Донецкий Национальный Технический Университет

Факультет КНТ, кафедра ПМИ

По предмету «Архитектура и проектирование графических систем»

Тема «Использование графических сред для создания игровых приложений. Спецчасть : квест - комната для абитуриентов направления программная инженерия»

Выполнил:

ст. группы ИПЗ -13

Лысенко А. С.

Проверила:

Золотухина О. А.

Красноармейск 2016

Вступление

Курсовой проект выполняется студентом самостоятельно или в команде из 2-3 человек (в зависимости от сложности сценария и использованных объектов). Средствами разработки выступают среды 3ds Max, Unity3d и языка программирования скриптов.

Источник информации интернет.

Основная часть пояснительной записки

Графические материалы

В проекте должны быть реализованы следующие элементы.

1. Меню для управления игровым процессом.

2. Основное помещение игрового пространства (комната, аудитория, зал и т.д.). Требования к помещению:

− наличие минимум одной двери;

− наличие минимум одного окна;

− наличие минимум одного источника освещения;

− различные элементы помещения (пол, потолок, стены, окна, двери) должны иметь разные текстуры.

3. Предметы интерьера:

− каждый студент реализует не менее 2-х видов мебели и 3-х других разнотипных предметов;

− хотя бы один из реализованных предметов должен иметь в своем составе стеклянный или зеркальный элемент;

− хотя бы один предмет / элемент должен иметь гибкую поверхность (например, штора)

− внешний вид предметов должен быть приближен к реальным объектам;

− физика игровых предметов должна быть реализована в соответствии с поведения реальных объектов (на предметы действует сила притяжения, везде твердые поверхности нельзя проходить и т.п.), кроме случаев, когда иной тип поведения обусловлен концепцией игры.

4. Внешняя среда. Представляет собой пространство за пределами игрового помещения и может быть реализован как коридор или, например, уличное пространство. Требования к внешней среде:

− наличие растений (уличные или комнатные)

− использование систем частиц (например, осадки, фонтан, источник и т.д.);

− наличие ландшафтных или архитектурных элементов (ступени, забор, ограждение и т.д.);

− наличие природных явлений (ветер, осадки и т.д.).

Примечание: указанные элементы могут быть реализованы в рамках основного игрового помещения.

5. Игровые задачи. Каждый студент реализует минимум 2 игровых задачи, которые имеют привязку к предметной области. Примеры задач:

− найти и собрать в помещении элементы головоломки;

− составить изображения

− распутать связи;

− расшифровать код;

− найти соответствия;

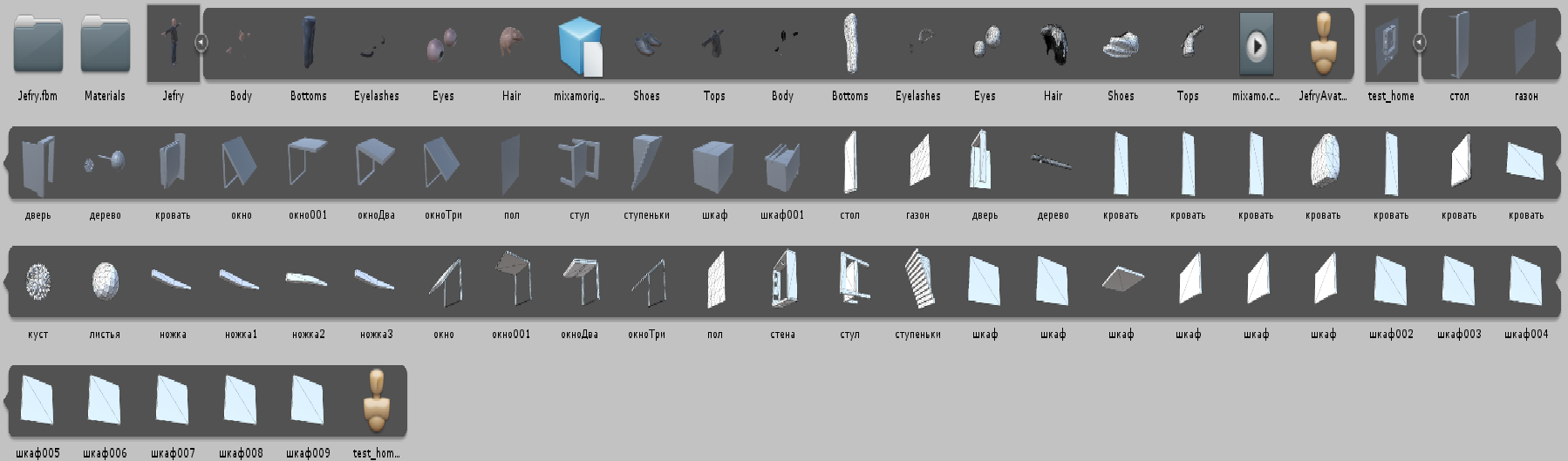
− пройти тестирование;

− разгадать ребус;

− решить кроссворд;

− разгадать шараду;

− решить логическую или математическую задачу и т.п.  
6. Система подсказок. Может быть реализована как отдельное игровое окно, часть интерьера помещения (например , специальный стенд с подсказками ) , или как интерактивный элемент игрового пространства (например за счет изменения цвета / текстуры элемента и т.д.).



Выполнены пункты

1.

2.

3. Из 3 пункта не выполнено

− хотя бы один из реализованных предметов должен иметь в своем составе стеклянный или зеркальный элемент;

− хотя бы один предмет / элемент должен иметь гибкую поверхность (например, штора)

4. Из 4 пункта не выполнено

− использование систем частиц (например, осадки, фонтан, источник и т.д.);

− наличие природных явлений (ветер, осадки и т.д.).

5. Из 5 пункта не выполнено 1 задача

6.

Постановка задач

Общая постановка задач курсового проекта:

− разработать сценарий игрового приложения для заданной предметной области (рекомендуется разработку приложения квестов типа);

− ознакомиться со средствами работы с графическими объектами в средах 3ds Max и Unity3d;

− провести анализ графических объектов, которые необходимо использовать для реализации сценария;

− провести анализ взаимодействия объектов приложения, выбрать необходимые средства программной реализации с учетом характеристик объектов и их взаимодействия;

− спроектировать и реализовать объекты сцен и необходимые анимационные фрагменты в среде 3ds Max ;

− спроектировать и реализовать другие объекты сцен и игрока (игроков ) в среде Unity3d , объекты 3ds Max экспортировать в Unity3d ;

− реализовать логику игрового приложения с использованием скриптов.

Проектирование и разработка сценария и логики игры

-концепт игры: за определенный промежуток времени выполнить задание

-игровое пространство дом и лужайка, персонаж человек

-задание зайти в здание, подойти к столу, взять квест, подойди к шкафу, взять книгу, принести книгу на стол.

- за определенный промежуток времени зайти в здание, подойти к столу, взять квест, подойди к шкафу, взять книгу, принести книгу на стол.

Проектирование и разработка объектов игрового приложения

Для каждого объекта из 3ds max:

* газон – объект plane
* дерево – объект box and sphere
* куст – sphere + extrude
* шкафы box
* кровать box
* стол box, взаимодействует с книгой, получение квеста
* стул box
* стена объект wall
* дверь объект door
* окно объект window
* ступеньки объект StraightStair

Для каждого объекта из Unity3d:

* забор cube
* потолок cube
* книга cube, предмет исчезает при подборе, взаимодействует со столом

Для каждой сцены:

* освещение point light белый, синий, зеленый, красный, желтый, расположение по углам здания, белый в центре.

Описание программной реализации

FreeLookCam (Script)

using System;

using UnityEngine;

using UnityStandardAssets.CrossPlatformInput;

namespace UnityStandardAssets.Cameras

{

public class FreeLookCam : PivotBasedCameraRig

{

// This script is designed to be placed on the root object of a camera rig,

// comprising 3 gameobjects, each parented to the next:

// Camera Rig

// Pivot

// Camera

[SerializeField] private float m\_MoveSpeed = 1f; // How fast the rig will move to keep up with the target's position.

[Range(0f, 10f)] [SerializeField] private float m\_TurnSpeed = 1.5f; // How fast the rig will rotate from user input.

[SerializeField] private float m\_TurnSmoothing = 0.1f; // How much smoothing to apply to the turn input, to reduce mouse-turn jerkiness

[SerializeField] private float m\_TiltMax = 75f; // The maximum value of the x axis rotation of the pivot.

[SerializeField] private float m\_TiltMin = 45f; // The minimum value of the x axis rotation of the pivot.

[SerializeField] private bool m\_LockCursor = false; // Whether the cursor should be hidden and locked.

[SerializeField] private bool m\_VerticalAutoReturn = false; // set wether or not the vertical axis should auto return

private float m\_LookAngle; // The rig's y axis rotation.

private float m\_TiltAngle; // The pivot's x axis rotation.

private const float k\_LookDistance = 100f; // How far in front of the pivot the character's look target is.

private Vector3 m\_PivotEulers;

private Quaternion m\_PivotTargetRot;

private Quaternion m\_TransformTargetRot;

protected override void Awake()

{

base.Awake();

// Lock or unlock the cursor.

Cursor.lockState = m\_LockCursor ? CursorLockMode.Locked : CursorLockMode.None;

Cursor.visible = !m\_LockCursor;

m\_PivotEulers = m\_Pivot.rotation.eulerAngles;

m\_PivotTargetRot = m\_Pivot.transform.localRotation;

m\_TransformTargetRot = transform.localRotation;

}

protected void Update()

{

HandleRotationMovement();

if (m\_LockCursor && Input.GetMouseButtonUp(0))

{

Cursor.lockState = m\_LockCursor ? CursorLockMode.Locked : CursorLockMode.None;

Cursor.visible = !m\_LockCursor;

}

}

private void OnDisable()

{

Cursor.lockState = CursorLockMode.None;

Cursor.visible = true;

}

protected override void FollowTarget(float deltaTime)

{

if (m\_Target == null) return;

// Move the rig towards target position.

transform.position = Vector3.Lerp(transform.position, m\_Target.position, deltaTime\*m\_MoveSpeed);

}

private void HandleRotationMovement()

{

if(Time.timeScale < float.Epsilon)

return;

// Read the user input

var x = CrossPlatformInputManager.GetAxis("Mouse X");

var y = CrossPlatformInputManager.GetAxis("Mouse Y");

// Adjust the look angle by an amount proportional to the turn speed and horizontal input.

m\_LookAngle += x\*m\_TurnSpeed;

// Rotate the rig (the root object) around Y axis only:

m\_TransformTargetRot = Quaternion.Euler(0f, m\_LookAngle, 0f);

if (m\_VerticalAutoReturn)

{

// For tilt input, we need to behave differently depending on whether we're using mouse or touch input:

// on mobile, vertical input is directly mapped to tilt value, so it springs back automatically when the look input is released

// we have to test whether above or below zero because we want to auto-return to zero even if min and max are not symmetrical.

m\_TiltAngle = y > 0 ? Mathf.Lerp(0, -m\_TiltMin, y) : Mathf.Lerp(0, m\_TiltMax, -y);

}

else

{

// on platforms with a mouse, we adjust the current angle based on Y mouse input and turn speed

m\_TiltAngle -= y\*m\_TurnSpeed;

// and make sure the new value is within the tilt range

m\_TiltAngle = Mathf.Clamp(m\_TiltAngle, -m\_TiltMin, m\_TiltMax);

}

// Tilt input around X is applied to the pivot (the child of this object)

m\_PivotTargetRot = Quaternion.Euler(m\_TiltAngle, m\_PivotEulers.y , m\_PivotEulers.z);

if (m\_TurnSmoothing > 0)

{

m\_Pivot.localRotation = Quaternion.Slerp(m\_Pivot.localRotation, m\_PivotTargetRot, m\_TurnSmoothing \* Time.deltaTime);

transform.localRotation = Quaternion.Slerp(transform.localRotation, m\_TransformTargetRot, m\_TurnSmoothing \* Time.deltaTime);

}

else

{

m\_Pivot.localRotation = m\_PivotTargetRot;

transform.localRotation = m\_TransformTargetRot;

}

}

}

}

ProtectCameraFromWallClip (Script)

using System;

using System.Collections;

using UnityEngine;

namespace UnityStandardAssets.Cameras

{

public class ProtectCameraFromWallClip : MonoBehaviour

{

public float clipMoveTime = 0.05f; // time taken to move when avoiding cliping (low value = fast, which it should be)

public float returnTime = 0.4f; // time taken to move back towards desired position, when not clipping (typically should be a higher value than clipMoveTime)

public float sphereCastRadius = 0.1f; // the radius of the sphere used to test for object between camera and target

public bool visualiseInEditor; // toggle for visualising the algorithm through lines for the raycast in the editor

public float closestDistance = 0.5f; // the closest distance the camera can be from the target

public bool protecting { get; private set; } // used for determining if there is an object between the target and the camera

public string dontClipTag = "Player"; // don't clip against objects with this tag (useful for not clipping against the targeted object)

private Transform m\_Cam; // the transform of the camera

private Transform m\_Pivot; // the point at which the camera pivots around

private float m\_OriginalDist; // the original distance to the camera before any modification are made

private float m\_MoveVelocity; // the velocity at which the camera moved

private float m\_CurrentDist; // the current distance from the camera to the target

private Ray m\_Ray; // the ray used in the lateupdate for casting between the camera and the target

private RaycastHit[] m\_Hits; // the hits between the camera and the target

private RayHitComparer m\_RayHitComparer; // variable to compare raycast hit distances

private void Start()

{

// find the camera in the object hierarchy

m\_Cam = GetComponentInChildren<Camera>().transform;

m\_Pivot = m\_Cam.parent;

m\_OriginalDist = m\_Cam.localPosition.magnitude;

m\_CurrentDist = m\_OriginalDist;

// create a new RayHitComparer

m\_RayHitComparer = new RayHitComparer();

}

private void LateUpdate()

{

// initially set the target distance

float targetDist = m\_OriginalDist;

m\_Ray.origin = m\_Pivot.position + m\_Pivot.forward\*sphereCastRadius;

m\_Ray.direction = -m\_Pivot.forward;

// initial check to see if start of spherecast intersects anything

var cols = Physics.OverlapSphere(m\_Ray.origin, sphereCastRadius);

bool initialIntersect = false;

bool hitSomething = false;

// loop through all the collisions to check if something we care about

for (int i = 0; i < cols.Length; i++)

{

if ((!cols[i].isTrigger) &&

!(cols[i].attachedRigidbody != null && cols[i].attachedRigidbody.CompareTag(dontClipTag)))

{

initialIntersect = true;

break;

}

}

// if there is a collision

if (initialIntersect)

{

m\_Ray.origin += m\_Pivot.forward\*sphereCastRadius;

// do a raycast and gather all the intersections

m\_Hits = Physics.RaycastAll(m\_Ray, m\_OriginalDist - sphereCastRadius);

}

else

{

// if there was no collision do a sphere cast to see if there were any other collisions

m\_Hits = Physics.SphereCastAll(m\_Ray, sphereCastRadius, m\_OriginalDist + sphereCastRadius);

}

// sort the collisions by distance

Array.Sort(m\_Hits, m\_RayHitComparer);

// set the variable used for storing the closest to be as far as possible

float nearest = Mathf.Infinity;

// loop through all the collisions

for (int i = 0; i < m\_Hits.Length; i++)

{

// only deal with the collision if it was closer than the previous one, not a trigger, and not attached to a rigidbody tagged with the dontClipTag

if (m\_Hits[i].distance < nearest && (!m\_Hits[i].collider.isTrigger) &&

!(m\_Hits[i].collider.attachedRigidbody != null &&

m\_Hits[i].collider.attachedRigidbody.CompareTag(dontClipTag)))

{

// change the nearest collision to latest

nearest = m\_Hits[i].distance;

targetDist = -m\_Pivot.InverseTransformPoint(m\_Hits[i].point).z;

hitSomething = true;

}

}

// visualise the cam clip effect in the editor

if (hitSomething)

{

Debug.DrawRay(m\_Ray.origin, -m\_Pivot.forward\*(targetDist + sphereCastRadius), Color.red);

}

// hit something so move the camera to a better position

protecting = hitSomething;

m\_CurrentDist = Mathf.SmoothDamp(m\_CurrentDist, targetDist, ref m\_MoveVelocity,

m\_CurrentDist > targetDist ? clipMoveTime : returnTime);

m\_CurrentDist = Mathf.Clamp(m\_CurrentDist, closestDistance, m\_OriginalDist);

m\_Cam.localPosition = -Vector3.forward\*m\_CurrentDist;

}

// comparer for check distances in ray cast hits

public class RayHitComparer : IComparer

{

public int Compare(object x, object y)

{

return ((RaycastHit) x).distance.CompareTo(((RaycastHit) y).distance);

}

}

}

}

ThirdPersonCharacter (Script)

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, ~0, QueryTriggerInteraction.Ignore))

{

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, ~0, QueryTriggerInteraction.Ignore))

{

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 (Script)

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;

}

}

}

MissionPlayer (Script)

using UnityEngine;

using System.Collections;

public class MissionPlayer : MonoBehaviour

{

public bool quest; // отображает название квеста на экране;

public string MissionText; // название квеста;

public string ObjectTag; // тэг объекта;

public bool MissionObjects; //отображает собран предмет или нет;

public int Money; // количество денег;

void OnGUI()

{

if (quest)

{

GUI.Label(new Rect(20, 80, 300, 30), " " + MissionText); // значение названия квеста будет браться из скрипта Misson Bot;

if (MissionObjects)

{ // если предмет собран;

GUI.Label(new Rect(150, 80, 200, 30), "[Предмет собран]"); // выводит надпись;

}

}

GUI.Label(new Rect(20, 100, 100, 30), "Деньги: " + Money); // отображает количество денег на экране;

}

}

MissionObejct (Script)

using UnityEngine;

using System.Collections;

public class MissionObject : MonoBehaviour

{

private MissionPlayer MP; // опять подключаем скрипт MissionPlayer;

void Start()

{

MP = GameObject.FindGameObjectWithTag("Player").GetComponent<MissionPlayer>();

} // определяем что скрипт MissionPlayer будет находится на персонаже с тэгом player;

void OnMouseUp()

{

if (Input.GetMouseButtonUp(0)) // если при нажатии на правую кнопку мыши;

{

if (MP.ObjectTag == gameObject.tag) // и при том что тэг объекта равен тому значению которое написано в ObjectTag;

{

MP.MissionObjects = true; // то переменная принимает значение true, т.е. считается что предмет собран;

Destroy(gameObject); // и удаляем этот объект со сцены;

}

}

}

}

MissionBot (Script)

using UnityEngine;

using System.Collections;

public class MissionBot : MonoBehaviour

{

private float myTimer = -1;

private string gameOverText = "";

private bool gameOverActive;

int window = 0;

public bool quest; //переменная, которая обозначает взят квест или нет;

public bool vis = true; // переменная, которая будет отображать диалог между персонажами;

public string missionText; // Текст который будет отображать наименование квеста;

public string missionTag; //Тэг объекта, который необходимо принести;

private MissionPlayer MP; // подключаем скрипт MissionPlayer;

void Start()

{

MP = GameObject.FindGameObjectWithTag("Player").GetComponent<MissionPlayer>(); // определяем что скрипт MissionPlayer будет находится на персонаже с тэгом player;

}

void Update()

{

if (myTimer > 0)

{

myTimer -= Time.deltaTime;

}

if (myTimer <= 0)

{

gameOverActive = true;

myTimer = 0.0f;

}

GameObject MissionTagScanner = GameObject.FindGameObjectWithTag("Player"); // персонаж у которого берем квест будет взаимодействовать только с тем объектом у которого тэг Player;

if (Input.GetKeyDown(KeyCode.E) & Vector3.Distance(transform.position, MissionTagScanner.transform.position) < 2) // При нажатии на клавишу Е и при дистанции между персонажами меньше чем 2;

{

vis = true; // переменная vis принимает значение true;

}

}

void OnGUI()

{

GUI.Label(new Rect(10, 10, 100, 100), "Время " + myTimer);

if (window == 0)

{ // теперь главное меню активировано при window = 0

GUI.Box(new Rect(Screen.width / 2 - 100, Screen.height / 2 - 100, 200, 100), "Главное меню");

myTimer = 15;

if (GUI.Button(new Rect(Screen.width / 2 - 90, Screen.height / 2 - 80, 180, 30), "Играть"))

{

window = 1;

}

if (GUI.Button(new Rect(Screen.width / 2 - 90, Screen.height / 2 - 40, 180, 30), "Выход"))

{

window = 1;

myTimer = 0.0f;

gameOverActive = true;

}

}

if (gameOverActive == true && MP.Money == 0 && myTimer == 0)

GUI.Label(new Rect(Screen.width / 2 - 50, Screen.height / 2 - 50, 100, 100), "Игра закончена");

if (MP.Money > 0 && myTimer >= 0)

GUI.Label(new Rect(Screen.width / 2 - 50, Screen.height / 2 - 50, 100, 100), "Победа!");

if (vis) // если переменная равна true;

{

if (!quest) // и если квест еще не взят;

{

GUI.Box(new Rect((Screen.width - 300) / 2, (Screen.height - 300) / 2, 300, 300), "Квест"); // то на экране отображается окно с названием Квест;

GUI.Label(new Rect((Screen.width - 300) / 2 + 5, (Screen.height - 300) / 2 + 15, 290, 250), "Принеси мне книгу"); // и текстом который описывает квест;

if (GUI.Button(new Rect((Screen.width - 100) / 2, (Screen.height - 300) / 2 + 250, 100, 40), "Ок")) // при нажатии на кнопку Ok;

{

quest = true; // квест взят;

MP.quest = true; // отображает название квеста на экране;

MP.MissionText = "Принести книгу"; // текст квеста;

MP.ObjectTag = missionTag; // тэг объекта который необходимо принести;

vis = false; // все диалоговые окна закрываются;

}

}

else

{ // если же квест уже взят;

GUI.Box(new Rect((Screen.width - 300) / 2, (Screen.height - 300) / 2, 300, 300), "Квест");

GUI.Label(new Rect((Screen.width - 300) / 2 + 5, (Screen.height - 300) / 2 + 15, 290, 250), "Принес?"); //то описание квеста меняется на другой текст;

if (MP.MissionObjects) // если вы уже подобрали объект;

{

if (GUI.Button(new Rect((Screen.width - 100) / 2, (Screen.height - 300) / 2 + 250, 100, 40), "Да")) // то появится кнопка да, при нажатии на которую;

{

quest = false; // переменная квест принимает значение false, т.е. не взят ;

MP.quest = false; // название квеста не будет отображаться на экране ;

MP.MissionText = ""; // убирается название квеста;

MP.ObjectTag = ""; // обнуляется тэг объекта;

MP.MissionObjects = false; // объект считается не подобранным;

MP.Money = MP.Money + 100; //добавление денег за выполнение квеста;

vis = false; // диалоговое окно закрывается;

}

}

else

{ // если вы еще не подобрали объект;

if (GUI.Button(new Rect((Screen.width - 100) / 2, (Screen.height - 300) / 2 + 250, 100, 40), "Нет")) // то вместо кнопки да, будет кнопка нет;

{

vis = false; // при нажатии на которую, окно просто закроется;

}

}

}

}

}

}

Вывод

Готовый продукт в котором, за определенный промежуток времени зайти в здание, подойти к столу, взять квест, подойди к шкафу, взять книгу, принести книгу на стол.

