**Unlock the Chest**

# Story

In this game you are an adventurer on the hunt to find some treasure. The character has spent multiple years of his life conducting research about an unknown cluster of islands in the Pacific Ocean; he has been lead to believe that these islands hold treasure from the golden age of piracy. The objective of the game is to collect all of the keys on the island to then be able to open the treasure chest.

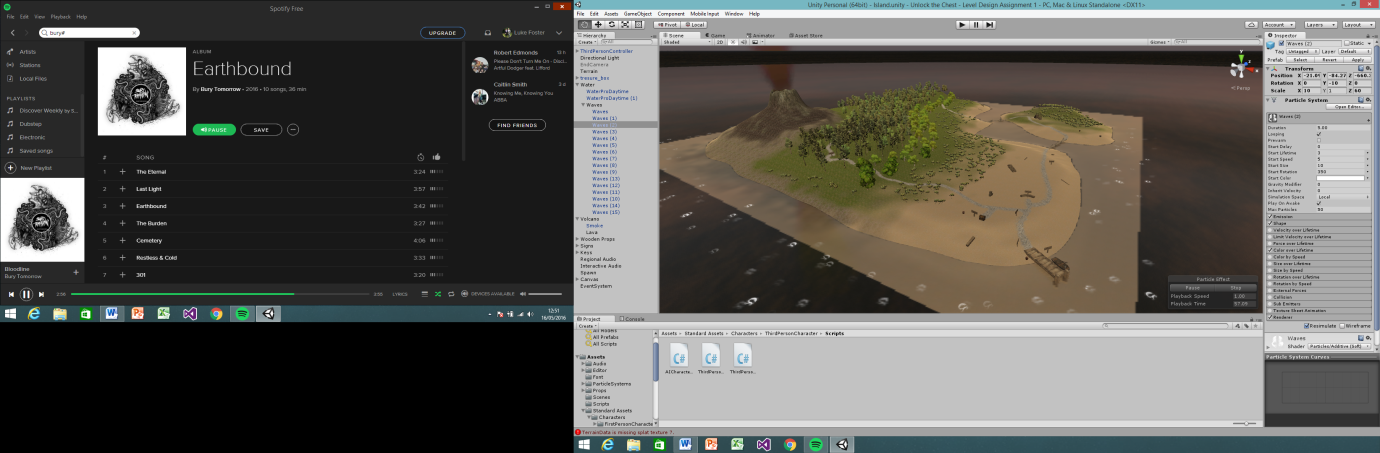
Each level in this game is a new island; on each new island there will be more complex ways of collecting these keys which will challenge the player. Also in each new level/island the keys and chest will be in completely new locations.

The level I have made is a tutorial level, this means it’s very simplistic and won’t be very challenging for the user at this stage. The idea of the level is to give full instructions to the user with signs around the map to indicate the button configurations and a path finder on the floor to guide the player to the keys. As the player gets to harder levels there will then be no signs or path finder.

UI consists of a counter that will indicate how many keys have been collected, text instructions and objective instructions. Another feature in the game is that each island has a timer starting at zero seconds counting up until the level has been completed. Once completed the score will be put up on an end screen, this will then mean players will be able to play the game competitively and try and get the best time completion.

# Screenshots

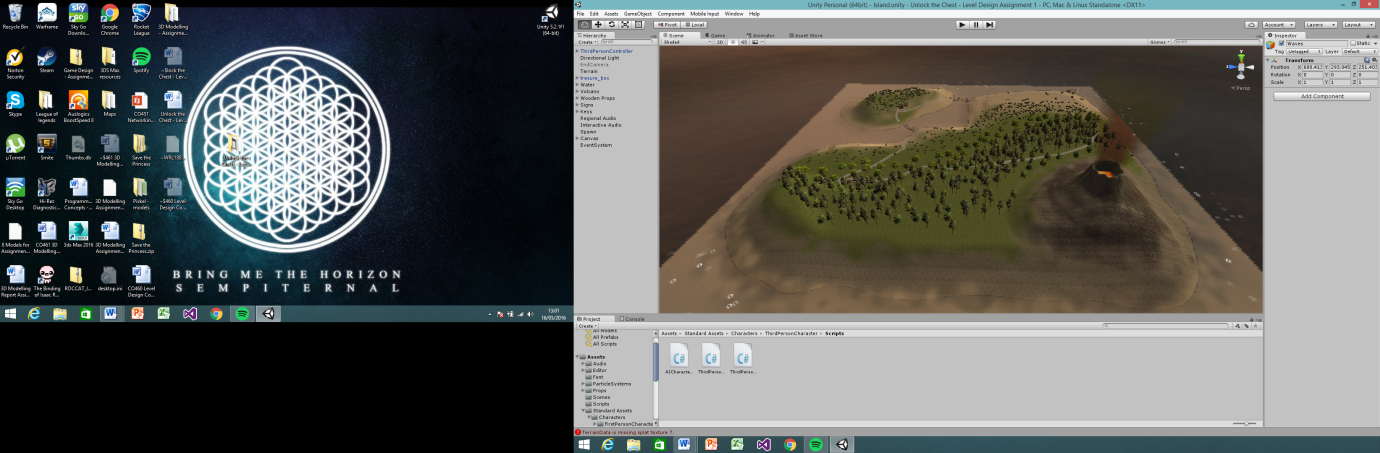
## Island Overview

In the screenshot below I have given an overview of my level with the particles effect enabled so you can see the volcano smoke and waves.

## Beach View

## Mini Island

# Back of Island



## Inside Forest

# Key Checkpoint

# Scripts

## Keys

using UnityEngine;

using System.Collections;

public class Keys : MonoBehaviour {

public GameObject Key; //Key is gameobject

public GameObject text;

public GameObject InteractiveAudio;

void OnTriggerEnter () //Trigger has been activated

{

InteractiveAudio.GetComponent<AudioSource>().Play(); // plays the audio clip from other gameobject because key gets destroyed

Counter i = text.GetComponent<Counter> (); // refers to counter script

i.Increment += 1f; // increments counter script

Destroy (Key); //key gets destroyed

}

}

## MenuScript

using UnityEngine;

using UnityEngine.UI; // Allows access to more parameters from UI

using System.Collections;

public class MenuScript : MonoBehaviour {

public Canvas QuitMenu;

public Button startText; // declaring the two start menu button variables

public Button exitText;

void Start ()

{

QuitMenu = QuitMenu.GetComponent<Canvas> ();

startText = startText.GetComponent<Button> (); // gets these components from within the parent where the script is held

exitText = exitText.GetComponent<Button> ();

QuitMenu.enabled = false;

}

public void ExitPress() // takes user to quit menu

{

QuitMenu.enabled = true;

startText.enabled = false;

exitText.enabled = false;

}

public void NoPress() // takes the user from quit menu back to start menu

{

QuitMenu.enabled = false;

startText.enabled = true;

exitText.enabled = true;

}

public void StartLevel() // plays the level

{

Application.LoadLevel ("Island");

}

public void ExitGame() // quits the game

{

Application.Quit ();

}

}

## Drown

using UnityEngine;

using System.Collections;

public class Drown : MonoBehaviour {

public GameObject Spawn;

public float Delay = 6f; // 6 second delay before death

public float Bubble = 6f;

public GameObject DrownCounter1;

public GameObject DrownCounter2; // bubble image gameobjects

public GameObject DrownCounter3;

public GameObject DrownCounter4;

public GameObject DrownCounter5;

void Update () {

if (transform.position.y <= 1.25) { // if below water level

DrownCounter1.SetActive (true);

DrownCounter2.SetActive (true);

DrownCounter3.SetActive (true); // makes bubble images visible

DrownCounter4.SetActive (true);

DrownCounter5.SetActive (true);

Delay -= Time.deltaTime;

if (Delay < 1) { // as soon as timer becomes less than 1

transform.position = Spawn.transform.position; // returns player to spawn

}

Bubble -= Time.deltaTime;

if (Bubble < 5)

{

DrownCounter1.SetActive (false); // each of these steps makes bubble image invisible after ever 1 second

}

if (Bubble < 4)

{

DrownCounter2.SetActive (false);

}

if (Bubble < 3)

{

DrownCounter3.SetActive (false);

}

if (Bubble < 2)

{

DrownCounter4.SetActive (false);

}

if (Bubble < 1)

{

DrownCounter5.SetActive (false);

}

}

else

{

Delay = 6f;

Bubble = 6f;

DrownCounter1.SetActive (false); // returns bubble images back to hidden when exiting water

DrownCounter2.SetActive (false);

DrownCounter3.SetActive (false);

DrownCounter4.SetActive (false);

DrownCounter5.SetActive (false);

}

}

}

## TextScript

using UnityEngine;

using System.Collections;

public class TextScript : MonoBehaviour {

public GameObject Message;

void OnTriggerEnter () { // when trigger box collider has been entered

Message.SetActive (true); // makes text on sign visible

}

void OnTriggerExit() { // when trigger box collider has been exited

Message.SetActive (false); // makes text on sign hidden

}

}

## Counter

using UnityEngine;

using UnityEngine.UI;

using System.Collections;

public class Counter : MonoBehaviour {

public float Increment = 0f; // Starts the counter as 0

public Text KeyNumber;

void Update () {

KeyNumber.text = "Keys: " + Increment; // changes the UI element of the key counter to updated version

}

}

## Timer

using UnityEngine;

using UnityEngine.UI;

using System.Collections;

public class Timer : MonoBehaviour {

public bool Complete = false; // Starts boolean as false

public Text CountUp;

public float Minutes = 0f; // Starts seconds and minutes as 0

public float Seconds = 0f;

void Update ()

{

if (Complete == false) {

CountUp.text = Minutes + "." + Seconds;

if (Seconds >= 60f) { // caps seconds at 60

Minutes += 1f;

Seconds = 0f; // resets seconds to 0 once hitting 60

} else {

Seconds += Time.deltaTime; // time goes up from 0

}

}

}

}

## MasterKey

using UnityEngine;

using UnityEngine.UI;

using System.Collections;

public class MasterKey : MonoBehaviour {

public GameObject Key; //Key is gameobject

public GameObject text;

public GameObject InteractiveAudio;

void OnTriggerEnter () //Trigger has been activated

{

InteractiveAudio.GetComponent<AudioSource>().Play(); // plays audio source

Counter i = text.GetComponent<Counter> (); // refers to counter script

i.Increment = 4f; // instanstly makes key counter 4

Destroy (Key); //key gets destroyed

}

}

## RegionalAudio

using UnityEngine;

using System.Collections;

public class RegionalAudio : MonoBehaviour {

public GameObject Character;

public GameObject Regional;

void OnTriggerEnter () // once box collider is triggered

{

Character.GetComponent<AudioSource> ().Pause (); // pauses constant background music

Regional.GetComponent<AudioSource> ().Play (); // plays new sound clip

}

void OnTriggerExit () // once leaving box collider

{

Character.GetComponent<AudioSource> ().UnPause (); // resumes constant background music

Regional.GetComponent<AudioSource> ().Stop (); // stops new sound clip

}

}

## MouseOrbit

using UnityEngine;

using System.Collections;

public class MouseOrbit : MonoBehaviour {

public Transform target;

public float distance = 5.0f;

public float xSpeed = 120.0f;

public float ySpeed = 120.0f;

public float yMinLimit = -20f;

public float yMaxLimit = 80f;

public float distanceMin = .5f;

public float distanceMax = 15f;

private Rigidbody rigidbody;

float x = 0.0f;

float y = 0.0f;

void Start ()

{

Vector3 angles = transform.eulerAngles;

x = angles.y;

y = angles.x;

rigidbody = GetComponent<Rigidbody>();

// Make the rigid body not change rotation

if (rigidbody != null)

{

rigidbody.freezeRotation = true;

}

}

void LateUpdate ()

{

if (target)

{

x += Input.GetAxis("Mouse X") \* xSpeed \* distance \* 0.02f;

y -= Input.GetAxis("Mouse Y") \* ySpeed \* 0.02f;

y = ClampAngle(y, yMinLimit, yMaxLimit);

Quaternion rotation = Quaternion.Euler(y, x, 0);

distance = Mathf.Clamp(distance - Input.GetAxis("Mouse ScrollWheel")\*5, distanceMin, distanceMax);

RaycastHit hit;

if (Physics.Linecast (target.position, transform.position, out hit))

{

distance -= hit.distance;

}

Vector3 negDistance = new Vector3(0.0f, 0.0f, -distance);

Vector3 position = rotation \* negDistance + target.position;

transform.rotation = rotation;

transform.position = position;

}

}

public static float ClampAngle(float angle, float min, float max)

{

if (angle < -360F)

angle += 360F;

if (angle > 360F)

angle -= 360F;

return Mathf.Clamp(angle, min, max);

}

}

## EndGame

using UnityEngine;

using UnityEngine.UI;

using System.Collections;

public class EndGame : MonoBehaviour {

public Button retryText; // declaring the two end menu button variables

public Button exitText;

public GameObject textQuestion;

public GameObject textRetry;

public GameObject textQuit; // declaring gameobjects from the hierarchy that need to be hidden

public GameObject Spawn;

public GameObject Character;

public GameObject Timer;

public GameObject KeyCounter;

public GameObject MainCamera;

public GameObject text;

public GameObject numbers;

public GameObject Sign;

public GameObject Location;

void Start ()

{

retryText = retryText.GetComponent<Button> (); //finding the ui buttons so this script can be applied to them

exitText = exitText.GetComponent<Button> (); // also means the fuction can be initiated

}

void OnTriggerEnter ()

{

Counter i = text.GetComponent<Counter> (); // refering to another script which holds counter

if (i.Increment == 4f) // only executes if 4 keys are on counter

{

GetComponent<AudioSource>().Play(); // activating chest sound

GetComponent<Animation>().Play (); // activating chest animation

Timer x = numbers.GetComponent<Timer> (); // refering to timer script

x.Complete = true; // stops the timer from continuing counting up

Timer.transform.position = Location.transform.position; //moving timer position to centre of end menu

Timer.transform.position = Location.transform.position;

Character.transform.position = Spawn.transform.position; // moves player back to spawn

Character.SetActive (false); // making player unable to move or see character

MainCamera.SetActive (true); // making the end menu visible and interactable

textQuestion.SetActive (true);

textRetry.SetActive (true);

textQuit.SetActive (true);

KeyCounter.SetActive (false); // hiding unecessary UI element now

Sign.SetActive (false);

}

}

public void Restart()

{

Application.LoadLevel ("Island"); // restarts level by loading the same scene again

}

public void Quit()

{

Application.LoadLevel ("StartMenu"); // Send user back to start menu

}

}

## ThirdPersonCharacter

using UnityEngine;

namespace UnityStandardAssets.Characters.ThirdPerson

{

[RequireComponent(typeof(Rigidbody))]

[RequireComponent(typeof(CapsuleCollider))]

[RequireComponent(typeof(Animator))]

public class ThirdPersonCharacter : MonoBehaviour

{

[SerializeField] float m\_MovingTurnSpeed = 360;

[SerializeField] float m\_StationaryTurnSpeed = 180;

[SerializeField] float m\_JumpPower = 12f;

[Range(1f, 4f)][SerializeField] float m\_GravityMultiplier = 2f;

[SerializeField] float m\_RunCycleLegOffset = 0.2f; //specific to the character in sample assets, will need to be modified to work with others

[SerializeField] float m\_MoveSpeedMultiplier = 1f;

[SerializeField] float m\_AnimSpeedMultiplier = 1f;

[SerializeField] float m\_GroundCheckDistance = 0.1f;

Rigidbody m\_Rigidbody;

Animator m\_Animator;

bool m\_IsGrounded;

float m\_OrigGroundCheckDistance;

const float k\_Half = 0.5f;

float m\_TurnAmount;

float m\_ForwardAmount;

Vector3 m\_GroundNormal;

float m\_CapsuleHeight;

Vector3 m\_CapsuleCenter;

CapsuleCollider m\_Capsule;

bool m\_Crouching;

void Start()

{

m\_Animator = GetComponent<Animator>();

m\_Rigidbody = GetComponent<Rigidbody>();

m\_Capsule = GetComponent<CapsuleCollider>();

m\_CapsuleHeight = m\_Capsule.height;

m\_CapsuleCenter = m\_Capsule.center;

m\_Rigidbody.constraints = RigidbodyConstraints.FreezeRotationX | RigidbodyConstraints.FreezeRotationY | RigidbodyConstraints.FreezeRotationZ;

m\_OrigGroundCheckDistance = m\_GroundCheckDistance;

}

public void Move(Vector3 move, bool crouch, bool jump)

{

// convert the world relative moveInput vector into a local-relative

// turn amount and forward amount required to head in the desired

// direction.

if (move.magnitude > 1f) move.Normalize();

move = transform.InverseTransformDirection(move);

CheckGroundStatus();

move = Vector3.ProjectOnPlane(move, m\_GroundNormal);

m\_TurnAmount = Mathf.Atan2(move.x, move.z);

m\_ForwardAmount = move.z;

ApplyExtraTurnRotation();

// control and velocity handling is different when grounded and airborne:

if (m\_IsGrounded)

{

HandleGroundedMovement(crouch, jump);

}

else

{

HandleAirborneMovement();

}

ScaleCapsuleForCrouching(crouch);

PreventStandingInLowHeadroom();

// send input and other state parameters to the animator

UpdateAnimator(move);

}

void ScaleCapsuleForCrouching(bool crouch)

{

if (m\_IsGrounded && crouch)

{

if (m\_Crouching) return;

m\_Capsule.height = m\_Capsule.height / 2f;

m\_Capsule.center = m\_Capsule.center / 2f;

m\_Crouching = true;

}

else

{

Ray crouchRay = new Ray(m\_Rigidbody.position + Vector3.up \* m\_Capsule.radius \* k\_Half, Vector3.up);

float crouchRayLength = m\_CapsuleHeight - m\_Capsule.radius \* k\_Half;

if (Physics.SphereCast(crouchRay, m\_Capsule.radius \* k\_Half, crouchRayLength))

{

m\_Crouching = true;

return;

}

m\_Capsule.height = m\_CapsuleHeight;

m\_Capsule.center = m\_CapsuleCenter;

m\_Crouching = false;

}

}

void PreventStandingInLowHeadroom()

{

// prevent standing up in crouch-only zones

if (!m\_Crouching)

{

Ray crouchRay = new Ray(m\_Rigidbody.position + Vector3.up \* m\_Capsule.radius \* k\_Half, Vector3.up);

float crouchRayLength = m\_CapsuleHeight - m\_Capsule.radius \* k\_Half;

if (Physics.SphereCast(crouchRay, m\_Capsule.radius \* k\_Half, crouchRayLength))

{

m\_Crouching = true;

}

}

}

void UpdateAnimator(Vector3 move)

{

// update the animator parameters

m\_Animator.SetFloat("Forward", m\_ForwardAmount, 0.1f, Time.deltaTime);

m\_Animator.SetFloat("Turn", m\_TurnAmount, 0.1f, Time.deltaTime);

m\_Animator.SetBool("Crouch", m\_Crouching);

m\_Animator.SetBool("OnGround", m\_IsGrounded);

if (!m\_IsGrounded)

{

m\_Animator.SetFloat("Jump", m\_Rigidbody.velocity.y);

}

// calculate which leg is behind, so as to leave that leg trailing in the jump animation

// (This code is reliant on the specific run cycle offset in our animations,

// and assumes one leg passes the other at the normalized clip times of 0.0 and 0.5)

float runCycle =

Mathf.Repeat(

m\_Animator.GetCurrentAnimatorStateInfo(0).normalizedTime + m\_RunCycleLegOffset, 1);

float jumpLeg = (runCycle < k\_Half ? 1 : -1) \* m\_ForwardAmount;

if (m\_IsGrounded)

{

m\_Animator.SetFloat("JumpLeg", jumpLeg);

}

// the anim speed multiplier allows the overall speed of walking/running to be tweaked in the inspector,

// which affects the movement speed because of the root motion.

if (m\_IsGrounded && move.magnitude > 0)

{

m\_Animator.speed = m\_AnimSpeedMultiplier;

}

else

{

// don't use that while airborne

m\_Animator.speed = 1;

}

}

void HandleAirborneMovement()

{

// apply extra gravity from multiplier:

Vector3 extraGravityForce = (Physics.gravity \* m\_GravityMultiplier) - Physics.gravity;

m\_Rigidbody.AddForce(extraGravityForce);

m\_GroundCheckDistance = m\_Rigidbody.velocity.y < 0 ? m\_OrigGroundCheckDistance : 0.01f;

}

void HandleGroundedMovement(bool crouch, bool jump)

{

// check whether conditions are right to allow a jump:

if (jump && !crouch && m\_Animator.GetCurrentAnimatorStateInfo(0).IsName("Grounded"))

{

// jump!

m\_Rigidbody.velocity = new Vector3(m\_Rigidbody.velocity.x, m\_JumpPower, m\_Rigidbody.velocity.z);

m\_IsGrounded = false;

m\_Animator.applyRootMotion = false;

m\_GroundCheckDistance = 0.1f;

}

}

void ApplyExtraTurnRotation()

{

// help the character turn faster (this is in addition to root rotation in the animation)

float turnSpeed = Mathf.Lerp(m\_StationaryTurnSpeed, m\_MovingTurnSpeed, m\_ForwardAmount);

transform.Rotate(0, m\_TurnAmount \* turnSpeed \* Time.deltaTime, 0);

}

public void OnAnimatorMove()

{

// we implement this function to override the default root motion.

// this allows us to modify the positional speed before it's applied.

if (m\_IsGrounded && Time.deltaTime > 0)

{

Vector3 v = (m\_Animator.deltaPosition \* m\_MoveSpeedMultiplier) / Time.deltaTime;

// we preserve the existing y part of the current velocity.

v.y = m\_Rigidbody.velocity.y;

m\_Rigidbody.velocity = v;

}

}

void CheckGroundStatus()

{

RaycastHit hitInfo;

#if UNITY\_EDITOR

// helper to visualise the ground check ray in the scene view

Debug.DrawLine(transform.position + (Vector3.up \* 0.1f), transform.position + (Vector3.up \* 0.1f) + (Vector3.down \* m\_GroundCheckDistance));

#endif

// 0.1f is a small offset to start the ray from inside the character

// it is also good to note that the transform position in the sample assets is at the base of the character

if (Physics.Raycast(transform.position + (Vector3.up \* 0.1f), Vector3.down, out hitInfo, m\_GroundCheckDistance))

{

m\_GroundNormal = hitInfo.normal;

m\_IsGrounded = true;

m\_Animator.applyRootMotion = true;

}

else

{

m\_IsGrounded = false;

m\_GroundNormal = Vector3.up;

m\_Animator.applyRootMotion = false;

}

}

}

}

## ThirdPersonUserControl

using System;

using UnityEngine;

using UnityStandardAssets.CrossPlatformInput;

namespace UnityStandardAssets.Characters.ThirdPerson

{

[RequireComponent(typeof (ThirdPersonCharacter))]

public class ThirdPersonUserControl : MonoBehaviour

{

private ThirdPersonCharacter m\_Character; // A reference to the ThirdPersonCharacter on the object

private Transform m\_Cam; // A reference to the main camera in the scenes transform

private Vector3 m\_CamForward; // The current forward direction of the camera

private Vector3 m\_Move;

private bool m\_Jump; // the world-relative desired move direction, calculated from the camForward and user input.

private void Start()

{

// get the transform of the main camera

if (Camera.main != null)

{

m\_Cam = Camera.main.transform;

}

else

{

Debug.LogWarning(

"Warning: no main camera found. Third person character needs a Camera tagged \"MainCamera\", for camera-relative controls.");

// we use self-relative controls in this case, which probably isn't what the user wants, but hey, we warned them!

}

// get the third person character ( this should never be null due to require component )

m\_Character = GetComponent<ThirdPersonCharacter>();

}

private void Update()

{

if (!m\_Jump)

{

m\_Jump = CrossPlatformInputManager.GetButtonDown("Jump");

}

}

// Fixed update is called in sync with physics

private void FixedUpdate()

{

// read inputs

float h = CrossPlatformInputManager.GetAxis("Horizontal");

float v = CrossPlatformInputManager.GetAxis("Vertical");

bool crouch = Input.GetKey(KeyCode.C);

// calculate move direction to pass to character

if (m\_Cam != null)

{

// calculate camera relative direction to move:

m\_CamForward = Vector3.Scale(m\_Cam.forward, new Vector3(1, 0, 1)).normalized;

m\_Move = v\*m\_CamForward + h\*m\_Cam.right;

}

else

{

// we use world-relative directions in the case of no main camera

m\_Move = v\*Vector3.forward + h\*Vector3.right;

}

#if !MOBILE\_INPUT

// walk speed multiplier

if (Input.GetKey(KeyCode.LeftShift)) m\_Move \*= 0.5f;

#endif

// pass all parameters to the character control script

m\_Character.Move(m\_Move, crouch, m\_Jump);

m\_Jump = false;

}

}

}