 // if(Input.touches[i].fingerid == lastFingerIndex)  
 // {  
 // lastIndex = i;  
 // }  
 }  
   
 }  
 return lastIndex;  
 }  
 float Vertical {get; set;}  
 float Horizontal {get;set;}  
}

# turnaround.cs

using System.Collections;  
using System.Collections.Generic;  
using Unity.VisualScripting;  
using UnityEngine;  
using UnityEngine.EventSystems;  
  
public class turnaround : MonoBehaviour  
{   
 [SerializeField] private Camera \_cam;  
 [SerializeField] private Joystick dyn\_joy;  
 [SerializeField] private float \_xspeed, \_yspeed;  
  
 public Camera cam  
 {  
 get => \_cam;  
 set => \_cam = value;  
 }  
 public float xspeed  
 {  
 get => \_xspeed;  
 set => \_xspeed = value;  
 }  
 public float yspeed  
 {  
 get => \_yspeed;  
 set => \_yspeed = value;  
 }  
  
 void Start()  
 {  
 dyn\_joy = FindObjectOfType<DynamicJoystick>();  
 }  
 void Update(){  
 float y = dyn\_joy.Horizontal;  
 float x = -dyn\_joy.Vertical;  
 transform.Rotate(y \* \_yspeed \* Vector3.up );  
  
 //\_cam rotate   
 Vector3 camrotate = \_cam.transform.localEulerAngles;  
 // Debug.Log(camrotate);  
 camrotate.x += x \* \_xspeed;  
 //290-360 and 0 to 20   
 camrotate.x = Mathf.Clamp(camrotate.x, 0f , 12f);  
 camrotate.z = 0;  
 // if(camrotate.x>-19 && camrotate.x<12){  
 \_cam.transform.localEulerAngles = camrotate;  
 // }   
 }  
}

MenuScreen

# continuePanel.cs

using System;  
using System.Collections;  
using System.Collections.Generic;  
using UnityEngine;  
using UnityEngine.UI;  
  
public class continuePanel : MonoBehaviour  
{  
 private Button continueButton;  
  
 private void Start()  
 {  
 continueButton = gameObject.GetComponent<Button>();  
 continueButton.onClick.AddListener(ClosePanel);  
 }  
 private void ClosePanel()  
 {  
 Destroy(this.gameObject);  
 }  
}

# fadeOutImage.cs

ï»¿using UnityEngine;  
  
public class fadeoutimage : MonoBehaviour  
{  
  
 private void Update()  
 {  
 if (Input.touchCount > 0 && Input.GetTouch(0).phase == TouchPhase.Began)  
 Destroy(this.gameObject);  
 }  
}

# LeaderBoarddata.cs

using System.Collections.Generic;  
using UnityEngine;  
using System;  
using System.Threading.Tasks;  
using Newtonsoft.Json;  
using Unity.VisualScripting;  
using UnityEngine.UI;  
  
  
public class LeaderBoarddata : MenuPanels  
{  
 private Transform slot;  
  
 public void external(bool allStat)  
 {  
 if(allStat)Application.OpenURL("");  
 else Application.OpenURL("");  
 }  
 public override void Start()  
 {  
 slot = GameObject.Find("LeaderboardDataSlot").transform;  
 Button[] buttons = GameObject.Find("Title").GetComponentsInChildren<Button>();  
 for (int i = 0; i < buttons.Length; i++)  
 {  
 int dummyi = i;  
 buttons[i].onClick.AddListener(delegate { Sorting(dummyi); });  
 // Debug.Log(buttons[i]);  
 }  
 base.Start();  
 }  
  
 public void Sorting(int index)  
 {  
 MergeSortClass merge = new MergeSortClass();  
 printData(merge.MergeSort(GetData().ToArray(), index));  
 }  
 private List<Leaderboard> ParseJSON(string value)  
 {  
 List<Leaderboard> alist = new List<Leaderboard>();  
 List<string> stringList = new List<string>();  
 value = value.Replace(" ", "");  
 int startIndex = 0, index = 0;  
 while (index != -1)  
 {  
 index = value.IndexOf(';', startIndex);  
 if (index == -1)  
 {  
 // stringList.Add(value[startIndex..]);  
 break;  
 }  
  
 stringList.Add(value.Substring(startIndex, index - startIndex));  
 startIndex = index + 1;  
 }  
  
 stringList.ToArray();  
 // Debug.Log(JsonConvert.SerializeObject(stringList));  
 List<string[]> jsonArray = new List<string[]>();  
 for (int i = 0; i < stringList.Count; i++)  
 {  
 string[] temp = stringList[i].Split(',');  
 jsonArray.Add(temp);  
 alist.Add(new Leaderboard());  
 alist[i].Username = jsonArray[i][0];  
 alist[i].Level = int.Parse(jsonArray[i][1]);  
 alist[i].Score = int.Parse(jsonArray[i][2]);  
 alist[i].Since = int.Parse(jsonArray[i][3]);  
 }  
  
 return alist;  
 }  
  
 public void WebLeader(int index)  
 {  
 string[] website = { "pivot", "cross" };  
 Application.OpenURL($"https://maze.just4fun.tk/{website[index]}");  
 }  
  
  
 public List<Leaderboard> GetData()  
 {  
 foreach (var VARIABLE in FindObjectsOfType<leaderslot>())  
 {  
 Destroy(VARIABLE.gameObject);  
 }  
 return ParseJSON(Task.Run(() =>  
 Singleton.CsharpAPI.GetData(true, $"SELECT pl.username,p.levelID, p.score, p.time FROM Progress p INNER JOIN Player pl ON p.playerID = pl.playerID", "")).Result);  
 }  
  
 public void printData(Leaderboard[] result)  
 {  
 for (int i = 0; i < result.Length; i++)  
 {  
 var local = Instantiate(Resources.Load("UI/LeaderButton"), slot);  
 var localscript = local.GetComponent<leaderslot>();  
 localscript.User.text = result[i].Username;  
 localscript.Level.text = result[i].Level.ToString();  
 localscript.Score.text = result[i].Score.ToString();  
 localscript.Since.text = result[i].Since.ToString();  
 }  
 }  
}  
[Serializable] public class Leaderboard{  
 public string Username { get; set; }  
 public int Level { get; set; }  
 public int Score { get; set; }  
 public int Since { get; set; }  
 }

# leaderslot.cs

using System.Collections;  
using System.Collections.Generic;  
using UnityEngine;  
using TMPro;  
public class leaderslot : MonoBehaviour  
{  
 [SerializeField] private TMP\_Text user, level, score, since;  
  
 public TMP\_Text User  
 {  
 get => user;  
 set => user = value;  
 }  
  
 public TMP\_Text Level  
 {  
 get => level;  
 set => level = value;  
 }  
  
 public TMP\_Text Score  
 {  
 get => score;  
 set => score = value;  
 }  
  
 public TMP\_Text Since  
 {  
 get => since;  
 set => since = value;  
 }  
}

# loginState.cs

using System;  
using System.IO;  
using System.Threading.Tasks;  
using Newtonsoft.Json;  
using UnityEngine;  
using UnityEngine.UI;  
using TMPro;  
public class loginState : MenuPanels  
{  
 [Header("welcomeBack")]  
 [SerializeField] private Button logbut;  
 [SerializeField] private TMP\_InputField usernameField, PasswordField;  
 [SerializeField] private TMP\_Text info;  
 [SerializeField] private GameObject welcomebackPanel;  
  
 [Header("login")] [SerializeField]private GameObject loginPanel;  
 [SerializeField] private Button notMeButton, playButton;  
 [SerializeField]private TMP\_Text getusername;  
  
 private string user;  
 private int ID;  
 string path => Application.persistentDataPath + "/login.bin";  
 public override void Start()  
 { }  
  
 public void OnEnable()  
 {  
   
 Singleton.Instance.init();  
 if (File.Exists(path))  
 {  
 var stream = File.OpenRead(path);  
 BinaryReader br = new BinaryReader(stream);  
 try{ID = br.ReadInt32();  
 user = br.ReadString();}catch{}  
 br.Close();  
 try{if (user.Length != 0)  
 {  
 welcomeBack();  
 }}catch{welcomebackPanel.SetActive(false);  
 loginPanel.SetActive(true);  
 infoHandler(false);}  
 }  
 else  
 {  
 welcomebackPanel.SetActive(false);  
 loginPanel.SetActive(true);  
 infoHandler(false);  
 }  
 }  
 void welcomeBack()  
 {  
 welcomebackPanel.SetActive(true);  
 Singleton.Instance.username = this.user;  
 Singleton.Instance.playerID = this.ID;  
 getusername.text = Singleton.Instance.username;  
 loginPanel.SetActive(false);  
 }  
 public void Play()  
 {  
 welcomebackPanel.SetActive(false);  
 loginPanel.SetActive(false);  
 this.ShowHide(false);  
 }  
 public void NotYou()  
 {  
 File.Delete(path);  
 Singleton.Instance.username = "";  
 Singleton.Instance.playerID = 0;  
 loginPanel.SetActive(true);  
 welcomebackPanel.SetActive(false);  
 }  
 public void Register()  
 {  
 Application.OpenURL("https://maze.just4fun.tk/register");  
 }  
 public void Login()  
 {  
 FileStream stream = new FileStream(path, FileMode.Create);  
 BinaryWriter writer = new BinaryWriter(stream);  
 if (PasswordField.text.Length < 8 || usernameField.text.Length == 0)  
 {  
 infoHandler(true, "password length must be longer than 8 characters");  
 return;  
 }  
 var result = Task.Run(() => Singleton.CsharpAPI.GetData(true,   
 "SELECT hash, salt, playerID, username FROM Player WHERE email = ?",  
 usernameField.text)).Result.Split(",");  
 try  
 {  
 if(Math.Floor(hexadeciamToDenary(result[0])/100000000f)== Math.Floor(hashFunction(PasswordField.text, int.Parse(result[1]))/100000000f))  
 {var playerID = int.Parse(result[2]);  
 var username = result[3];  
 writer.Write(playerID);  
 writer.Write(username);  
 Singleton.Instance.username = username;  
 Singleton.Instance.playerID = playerID;  
 infoHandler(true,"Logged in, please close this window");  
 Singleton.Instance.Loggedin = true;  
 } else  
 {  
 infoHandler(true,"Wrong password");  
 } }  
 catch  
 {  
 infoHandler(true, JsonConvert.SerializeObject(result));  
 }  
 writer.Close(); stream.Close();  
 }  
  
 void infoHandler(bool enable, string text ="")  
 {  
 info.text = text;  
 info.enabled = enable;  
 }  
 float hashFunction(string password, float salt)  
 {  
 float hash = 0, count = 10;  
 for (int i = 0; i < password.Length; i++) {  
 char c = password[i];  
 hash += (float)(c \* Math.Pow(10, count - i) + salt);  
 }  
 return hash;  
 }  
  
 // public float DecToHex(float hash)  
 // {  
 // return (int)(hash / 16) + ((hash / 16) - (int)(hash / 16)) \* 16;  
 // }  
 public float hexadeciamToDenary(string hexa)  
 {  
 float tempnum = 0, count = 0;  
 for (int i = 0; i < hexa.Length; i++)  
 {  
 switch (hexa[i])  
 {  
 case '0':case '1':case '2':case '3':case '4':  
 case '5':case '6':case '7':case '8':case '9':  
 tempnum = hexa[i]-48;  
 break;  
 case 'a':  
 tempnum = 10;  
 break;  
 case 'b':  
 tempnum = 11;  
 break;  
 case 'c':  
 tempnum = 12;  
 break;  
 case 'd':  
 tempnum = 13;  
 break;  
 case 'e':  
 tempnum = 14;  
 break;  
 case 'f':  
 tempnum = 15;  
 break;  
 }  
 count +=(tempnum \* Mathf.Pow(16, hexa.Length-i-1));  
 }return count;  
 }  
   
}

# MainMenuManager.cs

ï»¿using System;  
using System.Collections.Generic;  
using System.IO;  
using System.Threading.Tasks;  
using MenuScreen.panels;  
using TMPro;  
using UnityEngine;  
using UnityEngine.SceneManagement;  
using UnityEngine.UI;  
using Random = UnityEngine.Random;  
  
namespace MenuScreen  
{  
 public class MainMenuManager : MonoBehaviour  
 {  
 private bool \_manSfXplay;  
 private Animator \_manAnimator;  
 [SerializeField] private Image fade;  
  
 [SerializeField]private GameObject \_continuePanel;  
   
 [Header("Audio")]   
 [SerializeField] private List<AudioClip> menubuttonSfx;  
 [SerializeField] private AudioClip[] manSfx;  
 [SerializeField] private AudioClip quitButtonSfx;  
  
 [Header("Button")] [SerializeField] private GameObject menuButtons;  
 private Button[] \_menuSelectorButton;  
 private SettingManager \_settingManager;  
 private QuitMenuPanel \_quitMenuPanel;  
 private LeaderBoarddata \_leaderboardPanel;  
 private SlotSelectionPanel \_slotSelectionPanel;  
 private GameObject[] panels;  
   
 private loginState \_loginState;  
 private int continueFrom;  
 [SerializeField]private TMP\_Text loginstate;  
 [SerializeField] private Button continueButt;  
  
 public Button ContinueButt  
 {  
 get => continueButt;  
 set => continueButt = value;  
 }  
  
 void Awake()  
 {  
 menuButtons = GameObject.Find("MenuSelector").gameObject;  
 fade = GameObject.Find("FadeOutImage").GetComponent<Image>();  
 Singleton.Fade.FadeOut(fade);  
 \_manAnimator = GameObject.Find("casual\_male\_unshaded").GetComponentInChildren<Animator>();  
  
 \_settingManager = FindObjectOfType<SettingManager>();  
 \_quitMenuPanel = FindObjectOfType<QuitMenuPanel>();  
 \_leaderboardPanel = FindObjectOfType<LeaderBoarddata>();  
 \_slotSelectionPanel = FindObjectOfType<SlotSelectionPanel>();  
 \_loginState = FindObjectOfType<loginState>();  
   
 panels = new[]  
 {  
 \_loginState.gameObject, \_slotSelectionPanel.gameObject, \_leaderboardPanel.gameObject,  
 \_settingManager.gameObject, \_quitMenuPanel.gameObject  
 };  
   
 \_menuSelectorButton = menuButtons.GetComponentsInChildren<Button>();  
 foreach (Button button in \_menuSelectorButton){  
 button.onClick.AddListener(PlaySound);  
 }  
 for (int i = 0; i < \_menuSelectorButton.Length; i++)  
 {  
 int dummyi = i;  
 \_menuSelectorButton[i].onClick.AddListener(() => menuSelectionButtonAction(dummyi));  
 }  
  
 string path = Application.persistentDataPath + "/continue.txt";  
 if (File.Exists(path))  
 {  
 StreamReader writer = new StreamReader(path);  
 string text = Task.Run(() =>writer.ReadToEndAsync()).Result;  
 if (text.Length > 0)  
 {  
 continueButt.interactable = true;  
 continueFrom = int.Parse(text);  
 }  
 }  
  
 if (!Singleton.Instance.ContinuePrompt)  
 {  
 \_continuePanel.SetActive(false);  
 }  
  
 //   
 Singleton.Instance.ShowDetail("Login","Remember login to Save progress \n Inside: Setting > Login");  
   
 }  
  
 public void continuePrompt()  
 {  
 Singleton.Instance.ContinuePrompt = false;  
 Destroy(GameObject.Find("ContinuePanel").gameObject);  
 Singleton.Instance.ContinuePrompt = false;  
 }  
 private void menuSelectionButtonAction(int selection)  
 {  
 panels[selection].SetActive(true);  
 if (selection == 2)  
 {  
 Singleton.Instance.ShowDetail("Data","click on the column header to show data \n All data and stat in the left panel");  
  
 }  
 }  
   
 void Update(){  
 if(\_manAnimator.IsInTransition(0) && \_manSfXplay){  
 Singleton.AudioPlayer.PlaySfx(manSfx[Random.Range(0,manSfx.Length)]);  
 \_manSfXplay = false;  
 }  
  
 if (Singleton.Instance.username.Length > 0)  
 loginstate.text = Singleton.Instance.username;  
 else loginstate.text = "Please Login";  
 }  
 void PlaySound(){  
 int num = Random.Range(0,menubuttonSfx.Count);  
 Debug.Log(menubuttonSfx[num]);  
 Singleton.AudioPlayer.PlaySfx(menubuttonSfx[num]);  
 }  
 //public  
 public void Move(){  
 \_manAnimator.SetTrigger("exit");  
 \_manSfXplay = true;  
 }  
 public void ShowQuit(bool show){  
 // \_quitpanel.ShowHide(show);  
 if(show){  
 Singleton.AudioPlayer.PlaySfx(quitButtonSfx);  
 }  
 }  
  
 public void ContinueButton() => \_slotSelectionPanel.SavefileSelector(continueFrom);  
 }  
}

# onLoadMainMenu.cs

using System.Collections;  
using System.Collections.Generic;  
using UnityEngine;  
  
public class onLoadMainMenu : MonoBehaviour  
{  
 void Awake()  
 {  
 GameObject singleton\_ins = Singleton.Instance.gameObject;  
 Singleton.Instance.init();  
 }  
 void Update()  
 {  
   
 }  
}

MenuScreen > Panels

# menuBaseClass.cs

using UnityEngine;  
  
public class MenuPanels: MonoBehaviour  
{  
 [HideInInspector]  
 public GameObject \_menu;  
  
 public void Awake()  
 {  
 \_menu = gameObject;  
 }  
  
 public virtual void Start()  
 {  
 ShowHide(false);  
 }  
  
 public void ShowHide(bool show) {  
 \_menu.SetActive(show);  
 }  
 public virtual void Function() {  
   
 }  
}

# QuitMenuPanel.cs

using System.Collections;  
using System.Collections.Generic;  
using UnityEngine;  
using UnityEngine.UI;  
public class QuitMenuPanel : MenuPanels  
{  
 public void appQuit(){  
 Application.Quit();  
 Debug.Log("Quited");  
 }  
}

# SlotSelectionPanel.cs

ï»¿using System;  
using UnityEngine;  
using UnityEngine.UI;  
using System.Collections;  
using System.Collections.Generic;  
using UnityEngine.SceneManagement;  
using System.IO;  
using System.Threading.Tasks;  
  
namespace MenuScreen.panels  
{  
 public class SlotSelectionPanel : MenuPanels  
 {  
 private string filePath;  
 public override void Start()  
 {  
 filePath = Application.persistentDataPath;  
 Button[] buttonSlot = gameObject.transform.Find("Slot").GetComponentsInChildren<Button>();  
 for (int i = 0; i < buttonSlot.Length; i++)  
 {  
 int dummyi = i;  
 if(i%2 == 0) {dummyi /=2;  
 buttonSlot[i].onClick.AddListener(delegate{SavefileSelector(dummyi);});}  
 else {dummyi = (dummyi-1)/2;  
 buttonSlot[i].onClick.AddListener(delegate{popUpDetail(dummyi); });}  
 }  
 base.Start();  
 }  
  
 public void SavefileSelector(int index = 0)  
 {  
   
 //file check slot1 exist  
 //if not, image = not exist (create new [first time])  
 //if exist  
 Singleton.Instance.LoadNumber = index;  
 string path = filePath + "/continue.txt";  
 Debug.Log(File.Exists(path));  
 // StreamWriter writer = new StreamWriter(path, false);  
 // Write some text to the file  
 File.WriteAllText(path,index.ToString());  
 // writer.Close();  
 Debug.Log(index.ToString());  
 if (File.Exists($"{filePath}/Save{index}.sqlite3"))  
 {  
 Debug.Log("!ft");  
 Singleton.Instance.FirstTime = false;  
 Singleton.LoadScreenclass.LoadScreen(true, false, SceneManager.GetActiveScene().buildIndex + 2, true, true,  
 true);  
 }  
 else{ Debug.Log("ft");  
 Singleton.Instance.FirstTime = true;  
 // File.Create($"{filePath}/Save{index}.maze");  
 Task.Run(() => Singleton.Localdb.Query(@"CREATE TABLE GAMEDATA ('characterSelection' INTEGER, 'playerX' FLOAT, 'playerY' FLOAT , 'playerZ' FLOAT );",  
 $"Save{index}.sqlite3")  
 ).Wait();  
 Singleton.LoadScreenclass.LoadScreen(true,false,SceneManager.GetActiveScene().buildIndex+1,true,true,true);  
 }  
  
 }  
  
 public void DeleteSlot(int index)  
 {  
 Task.Run(() =>  
 {  
 Singleton.Localdb.Query($"DROP TABLE IF EXISTS GAMEDATA",  
 $"Save{index}.sqlite3");  
 }).Wait();  
 try{File.Delete($"{Application.persistentDataPath}/Save{index}.sqlite3");}catch {}  
 try{File.Delete(Application.persistentDataPath + "/continue.txt");}catch {}  
 FindObjectOfType<MainMenuManager>().ContinueButt.interactable = false;  
   
 }  
 private void popUpDetail(int index)  
 {  
 Singleton.Instance.ShowDetail($"Game Save #{index}",  
 $"Last Play: {File.GetLastWriteTime(Application.persistentDataPath + $"/Save{index}.sqlite3")}\n",  
 Resources.Load<Sprite>("Sprite/Star"));  
 }  
 }  
}

MazeScreen > Setting

# AudioSettingPanel.cs

ï»¿using UnityEngine;  
  
namespace MenuScreen.panels  
{  
 public class AudioSettingPanel : MenuPanels  
 {  
 void Start()  
 {  
 \_menu = gameObject;  
 ShowHide(false);  
 }  
 }  
}

# ControlSettingPanel.cs

ï»¿using UnityEngine;  
  
namespace MenuScreen.panels  
{  
 public class ControlSettingPanel : MenuPanels  
 {  
   
 }  
}

# GraphicsSettingPanel.cs

ï»¿using UnityEngine;  
  
namespace MenuScreen.panels  
{  
 public class GraphicsSettingPanel : MenuPanels  
 {  
   
 }  
}

# Setting.cs

using System;  
using System.IO;  
using System.Threading.Tasks;  
using Newtonsoft.Json;  
using UnityEngine;  
using UnityEngine.UI;  
  
public class Setting : MonoBehaviour  
  
{  
 private string dbname,filePath;   
 [SerializeField] private Slider xSenS, ySenS,MusicVolS, sfxVolS,cameraFOVS, minimapSizeS, joystickSizeS,graphicQualityS,brightnessS;  
 void Start()  
 {  
 dbname = "setting.maze";  
 // create default value  
 filePath = Application.persistentDataPath + "/"+ dbname;  
 if (File.Exists(filePath)) LoadSetting();  
 else Setup();  
 }  
  
 public void LoadSetting()  
 {  
 var result = Task.Run(() => Singleton.Localdb.Query(@"SELECT \* FROM 'Setting';", "setting.maze")).Result  
 .Split(';');  
 // Debug.Log(JsonConvert.SerializeObject(result));  
 xSenS.value= float.Parse(result[0]);  
 ySenS.value= float.Parse(result[1]);  
 MusicVolS.value = float.Parse(result[2]);  
 sfxVolS.value = float.Parse(result[3]);  
   
 cameraFOVS.value= float.Parse(result[4]);  
 minimapSizeS.value= float.Parse(result[5]);  
 joystickSizeS.value= float.Parse(result[6]);  
 brightnessS.value= float.Parse(result[7]);  
  
 SaveSetting();  
 }  
 public void SaveSetting(){  
 UpdateSetting(xSenS.value, ySenS.value,MusicVolS.value, sfxVolS.value,cameraFOVS.value, minimapSizeS.value,joystickSizeS.value,brightnessS.value);  
   
 // Singleton.Instance.init();  
 try{Singleton.AudioPlayer.Music.volume = MusicVolS.value/100f;  
 Singleton.AudioPlayer.Sfx.volume = sfxVolS.value/100f;}catch{}  
 try{FindObjectOfType<turnaround>().xspeed = xSenS.value/100f\* 3f +2f;}catch(Exception e){}  
  
 try { FindObjectOfType<turnaround>().yspeed = ySenS.value / 100f \* 3f +2f; }catch(Exception e){}  
  
 try { FindObjectOfType<VariableJoystick>().gameObject.transform.localScale =  
 (Vector3.one \* 1.6f \* joystickSizeS.value / 100); }catch(Exception e){}  
  
 try { FindObjectOfType<minimap>().MinimapCam.orthographicSize = minimapSizeS.value / 100f \* 20f + 5f; }catch { }  
  
 try{GameObject.Find("TPP").GetComponent<Camera>().fieldOfView = cameraFOVS.value /100f\* 30f + 50f;}catch{ }  
 try{GameObject.Find("FPP").GetComponent<Camera>().fieldOfView = cameraFOVS.value /100f\* 30f + 50f;}catch{ }  
 Screen.brightness = brightnessS.value/100f;  
 }//\*40f + 60f;  
 public void Setup(){  
 Task.Run( () => Singleton.Localdb.Query(@"CREATE TABLE if not exists 'Setting' (  
 'sfxVol' INTEGER, 'MusicVol' INTEGER,  
 'cameraFOV' INTEGER, 'xSen' INTEGER,  
 'ySen' INTEGER, 'joystickSize' INTEGER,  
 'graphicQuality' INTEGER, 'brightness' INTEGER,  
 'minimapSize' INTEGER  
 );","setting.maze")).Wait();  
 Task.Run( () => Singleton.Localdb.Query("INSERT INTO Setting(xSen) VALUES (0);","setting.maze")).Wait();  
 UpdateSetting();  
 }  
  
 private static void UpdateSetting(float xSen = 50, float ySen = 50, float MusicVol = 50, float sfxVol = 50, float cameraFOV = 50, float minimapSize = 50, float joystickSize = 50, float brightness = 50, float graphicQuality = 50)  
 {  
 // System.Threading.Thread.Sleep(1000);  
 Task.Run(() => Singleton.Localdb.Query($@"UPDATE Setting   
SET sfxVol = {sfxVol},  
 MusicVol = {MusicVol},  
 cameraFOV = {cameraFOV},  
 xSen = {xSen},  
 ySen = {ySen},  
 joystickSize = {joystickSize},  
 graphicQuality = {graphicQuality},  
 brightness = {brightness},  
 minimapSize = {minimapSize};","setting.maze")).Wait();  
 }  
}

# SettingManager.cs

using System;  
using MenuScreen.panels;  
using TMPro;  
using Unity.VisualScripting;  
using UnityEngine;  
using UnityEngine.Events;  
using UnityEngine.UI;  
public class SettingManager : MenuPanels  
{  
 // Start is called before the first frame update  
 private AudioSettingPanel \_audioPanel;  
 private VideoSettingPanel \_videoPanel;  
 private GraphicsSettingPanel \_graphicPanel;  
 private ControlSettingPanel \_controlPanel;  
 [SerializeField] private GameObject buttonObject;  
 private Button[] buttons;  
 [SerializeField]private TMP\_Text path;  
   
 public override void Start()  
 {  
 buttons = buttonObject.GetComponentsInChildren<Button>();  
 \_audioPanel = FindObjectOfType<AudioSettingPanel>();  
 \_videoPanel = FindObjectOfType<VideoSettingPanel>();  
 \_graphicPanel = FindObjectOfType<GraphicsSettingPanel>();  
 \_controlPanel = FindObjectOfType<ControlSettingPanel>();  
   
 string[] panelName = new string[5] { "AudioAction", "VideoAction", "GraphicAction", "ControlAction","login"};  
 for (int i = 0; i < buttons.Length -1; i++)  
 {  
 buttons[i].onClick.AddListener((UnityAction)Delegate.CreateDelegate(typeof(UnityAction), this, panelName[i]));  
 }  
 base.Start();  
 path.text = Application.persistentDataPath;  
   
 }  
  
 private void login()  
 {  
 loginState state = FindObjectOfType<loginState>();  
 Debug.Log(state);  
 // state.LoginButton();  
 }  
 private void AudioAction()  
 {  
 PanelHide();  
 \_audioPanel.ShowHide(true); //create delegant replacement  
 buttons[0].interactable = false;  
 }  
  
 private void VideoAction()  
 {  
 PanelHide();  
 \_videoPanel.ShowHide(true);  
 buttons[1].interactable = false;  
 }  
  
 private void GraphicAction()  
 {  
 PanelHide();  
 \_graphicPanel.ShowHide(true);  
 buttons[2].interactable = false;  
 }  
  
 private void ControlAction()  
 {  
 PanelHide();  
 \_controlPanel.ShowHide(true);  
 buttons[3].interactable = false;  
 }  
   
 private void PanelHide()  
 {  
 \_graphicPanel.ShowHide(false);  
 \_videoPanel.ShowHide(false);  
 \_audioPanel.ShowHide(false);  
 \_controlPanel.ShowHide(false);  
 for (int i = 0; i < buttons.Length; i++)  
 {  
 buttons[i].interactable = true;  
 }  
 }  
  
 public void QuitSetting()  
 {  
 gameObject.SetActive(false);  
 GetComponent<Setting>().SaveSetting(); //save file  
 }  
   
}

# VideoSettingPanel.cs

ï»¿using UnityEngine;  
  
namespace MenuScreen.panels  
{  
 public class VideoSettingPanel : MenuPanels  
 {  
   
 }  
}

MenuScreen > SplashScreen

# LoadingCircle.cs

ï»¿using UnityEngine;  
  
public class LoadingCircle : MonoBehaviour  
{  
 private void Update()  
 {  
 gameObject.transform.Rotate(0, 0, 1);  
 }  
}

# SplashScreen.cs

ï»¿using System.Collections;  
using UnityEngine;  
using UnityEngine.SceneManagement;  
  
namespace MenuScreen  
{  
 public class SplashScreen : MonoBehaviour  
 {  
 private bool allowChange = false;  
 private void Start()  
 {  
 Singleton.LoadScreenclass.LoadScreen(true,true,SceneManager.GetActiveScene().buildIndex+1);  
 }  
 }  
}

UniversalScript

# Fading.cs

using System.Collections;  
using UnityEngine;  
using UnityEngine.UI;  
public class Fading : MonoBehaviour  
{  
 [HideInInspector]public float fadeSpeed;  
  
 public Fading(float \_fadespeed= 0.5f)  
 {  
 fadeSpeed = \_fadespeed;  
 }  
   
 IEnumerator FadeInA(Image ren){  
 // if (ren = null)ren = this.GetComponent<Image>();  
 Color color = ren.color;   
 ren.color = new Color(color.r, color.g, color.b, 0);  
 while(ren.color.a <=1) {  
 var fadeAmount = color.a + (fadeSpeed \* Time.deltaTime);  
  
 color = new Color(color.r, color.g,color.b, fadeAmount);  
 ren.color = color;  
 yield return null;  
 }  
 }  
 IEnumerator FadeOutA(Image ren){  
 // if (ren = null)ren = this.GetComponent<Image>();  
 Color color = ren.color;   
 ren.color = new Color(color.r, color.g, color.b, 1);  
 // Debug.Log(ren.color.a);  
 while(ren.color.a >= 0f) {  
 // Debug.Log(ren.color);  
 var fadeAmount = color.a - (fadeSpeed \* Time.deltaTime);  
 color = new Color(color.r, color.g,color.b, fadeAmount);  
 ren.color = color;  
 yield return null;  
 }  
 DestroyImmediate(ren.gameObject);  
 }  
 public void FadeIn(Image ren){  
 Debug.Log(ren);StartCoroutine(FadeInA(ren));  
 }  
 public void FadeOut(Image ren){  
 StartCoroutine(FadeOutA(ren));  
 }  
}

# health.cs

using System.Collections;  
using System.Collections.Generic;  
using UnityEngine;  
using UnityEngine.UI;  
using TMPro;  
using UnityEngine.SceneManagement;  
  
public class playerhealth : MonoBehaviour  
{  
 // Start is called before the first frame update  
 Slider HealthFront;  
 TMP\_Text healthtext;  
 public float HealthValue;  
 public float maxHealth;  
 public float chipspeed;  
 Slider fillarea;  
 Image fillareaimg;  
  
 //timer effect  
 public bool increase, decrease;  
 public float targethealth;  
 Slider HealthBack;  
 float time;  
 void Update()  
 {  
 if ((HealthBack == null || HealthFront == null || healthtext == null || fillareaimg == null) &&  
 SceneManager.GetActiveScene().buildIndex == 3)  
 {HealthBack = GameObject.Find("HealthFront").GetComponent<Slider>();  
 HealthFront = GameObject.Find("HealthBack").GetComponent<Slider>();  
 healthtext = GameObject.Find("Health").GetComponentInChildren<TMP\_Text>();  
 fillareaimg = GameObject.Find("ChipAway").GetComponent<Image>();  
 HealthValue = maxHealth;  
 HealthBack.maxValue = maxHealth; HealthBack.value = maxHealth;HealthBack.interactable = false;  
 HealthFront.maxValue = maxHealth;HealthFront.value = maxHealth;HealthFront.interactable = false;  
 targethealth =0f;}  
 UpdateUI();  
 HealthValue = Mathf.Clamp(HealthValue,0,maxHealth);  
 healthtext.text = (int)HealthValue + "/" + maxHealth;  
 }  
 void UpdateUI(){  
 // Debug.Log(targethealth);  
 gameOverCheck();  
 time += Mathf.Pow(Time.deltaTime,0.5f);  
 float speed = time \* chipspeed;  
   
 if(decrease){  
 HealthBack.value = HealthValue;  
 HealthFront.value -= speed;  
 targethealth += speed;  
 changecolor(Color.red);  
 if(targethealth>0){  
 resetchange();  
 }  
 // healthtext.text = (int)HealthFront.value + "/" + maxHealth;  
 }else if(increase){  
 HealthFront.value = HealthValue;  
 HealthBack.value +=speed;  
 targethealth -= speed;  
 changecolor(Color.blue);  
 if(targethealth < 0){  
 resetchange();  
 }  
 // healthtext.text = (int)HealthBack.value + "/" + maxHealth;  
 }  
 }  
 void gameOverCheck(){  
 if(HealthValue <=0){  
 FindObjectOfType<MazeManager>().Dead();  
 }  
 }  
 void resetchange(){  
 targethealth = 0;  
 increase = false; decrease = false;  
 changecolor(Color.white);  
 }  
 void changecolor(Color color){  
 fillareaimg.color = color;  
 }  
 public void changeHP(int hp){  
 // Debug.Log(hp);  
 time = 0;  
 HealthValue += (float)hp;  
 targethealth += (hp);  
 if(targethealth >0) increase = true; else decrease = true;  
 }  
}

# LoadingPanel.cs

ï»¿using System;  
using System.Collections;  
using System.Collections.Generic;  
using TMPro;  
using UnityEngine;  
using UnityEngine.SceneManagement;  
using UnityEngine.UI;  
  
namespace Universal  
{  
 public class LoadingPanel : MenuPanels  
 {  
 [SerializeField] private TMP\_Text progressbartxt, touchContinue, Title;  
 [SerializeField] private GameObject loadingCircle, progressbarObj, firstTime, TouchButton;  
 [SerializeField] private Slider progressbar;  
  
 public AsyncOperation loadScene;  
  
 // [HideInInspector]  
 public bool loadComplete, showprogressbar, showprogressbartext;  
 [SerializeField] private List<string> wordlist;  
 private int currentTitle = 0;  
 private float timePeriod = 0;  
  
 private void OnEnable()  
 {  
 timePeriod = 0;  
 }  
  
 public void LoadScreen(bool load, bool splash, int index = 0, bool withBar = false, bool withBarText = false,  
 bool withTitle = false)  
 {  
 loadComplete = false;  
 if (splash) firstTime.SetActive(true);  
 else firstTime.SetActive(false);  
 if (withTitle) Title.enabled = true;  
 else Title.enabled = false;  
 if (load)  
 {  
  
 loadScene = SceneManager.LoadSceneAsync(index);  
 loadScene.allowSceneActivation = false;  
 if (withBar)  
 {  
 showprogressbar = true;  
 progressbarObj.SetActive(true);  
 }  
 else  
 {  
 showprogressbar = false;  
 progressbarObj.SetActive(false);  
 }  
  
 if (withBarText)  
 {  
 showprogressbartext = true;  
 progressbartxt.enabled = true;  
 }  
 else  
 {  
 showprogressbartext = false;  
 progressbartxt.enabled = false;  
 }  
 }  
  
 loadingCircle.SetActive(true);  
 gameObject.SetActive(true);  
 // Debug.Log("error");  
 }  
  
 public void Update()  
 {  
 timePeriod += Time.deltaTime;  
 if (timePeriod > 3f)  
 {  
 currentTitle = (currentTitle + 1) % (wordlist.Count);  
 Title.text = wordlist[currentTitle];  
 timePeriod = 0;  
 }  
  
 // Debug.Log(loadComplete);  
 if (showprogressbar)  
 {  
 progressbar.value = loadScene.progress;  
 }  
  
 if (showprogressbartext)  
 {  
 progressbartxt.text = (loadScene.progress \* 100).ToString();  
 }  
  
 if (loadScene.progress < 0.5f)  
 {  
 progressbartxt.color = Color.white;  
 }  
 else  
 {  
 progressbartxt.color = Color.black;  
 if (loadScene.progress >= 0.9f)  
 {  
 progressbartxt.text = "Load Completed";  
 loadComplete = true;  
 }  
 }  
  
 if (loadComplete)  
 {  
 loadingCircle.SetActive(false);  
 touchContinue.enabled = true;  
 TouchButton.GetComponent<Button>().onClick.AddListener(ChangeScene);  
 }  
 }  
  
 private void ChangeScene()  
 {  
 loadScene.allowSceneActivation = true;  
 TouchButton.GetComponent<Button>().onClick.RemoveAllListeners();  
 }  
  
 public override void Start()  
 {  
 ShowHide(true);  
 }  
 }

# objectcloner.cs

ï»¿using UnityEngine;  
  
public class ObjectCloner : MonoBehaviour  
{  
  
 public GameObject CloneObject(GameObject originalObject, Vector3 positionOffset)  
 {  
 GameObject clone = Instantiate(originalObject, originalObject.transform.position + positionOffset, originalObject.transform.rotation);  
 clone.transform.parent = originalObject.transform.parent;  
 return clone;  
 }  
}

# PopupPanel.cs

ï»¿using TMPro;  
using UnityEngine;  
using UnityEngine.UI;  
  
 public class PopupPanel : MenuPanels  
 {  
 [SerializeField] private Sprite Error;  
 [SerializeField] TMP\_Text itemName, itemDescription;  
 [SerializeField] Image \_itemImage;  
  
 public override void Start()  
 {  
 ShowHide(true);  
 }  
  
 public void ShowDetail(string name, string description, Sprite itemImage)  
 {  
 if(itemImage != null)\_itemImage.overrideSprite = itemImage;  
 itemName.text = name;  
 itemDescription.text = description;  
   
 }  
  
 public void ShowError(string detail)  
 {  
 \_itemImage.overrideSprite = Error;  
 itemName.text = "ERROR";  
 itemDescription.text = detail;  
 }  
   
 }

# Singleton.cs

using UnityEngine;  
using UnityEngine.SceneManagement;  
using Universal;  
  
public class Singleton : MonoBehaviour  
{  
 [SerializeField] private float fadespeed;  
   
 public static readonly localSqlite Localdb = new localSqlite();  
 public static readonly UnityREST UnityAPI = new UnityREST();  
 public static readonly CSharpREST CsharpAPI = new CSharpREST();  
 public static Fading Fade;  
 public static AudioService AudioPlayer;  
 public static playerhealth HealthClass;  
 public static LoadingPanel LoadScreenclass;  
 private PopupPanel popup;  
 [Header("MazeLoad")]  
 [SerializeField] private bool \_FirstTime;  
 [SerializeField] private int \_loadNumber;  
 [SerializeField] private int difficulty;   
 [Header("Setting")]  
 [SerializeField]private bool continue\_prompt;  
 [SerializeField] private bool loggedin, gameOver;  
 public int playerID;  
 public string username;  
 public float time;  
 void Start()  
 {  
 init();  
 Localdb.init();  
 }  
  
 public void init()  
 {  
 Localdb.init();  
 if (SceneManager.GetActiveScene().buildIndex == 3)  
 {  
 HealthClass = FindObjectOfType<playerhealth>();  
 if(HealthClass == null)  
 HealthClass = gameObject.AddComponent<playerhealth>();  
 }  
 AudioPlayer = FindObjectOfType<AudioService>();  
 if(AudioPlayer == null) AudioPlayer = gameObject.AddComponent<AudioService>();  
  
 popup = FindObjectOfType<PopupPanel>();  
 try {  
 if (popup == null)  
 popup = Instantiate(Resources.Load<PopupPanel>("UI/Pop-up"), GameObject.Find("UI").transform); //canvas  
 popup.ShowHide(false);  
 }catch { }  
 try  
 {  
 LoadScreenclass = FindObjectOfType<LoadingPanel>();   
 if(LoadScreenclass == null) LoadScreenclass = Instantiate(Resources.Load<LoadingPanel>("UI/Loading"), GameObject.Find("UIPanel").transform);//panel  
 LoadScreenclass.ShowHide(false);  
   
 }catch { }   
 }  
  
 public void ShowDetail(string name, string description, Sprite itemImage = null) {  
 popup.ShowDetail(name, description,itemImage);  
 popup.ShowHide(true);  
 }  
  
 public void ShowError(string detail)  
 {  
 popup.ShowError(detail);  
 popup.ShowHide(true);  
 }  
   
 //singleton //Singleton.Instance  
 private static Singleton \_instance;  
 public static Singleton Instance{  
 get{  
 if (\_instance == null){  
 \_instance = FindObjectOfType<Singleton>();  
 if (\_instance == null){  
 \_instance = new GameObject().AddComponent<Singleton>();  
 }  
 }  
 return \_instance;  
 }  
 }  
  
 public bool FirstTime  
 {  
 get => \_FirstTime;  
 set => \_FirstTime = value;  
 }  
  
 public int LoadNumber  
 {  
 get => \_loadNumber;  
 set => \_loadNumber = value;  
 }  
  
 public int Difficulty  
 {  
 get => difficulty;  
 set => difficulty = value;  
 }  
  
 public bool ContinuePrompt  
 {  
 get => continue\_prompt;  
 set => continue\_prompt = value;  
 }  
  
 public bool Loggedin  
 {  
 get => loggedin;  
 set => loggedin = value;  
 }  
  
 public bool GameOver  
 {  
 get => gameOver;  
 set => gameOver = value;  
 }  
  
 public float Time1  
 {  
 get => time;  
 set => time = value;  
 }  
  
 public int PlayerID  
 {  
 get => playerID;  
 set => playerID = value;  
 }  
  
 public string Username  
 {  
 get => username;  
 set => username = value;  
 }  
  
 void Awake(){  
 if (\_instance != null) Destroy(this);  
 DontDestroyOnLoad(this);  
   
 try {  
 Fade = FindObjectOfType<Fading>();  
 if(Fade == null) Fade = gameObject.AddComponent<Fading>();  
 Fade.fadeSpeed = fadespeed;  
 }catch { }   
 }  
}  
public class AudioService: MonoBehaviour{  
 private AudioSource \_music, \_sfx;  
  
 public AudioSource Music  
 {  
 get => \_music;  
 set => \_music = value;  
 }  
  
 public AudioSource Sfx  
 {  
 get => \_sfx;  
 set => \_sfx = value;  
 }  
  
 void OnEnable(){  
 \_music = this.gameObject.AddComponent<AudioSource>();  
 \_sfx = gameObject.AddComponent<AudioSource>();  
 }   
 public void PlayMusic(AudioClip clip){  
 \_music.Stop();  
 \_music.loop = true;  
 \_music.clip = clip;  
 \_music.Play(0);  
 }  
  
 public void PlaySfx(AudioClip clip) {  
 \_sfx.PlayOneShot(clip);  
 }  
}

Universal > NetworkingScript

# CsharpREST.cs

using System;  
using System.Collections.Generic;  
using System.Net.Http;  
using System.Threading.Tasks;  
  
  
public class CSharpREST  
{  
 public async Task<string> GetData(bool postdata, string sql, string argument ) {  
 try  
 {  
 HttpClient client = new HttpClient();  
 HttpResponseMessage response;  
 if(postdata)  
 {  
 var values = new Dictionary<string, string>  
 {   
 { "sql", sql },  
 { "args", argument},  
 };  
 response = await client.PostAsync(new Uri("https://unity.just4fun.tk"), new FormUrlEncodedContent(values));  
 }  
 else response = await client.GetAsync("https://unity.just4fun.tk/date.php");  
 response.EnsureSuccessStatusCode();  
 var responseBody = await response.Content.ReadAsStringAsync();  
 return responseBody;  
 }catch (HttpRequestException e){ return "404";}  
 }  
}

# db.cs

using System;  
using UnityEngine;  
using Mono.Data.Sqlite;  
using System.Data;  
using System.IO;  
using System.Threading.Tasks;  
using Unity.VisualScripting;  
  
public class localSqlite  
{  
 //"URI=file:" + Application.persistentDataPath + "/" + "data.db";  
 //"URI=file:" + Application.dataPath + "\\StreamingAssets" + "/"+ DataBaseName; //editor  
 private string datapath;  
 public void init()  
 {  
 datapath = $"{Application.persistentDataPath}";  
 }  
  
 public async Task<string> Query(string sql, string dbname = "game")  
 {  
  
   
 string outputdata = "";  
 //Application.persistant must run in main thread  
 var connection =$"URI=file:{datapath}/{dbname}";  
 // Debug.Log(connection);  
 if(!File.Exists(connection.Substring(9)))  
 Task.Run(() => { WhenFileCreated(connection.Substring(9)); }).Wait();  
 try  
 {  
 IDbConnection dbcon = new SqliteConnection(connection);  
 IDbCommand dbcmd;  
 IDataReader reader;  
 dbcon.Open();  
 dbcmd = dbcon.CreateCommand();  
 dbcmd.CommandText = sql;  
 reader = dbcmd.ExecuteReader();  
 while (reader.Read())  
 {  
 int count = 0;  
 try  
 {  
 while (true)  
 {  
 outputdata += reader.GetValue(count);  
 outputdata += ";";  
 count++;  
 }  
 }  
 catch  
 {  
 outputdata += ";";  
 }  
 }  
  
 reader.Close();  
 reader = null;  
 dbcmd.Dispose();  
 dbcmd = null;  
 dbcon.Close();  
 dbcon = null;  
 return outputdata;  
 }  
 catch (Exception e)  
 {  
 Debug.Log(sql);  
 Debug.Log(e);  
 Debug.Log(connection);  
 return e.ToString();  
 }  
 }  
 public async Task WhenFileCreated(string path)  
 {  
 if (File.Exists(path)) return;  
 // return Task.FromResult(true);  
  
 var tcs = new TaskCompletionSource<bool>();  
 FileSystemWatcher watcher = new FileSystemWatcher(Path.GetDirectoryName(path));  
  
 FileSystemEventHandler createdHandler = null;  
 RenamedEventHandler renamedHandler = null;  
 createdHandler = (s, e) =>  
 {  
 if (e.Name == Path.GetFileName(path))  
 {  
 tcs.TrySetResult(true);  
 watcher.Created -= createdHandler;  
 watcher.Dispose();  
 }  
 };  
  
 renamedHandler = (s, e) =>  
 {  
 if (e.Name == Path.GetFileName(path))  
 {  
 tcs.TrySetResult(true);  
 watcher.Renamed -= renamedHandler;  
 watcher.Dispose();  
 }  
 };  
  
 watcher.Created += createdHandler;  
 watcher.Renamed += renamedHandler;  
  
 watcher.EnableRaisingEvents = true;  
 return;  
 }  
}

# UnityREST.cs

using System.Collections;  
using System.Collections.Generic;  
using UnityEngine;  
using UnityEngine.Networking;  
  
public class UnityREST  
{  
 //ASP.NET has replaced WCF in .NET3.0  
 IEnumerator PostData(){  
 WWWForm form = new WWWForm();  
 form.AddField("sql","SELECT \* FROM Leaderboard WHERE Level = ? AND Username = ?;");  
 form.AddField("args","45,hey1");  
 using UnityWebRequest request = UnityWebRequest.Post("https://unity.shalevportal.ml", form);  
 yield return request.SendWebRequest();  
 if(request.isNetworkError || request.isHttpError){  
 Debug.Log(request.error);}  
 else  
 {  
 Debug.Log(request.downloadHandler.text);  
 }  
 }  
 IEnumerator GetData(){  
 using(UnityWebRequest request = UnityWebRequest.Get("https://unity.shalevportal.ml/date.php")){  
 yield return request.SendWebRequest();  
 if(request.isNetworkError || request.isHttpError){  
 Debug.Log(request.error);  
 }else{  
 Debug.Log(request.downloadHandler.text);  
 }  
 }  
 }  
}

PHP Online Query

# Date.php

<?php

$today = date("Y-m-d H:i:s");

echo $today;

?>

# Index.php

<?php

$db\_host = '192.168.0.252';

$db\_user = 'hey';

$db\_pass = 'Ss12345678\*';

$db\_database = 'unity';

$db\_port = '2086';

$link = mysqli\_connect($db\_host,$db\_user,$db\_pass,$db\_database,$db\_port);

//check connection

$sql = $\_POST['sql']; //with ?

$argument = $\_POST['args']; //with array - according to number of ?

$argument = explode(',',$argument);

$type = "";

for ($count=0; $count < substr\_count($sql, '?'); $count++) {

$type .= "s";

}

$result = $link->prepare($sql);

$result->bind\_param($type, ...$argument); //call\_user\_func\_array(array($stmt, 'bind\_param'), $params);

$result->execute();

$result1 = $result->get\_result();

while($row = $result1->fetch\_row()) {

echo implode(",",$row);

echo ";";

}

?>

Node.JS Register and Leaderboard Stat

# Index.js

const { Console } = require('console');

var express = require('express')

const mysql = require('mysql2');

const db = mysql.createConnection({

host: "192.168.0.252",

port: 2086,

user: "hey",

password: "Ss12345678\*",

database: 'unity',

multipleStatements: true,

debug: false,

})

var app = express()

app.use(express.json()); //accept data in JSON format

app.use(express.urlencoded({ extended: true }));

var sqlquery = [`SELECT pl.username,

max(CASE WHEN l.levelID=6 THEN p.score ELSE NULL END) AS Level\_3,

max(CASE WHEN l.levelID=7 THEN p.score ELSE NULL END) AS Level\_4,

max(CASE WHEN l.levelID=8 THEN p.score ELSE NULL END) AS Level\_5,

max(CASE WHEN l.levelID=9 THEN p.score ELSE NULL END) AS Level\_6,

max(CASE WHEN l.levelID=10 THEN p.score ELSE NULL END) AS Level\_7,

max(CASE WHEN l.levelID=11 THEN p.score ELSE NULL END) AS Level\_8,

max(CASE WHEN l.levelID=12 THEN p.score ELSE NULL END) AS Level\_9

FROM Progress p INNER JOIN Level l ON p.levelID = l.levelID, Player pl

WHERE p.playerID = pl.playerID

GROUP BY pl.username;`,

`SELECT l.levelName,

l.levelID,

COUNT(p.progressID) AS no\_player, -- need count for item only

MAX(p.score) AS Best\_Score,

(SELECT pl.username FROM Progress p, Player pl

WHERE p.playerID = pl.playerID AND p.score =

(SELECT MAX(score) FROM Progress p2 WHERE p2.levelID = l.levelID LIMIT 1)) AS Best\_score,

MIN(CASE WHEN l.levelID BETWEEN 6 AND 12 THEN p.time ELSE 100000 END) AS Best\_Time,

(SELECT pl.username FROM Progress p, Player pl

WHERE p.playerID = pl.playerID AND p.time =

(SELECT MIN(time) FROM Progress p2 WHERE p2.levelID = l.levelID LIMIT 1)) AS Best\_time,

AVG(p.score) AS AVG\_Score,

AVG(p.time) AS AVG\_Time

FROM Player pl,Progress p,Level l

WHERE p.levelID = l.levelID AND p.playerID = pl.playerID

GROUP BY l.levelName

ORDER BY l.levelID;

`]

app.use('/js',express.static(\_\_dirname+'/web'));

app.get('/',(req,res)=>{

res.redirect('/pivot')

})

app.get('/register',(req,res)=>{

generateHash("hi")

res.sendFile(\_\_dirname+"/web/register.html")

})

app.get('/pivot',async(req,res)=>{

res.sendFile(\_\_dirname+"/web/pivot.html")

generateHash("Ss12354678")

})

app.get('/cross',async(req,res)=>{

res.sendFile(\_\_dirname+"/web/cross.html")

})

app.post('/registerData',async(req,res)=>{

var pass = generateHash(req.body.pass);

var sql = `INSERT INTO Player(email, hash, salt, username) VALUES ('${req.body.email}','${pass.hash}',${pass.salt},'${req.body.user}');`;

try {

await db.promise().query(sql);

res.send("0")

} catch (error) {console.log(error)

res.send("1")

}

// "SELECT password FROM Player WHERE username = '"+req.body.username+"' AND password = '"+req.body.password

})

app.post('/getData',async(req, res)=> {

console.log(typeof(req.body.detail))

const result = await db.promise().query(sqlquery[req.body.detail])

res.send(result[0])

})

app.listen(3000);

function generateHash(password) {

var hash = 0, count =10;

const salt = Math.floor(Math.random() \* 1000000);

for (let index = 0; index < password.length; index++) {

const element = password[index];

hash += element.charCodeAt(0)\* Math.pow(10,count-index) + salt;

}

return {"hash":hash.toString(16),"salt": salt};

}

NodeJS website

# Cross.html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.3/jquery.min.js"></script>

<title>Document</title>

<style>

\*{

box-sizing: border-box;

-webkit-box-sizing: border-box;

-moz-box-sizing: border-box;

}

body{

font-family: Helvetica;

-webkit-font-smoothing: antialiased;

background: rgba( 71, 147, 227, 1);

}h2{

text-align: center;

font-size: 18px;

text-transform: uppercase;

letter-spacing: 1px;

color: white;

padding: 30px 0;

}

.table-wrapper{

margin: 10px 70px 70px;

box-shadow: 0px 35px 50px rgba( 0, 0, 0, 0.2 );

}

.fl-table {

border-radius: 5px;

font-size: 12px;

font-weight: normal;

border: none;

border-collapse: collapse;

width: 100%;

max-width: 100%;

white-space: nowrap;

background-color: white;

}

.fl-table td, .fl-table th {

text-align: center;

padding: 8px;

}

.fl-table td {

border-right: 1px solid #f8f8f8;

font-size: 12px;

}

.fl-table thead th {

color: #ffffff;

background: #4FC3A1;

}

.fl-table thead th:nth-child(odd) {

color: #ffffff;

background: #324960;

}

.fl-table tr:nth-child(even) {

background: #F8F8F8;

}

/\* Responsive \*/

@media (max-width: 767px) {

.fl-table {

display: block;

width: 100%;

}

.table-wrapper:before{

content: "Scroll horizontally >";

display: block;

text-align: right;

font-size: 11px;

color: white;

padding: 0 0 10px;

}

.fl-table thead, .fl-table tbody, .fl-table thead th {

display: block;

}

.fl-table thead th:last-child{

border-bottom: none;

}

.fl-table thead {

float: left;

}

.fl-table tbody {

width: auto;

position: relative;

overflow-x: auto;

}

.fl-table td, .fl-table th {

padding: 20px .625em .625em .625em;

height: 60px;

vertical-align: middle;

box-sizing: border-box;

overflow-x: hidden;

overflow-y: auto;

width: 120px;

font-size: 13px;

text-overflow: ellipsis;

}

.fl-table thead th {

text-align: left;

border-bottom: 1px solid #f7f7f9;

}

.fl-table tbody tr {

display: table-cell;

}

.fl-table tbody tr:nth-child(odd) {

background: none;

}

.fl-table tr:nth-child(even) {

background: transparent;

}

.fl-table tr td:nth-child(odd) {

background: #F8F8F8;

border-right: 1px solid #E6E4E4;

}

.fl-table tr td:nth-child(even) {

border-right: 1px solid #E6E4E4;

}

.fl-table tbody td {

display: block;

text-align: center;

}

}

</style>

</head>

<body>

<h2>Statistics</h2>

<div class="table-wrapper">

<table class="fl-table">

<thead>

<th>levelID</th>

<th>Description</th>

<th>Best Score</th>

<th>player</th>

<th>Best Time</th>

<th>player</th>

<th>number of player</th>

<th>average score</th>

<th>average time</th>

</thead>

<tbody id="place-here"></tbody>

</table>

</div>

<script>

$( document ).ready(()=>{

console.log("hi")

$.post('/getData', {detail:1}, function(result) {

let res = '';

result.forEach(data => {

res += `<tr>

<td>${data.levelID}</td>

<td>${data.levelName}</td>

<td>${data.Best\_Score}</td>

<td>${data.Best\_score}</td>

<td>${data.Best\_Time}</td>

<td>${data.Best\_time}</td>

<td>${data.no\_player}</td>

<td>${data.AVG\_Score}</td>

<td>${data.AVG\_Time}</td>

</tr>`

})

$('#place-here').html(res)

});

})

</script>

</body>

</html>

# Done.html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

<style>

body{text-align: center;}

</style>

</head>

<body>

Thank you! Now close the browser and back to the game

</body>

</html>

# Pivot.html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.3/jquery.min.js"></script>

<title>Document</title>

<style> \*{

box-sizing: border-box;

-webkit-box-sizing: border-box;

-moz-box-sizing: border-box;

}

body{

font-family: Helvetica;

-webkit-font-smoothing: antialiased;

background: rgba( 71, 147, 227, 1);

}

h2{

text-align: center;

font-size: 18px;

text-transform: uppercase;

letter-spacing: 1px;

color: white;

padding: 30px 0;

}

.table-wrapper{

margin: 10px 70px 70px;

box-shadow: 0px 35px 50px rgba( 0, 0, 0, 0.2 );

}

.fl-table {

border-radius: 5px;

font-size: 12px;

font-weight: normal;

border: none;

border-collapse: collapse;

width: 100%;

max-width: 100%;

white-space: nowrap;

background-color: white;

}

.fl-table td, .fl-table th {

text-align: center;

padding: 8px;

}

.fl-table td {

border-right: 1px solid #f8f8f8;

font-size: 12px;

}

.fl-table thead th {

color: #ffffff;

background: #4FC3A1;

}

.fl-table thead th:nth-child(odd) {

color: #ffffff;

background: #324960;

}

.fl-table tr:nth-child(even) {

background: #F8F8F8;

}

/\* Responsive \*/

@media (max-width: 767px) {

.fl-table {

display: block;

width: 100%;

}

.table-wrapper:before{

content: "Scroll horizontally >";

display: block;

text-align: right;

font-size: 11px;

color: white;

padding: 0 0 10px;

}

.fl-table thead, .fl-table tbody, .fl-table thead th {

display: block;

}

.fl-table thead th:last-child{

border-bottom: none;

}

.fl-table thead {

float: left;

}

.fl-table tbody {

width: auto;

position: relative;

overflow-x: auto;

}

.fl-table td, .fl-table th {

padding: 20px .625em .625em .625em;

height: 60px;

vertical-align: middle;

box-sizing: border-box;

overflow-x: hidden;

overflow-y: auto;

width: 120px;

font-size: 13px;

text-overflow: ellipsis;

}

.fl-table thead th {

text-align: left;

border-bottom: 1px solid #f7f7f9;

}

.fl-table tbody tr {

display: table-cell;

}

.fl-table tbody tr:nth-child(odd) {

background: none;

}

.fl-table tr:nth-child(even) {

background: transparent;

}

.fl-table tr td:nth-child(odd) {

background: #F8F8F8;

border-right: 1px solid #E6E4E4;

}

.fl-table tr td:nth-child(even) {

border-right: 1px solid #E6E4E4;

}

.fl-table tbody td {

display: block;

text-align: center;

}

}

</style>

</head>

<body>

<h2>Performance of each player</h2>

<div class="table-wrapper">

<table class="fl-table">

<thead>

<th>player\Level</th>

<th>3</th>

<th>4</th>

<th>5</th>

<th>6</th>

<th>7</th>

<th>8</th>

<th>9</th>

</thead>

<tbody id="place-here"></tbody>

</table>

</div>

<script>

$( document ).ready(()=>{

$.post('/getData', {detail:0}, function(result) {

let res = '';

result.forEach(data => {

res += `<tr>

<td>${data.username}</td>

<td>${data.Level\_3}</td>

<td>${data.Level\_4}</td>

<td>${data.Level\_5}</td>

<td>${data.Level\_6}</td>

<td>${data.Level\_7}</td>

<td>${data.Level\_8}</td>

<td>${data.Level\_9}</td>

</tr>`

})

$('#place-here').html(res)

});

})

</script>

</body>

</html>

# Register.html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Maze Runner Registration</title>

<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.3/jquery.min.js"></script>

<script src="/js/register.js"></script>

<style>

@font-face {

font-family: "CustomFont";

src: url("https://static.itch.io/fonts/04b03.woff2") format("woff");

}

html,body{

background-image: linear-gradient(45deg, #93a5cf 0%, #e4efe9 100%);

width: 100vw;height: 100vh;

overflow-x: hidden;

text-align: center;

}

@media (max-height: 500px) {

h2{

padding-top: 0px;

}

}

#submit{

border-radius: 20px;

background: linear-gradient(225deg, #6cd4f6, #5bb2cf);

box-shadow: -7px 7px 13px #284f5c,

7px -7px 13px #a2ffff;

min-width: auto;

transition: 0.5s;

width: 100px;

}

#submit:hover, #submit:focus{

background: linear-gradient(225deg, #ffcc00, #eeff00);

}

.form{

margin: auto;

width: 90%;

min-height: 70%;

max-width: 500px;

border-radius: 39px;

background: #0095c7;

box-shadow: 20px 20px 30px #007fa9,

-20px -20px 30px #00abe5;

transition: 0.8s;

}

input{

/\* onclick no border \*/

margin: 10px auto;

display: block;

outline:none;

min-width: 250px;

width: 20%;

height:40px;

border: 0px;

border-radius: 45px;

background: linear-gradient(315deg, #6cd4f6, #5bb2cf);

box-shadow: -2px 2px 5px #4c95ad,

2px -2px 5px #7ef8ff;

text-shadow: 2px 2px 5px black;

transition:all 0.5s;

font-size: 17px;

color: azure;

text-align: center;

}

input:focus {

box-shadow: inset 13px 13px 13px #284f5c,

inset -13px -13px 13px #a2ffff;

background-color: rgb(181, 255, 205);

}

input::placeholder{ /\* Chrome, Firefox, Opera, Safari 10.1+ \*/

text-shadow: 0px;

color: white;

opacity: 0.5; /\* Firefox \*/

}

h2{

padding-top: 50px;

color: white;

font-family: "CustomFont";

font-size: 50px;

font-weight: 25;

}

input[type=text] {

direction: rtl;

border-radius: 0;

box-shadow: none;

background: transparent;

border: none;

border-bottom: 1px solid #000000;

height: auto;

margin-bottom: 20px;

}

/\* Variation of work by @mrhyddenn for Radios \*/

.container{

margin: 25px auto;

transform: scale(1.5);

}

.check {

cursor: pointer;

position: relative;

margin: auto;

width: 18px;

height: 18px;

-webkit-tap-highlight-color: transparent;

transform: translate3d(0, 0, 0);

}

.check:before {

content: "";

position: absolute;

top: -15px;

left: -15px;

width: 48px;

height: 48px;

border-radius: 50%;

background: rgba(34, 50, 84, 0.5);

opacity: 0;

transition: opacity 0.2s ease;

}

.check svg {

position: relative;

z-index: 1;

fill: none;

stroke-linecap: round;

stroke-linejoin: round;

stroke: #c8ccd4;

stroke-width: 1.5;

transform: translate3d(0, 0, 0);

transition: all 0.2s ease;

}

.check svg path {

stroke-dasharray: 60;

stroke-dashoffset: 0;

}

.check svg polyline {

stroke-dasharray: 22;

stroke-dashoffset: 66;

}

.check:hover:before {

opacity: 1;

}

.check:hover svg {

stroke: var(--accent-color, #a3e583);

}

#cbx2:checked + .check svg {

stroke: var(--accent-color, #a3e583);

}

#cbx2:checked + .check svg path {

stroke-dashoffset: 60;

transition: all 0.3s linear;

}

#cbx2:checked + .check svg polyline {

stroke-dashoffset: 42;

transition: all 0.2s linear;

transition-delay: 0.15s;

}

</style>

</head>

<body>

<div class="form">

<form>

<h2>Maze Runner <br>Account Register</h2>

<input type="text" name="username" placeholder="username">

<input type="email" name="email" placeholder="email">

<input type="password" name="password" placeholder="password">

<input type="submit" id="submit">

</form>

<pre style="color:rgb(255, 225, 0);visibility: hidden;">hi</pre>

</div>

</body>

</html>

# Register.js

// /^[a-zA-Z0-9]+$/;

// /.\*/i

$( document ).ready(()=>{

// NBP.init("mostcommon\_500", "/js", true);

$("form").submit((e)=>{

e.preventDefault();

var username = $("input[name='username']").val().trim().replace(/\_/g, "").substr(0,20).toLowerCase();

var password = $("input[name='password']").val().trim();

var email = $("input[name='email']").val().trim().toLowerCase();

var usernameCheck = check(username);

var passwordCheck = check(password);

// console.log(email.length );

if (username.length == 0 || password.length

==0 || email.length == 0){

err("All fields are required");

}else if(password.length < 8){

err("Password must be at least 8 characters long");

}else if(usernameCheck.unused == true){

err("Username contains invalid characters");

}else if(passwordCheck.lower == false || passwordCheck.upper == false || passwordCheck.number == false){

err("Password must contain at least 1 uppercase, 1 lowercase, 1 number");

}else if(password.includes("passw") || password.includes("1234")){

err("Password is too common")

}else

{

$.post('/registerData', {'user': username, 'email':email, 'pass':password}, function(result) {

console.log(result);

if (result == "0") {

window.location = "/js/done.html"

}else if (result == "1") {

err("Duplicate email or username")

}

})

}

} )

$("input").click(()=>{

$(".form").css("background", "#0095c7");

$("pre").css("visibility", "hidden");

})

$("input[type='text']").keyup(()=>{

var text =$("input[type='text']").val().replace(/\_/g, "").substr(0,20);

var length = 20-text.length;

if(length == 20){$("input[type='text']").val("")}

else $("input[type='text']").val("\_".repeat(length)+text)

})

function err(detail){

$(".form").css("background", "red");

$("pre").html(detail);

$("pre").css("visibility", "visible");

}

function check(stringval){

var islower= false,isUpper= false, isNumber= false,isSpace= false, isUnused = false;

for (let index = 0; index < stringval.length; index++) {

var char = stringval.charCodeAt(index);

if(char >= 64 && char <= 90)

{isUpper = true}

else if(char >= 97 && char <= 122){

islower = true}

else if(char >= 48 && char <= 57){

isNumber = true;}

else if(char == 32){

isSpace = true;}

else{

isUnused = true;

}

}

return {"lower":islower ,

"upper":isUpper ,

"number": isNumber ,

"space": isSpace ,

"unused": isUnused};

}

})