**КОНТРОЛЬНЫЕ ВОПРОСЫ**

**ПО ТЕМЕ «РАЗРАБОТКА СИМУЛЯТОРА УСТАНОВКИ»**

***(все строки программного кода должны быть прокомментированы)***

1. Программа для обхода камерой вокруг установки

**public class Scroll : MonoBehaviour**

**{**

**// Целевая позиция, вокруг которой будет вращаться камера**

**[SerializeField]**

**Transform targetPos;**

**// Чувствительность вращения камеры**

**int sensivity = 3;**

**// Максимальная дистанция от камеры до цели**

**int maxdistance = 20;**

**// Минимальная дистанция от камеры до цели**

**int mindistance = 1;**

**void Update()**

**{**

**// Проверяем, удерживается ли правая кнопка мыши**

**if (Input.GetMouseButton(1))**

**{**

**// Вращаем камеру вокруг целевой позиции по оси Y**

**// Угол вращения определяется движением мыши по оси X, умноженным на чувствительность**

**transform.RotateAround(targetPos.position, Vector3.up, Input.GetAxis("Mouse X") \* sensivity);**

**}**

**}**

**}**

**Класс Scroll**: Позволяет вращать камеру вокруг заданной точки (targetPos) при удерживании правой кнопки мыши. Угол вращения зависит от движения мыши по оси X.

Данный код отвечает за вращение игрового объекта, на котором он висит, вокруг заданной точки.

Точка, вокруг которой будет происходить вращение объявляется в поле targetPos, типа Transform.

Помимо этого, есть приватные поля, отвечающие за некоторую настройку проекта:

Sensitivity – поле, отвечающее за настройку (настройкой это назвать сложно, просто захардкоженая циферка) чувствительность.

maxDistance и minDistance (этот восхитительный нейминг…) конкретно к этому блоку кода не относятся, тем не менее, просто скажу, что это поля отвечающие за минимальную и максимальную дистанцию приближения.

1. Программа сдвига камеры с ограничениями вдоль и вглубь помещения с установкой

**…**

**bool ControlDistance(float distance)**

**{**

**// Проверка, находится ли расстояние до цели в допустимых пределах**

**if (distance > mindistance && distance < maxdistance) return true;**

**return false;**

**}**

**void Update()**

**{**

**float x = Input.GetAxis("Horizontal"); // Получение ввода для горизонтального движения**

**float y = Input.GetAxis("Vertical"); // Получение ввода для вертикального движения**

**// Если есть движение по одной из осей**

**if (x != 0 || y != 0)**

**{**

**// Новая позиция камеры с учетом ввода и чувствительности**

**Vector3 newpos = transform.position + (transform.TransformDirection(new Vector3(x, 0, 0)) + Vector3.up \* y) / sensivity;**

**// Если новая позиция находится в допустимых границах, перемещаем камеру**

**if (ControlDistance(Vector3.Distance(newpos, targetPos.position)))**

**transform.position = newpos;**

**}**

**}**

 **Функция ControlDistance**: Проверяет, находится ли расстояние между камерой и целевой позицией в заданных границах (минимальной и максимальной).

 **Логика в Update**: Управляет движением камеры по горизонтали и вертикали в ответ на ввод пользователя, обновляя позицию только при соблюдении условий расстояния.

Данный участок кода предоставляет метод проверки дистанции на то, чтобы текущий зум был в пределах конкретных рамок (minDistance, maxDistance объявленные и инициализированные выше).

После чего, максимально где и куда вставлен код, который в теории должен быть в Update, но окей. Там мы получаем вертикальный и горизонтальный ввод с клавиатуры, проверяя не равны ли они нулю. В случае же, если хотя бы одна не равна нулю, мы создаем новую позицию с помощью структуры Vector3 где просто к текущей позиции добавляем некоторую сформированный вектор направления передвижения (**transform.TransformDirection(new Vector3(x, 0, 0)) + Vector3.up \* y) / sensivity).**

Дальше просто проверяем, не выходит ли новая позиция за рамки допустимы и

в случае, если не выходит – присваиваем позиции нашего объекта новое положение

1. Программа приближения и удаления камеры относительно установки

**…**

**// Проверка, прокручивается ли колесо мыши**

**if (Input.GetAxis("Mouse ScrollWheel") != 0)**

**{**

**// Вычисляем новую позицию камеры на основе прокрутки колесика**

**Vector3 newpos = transform.position +**

**transform.TransformDirection(Vector3.forward \* Input.GetAxis("Mouse ScrollWheel") \* scrollSpeed);**

**// Если новая позиция в допустимых границах, обновляем позицию камеры**

**if (ControlDistance(Vector3.Distance(newpos, targetPos.position))) transform.position = newpos;**

**}**

**Проверка прокрутки колесика мыши**: Если колесико прокручивается, вычисляется новая позиция камеры, и если она в допустимых границах, позиция обновляется.

Представленный выше код считывает, есть ли прокрутка колеса мыши, после чего формирует вектор новой позиции, проверяет, не выходит ли он за пределы и если не выходит – присваивает новую позицию

1. Правила создания и размещения на экране элементов интерфейса CANVAS

Холст (**Canvas**) в редакторе Unity представляет собой 2-мерный прозрачный объект, который накладывается на экран и обозначает границы экрана, а размещаемые на холсте **элементы UI** появляются на этом экране в заданных точках.

У всех объектов UI, размещаемых на холсте, существует **точка привязки**, отображаемая в редакторе в виде наклонного крестика с обводкой. Это инструмент для позиционирования элементов интерфейса.

*Привязкой (****Anchor****) объекта называется точка его присоединения к холсту или экрану, относительно этой точки указывается положение объекта*.

При этом объект остается статичным относительно точки привязки, а сама она может перемещаться относительно холста. Точку привязки можно задать, например, как «центр экрана», и она будет оставаться в центре, когда экран меняет свой размер. Аналогично привязка к правой стороне экрана позволит объекту оставаться справа даже при изменении размеров

Программирование интерактивного взаимодействия пользователя с элементами **UI** сводится к стандартной процедуре, общей для всех элементов:

1. В сцене на основе **CANVAS** создается рабочий **UI**-объект и кнопка **Button**.
2. К созданному **UI**-объекту добавляется сценарий, который будет вызываться при обращении к нему при совершении событий, доступных для кнопок.
3. В сценарий необходимо добавить директиву **using UnityEngine.UI;** для работы с кодом **UI.**
4. Для кнопки выбирается доступное для ее обработки событие и затем создается ее связь с **UI**-объектом, к которому присоединен разработанный предварительно соответствующий **сценарий**.
5. Для кнопки выбирается в качестве функции имя сценария, добавленного для **UI**-объекта, и в раскрывающемся списке функций указывается та функция в сценарии, которая должна выполнить действие при совершении события с кнопкой.
6. Принцип обработки щелчка мышью по кнопке

**using System.Collections;**

**using UnityEngine;**

**using UnityEngine.UI;**

**public class InText : MonoBehaviour {**

**// Ссылка на текстовое поле, в которое будет выводиться сообщение**

**[SerializeField]**

**Text message;**

**// Метод, вызываемый при щелчке по кнопке**

**public void OnPointer() {**

**// Устанавливаем текст в текстовом поле**

**message.text = "Установить начальные значения";**

**}**

**}**

По умолчанию кнопка реагирует только на событие **OnClick**, в то время как UI-элементы на CANVAS могут отвечать на разные варианты взаимодействий. Событие **OnClick** кнопки отвечает только на полноценный щелчок - кнопка мыши нажимается, а затем отпускается.

Для программирования взаимодействий, отличных от **OnClick**, используется компонент **EventTrigger** - добавим **Add Component** к кнопке новый компонент **Event,** а в меню этого компонента выберем вариант **EventTrigger**. Щелкнув на кнопке **Add New Event Type** раскроем список из 17 различных событий для выбора и добавления их к компоненту **EventTrigger.**

Например, для программирования реакции на надвижение курсора мыши выберем вариант **PointerEnter**. Появится пустое поле для этого события, полностью аналогичное полю для события OnClick. Щелкните на кнопке со значком + (плюс), чтобы добавить элемент, и перетащите на этот элемент объект со скриптом. Последовательность действий будет той же, что и для события **OnClick**, просто на этот раз мы укажем реакцию на другое событие – **PointerEnter**.

*Для использования переменных, связываемых с элементами* ***Canvas*** *необходимо в список директив скрипта добавить модуль* ***using UnityEngine.UI;***

1. Принцип обработки надвижения и ухода курсора мыши с кнопки

**using UnityEngine;**

**using System.Collections;**

**public class SettingsPopup : MonoBehaviour {**

**void Start() {**

**// Закрываем всплывающее окно при старте**

**Close();**

**}**

**// Метод для открытия настроек**

**public void OnOpenSettings() {**

**Open();**

**}**

**// Метод для открытия всплывающего окна**

**public void Open() {**

**gameObject.SetActive(true);** //   АКТИВИРОВАТЬ ОБЪЕКТ, ЧТОБЫ ОТКРЫТЬ ОКНО.

**}**

**// Метод для закрытия всплывающего окна**

**public void Close() {**

**gameObject.SetActive(false);** // ДЕАКТИВИРОВАТЬ ОБЪЕКТ, ЧТОБЫ ЗАКРЫТЬ ОКНО

**}**

**}**

1. Программирование выбора оптимального ракурса размещения камеры при щелчке мышью по кнопке

**public class Replacer : MonoBehaviour**

**{**

**// Флаг для контроля движения**

**bool move = true;**

**// Начальная и целевая позиции для перемещения**

**Vector3 startPosition;**

**Vector3 needPosition;**

**float speed = 0.01f; // Скорость перемещения**

**float offset = 0; // Смещение для интерполяции**

**Quaternion startRotation; // Начальная ориентация**

**Quaternion needRotaton; // Целевая ориентация**

**// Метод для инициализации перемещения**

**public void Move1()**

**{**

**move = true; // Включаем движение**

**startPosition = transform.position; // Запоминаем начальную позицию**

**startRotation = transform.rotation; // Запоминаем начальную ориентацию**

**needPosition = new Vector3(277.0f, 251, 6); // Задаем целевую позицию**

**needRotaton = Quaternion.AngleAxis(-180, new Vector3(0, 1, 0)); // Задаем целевую ориентацию**

**}**

**void Update()**

**{**

**// Если движение включено**

**if (move)**

**{**

**offset += speed; // Увеличиваем смещение**

**// Интерполируем позицию и ориентацию между начальной и целевой**

**transform.position = Vector3.Lerp(startPosition, needPosition, offset);**

**transform.rotation = Quaternion.Slerp(startRotation, needRotaton, offset);**

**// Проверяем, достигнуто ли целевое положение**

**if (offset >= 1)**

**{**

**move = false; // Останавливаем движение**

**offset = 0; // Сбрасываем смещение**

**}**

**}**

**}**

**}**

**Класс Replacer**: Инициализирует перемещение камеры к заданной позиции и ориентации, используя интерполяцию для плавного перемещения.

Этот класс необходим для возврата к исходной позиции и вращению (да, начальная позиция захардкожена в методе Move1() ☹)

Пройдемся по полям:

Move – поле, отвечает за то, в движении ли сейчас объект.

startPosition и startRotation – начальная позиция и вращение игрового объекта (начальная в данном контексте – позиция, которая была на момент вызова метода Move1())

speed – некоторая скорость, с которой двигается объект.

Offset – прогресс передвижения от начальной точки к конечной.

needPosition и needRotation – позиция и вращение, в которые необходимо переместить текущий объект.

1. Программирование подсветки элемента установки при наведении курсора мыши на кнопку

**using System.Collections.Generic;**

**using UnityEngine;**

**public class ChangeColor : MonoBehaviour**

**{**

**// Метод для изменения цвета на красный**

**public void ChangCol()**

**{**

**// Получаем компонент Renderer и меняем цвет материала на красный**

**GetComponent<Renderer>().material.color = new Color(1, 0, 0);**

**}**

**// Метод для изменения цвета обратно на белый**

**public void ChangCol1()**

**{**

**// Получаем компонент Renderer и меняем цвет материала обратно на белый**

**GetComponent<Renderer>().material.color = new Color(1, 1, 1);**

**}**

**}**

1. Программирование появления на экране текстового окна при щелчке по кнопке

|  |
| --- |
| **using System.Collections;**  **using System.Collections.Generic;**  **using UnityEngine;**  **public class TaskBtnScript : MonoBehaviour {**  **bool opened = false;**  **[SerializeField]**  **GameObject text;**  **public void ToggleText()**  **{**  **if (opened)**  **CloseText();**  **else**  **OpenText();**  **}**  **private void OpenText()**  **{**  **text.SetActive(true);**  **}**  **private void CloseText()**  **{**  **text.SetActive(false);**  **}**  **void Start () {**  **CloseText();**  **}**  **}** |

1. Настройка проекта для обработки щелчков мышью по 3D-объектам сцены(xz)
2. Правила создания таблицы результатов эксперимента на симуляторе установки

Для работы с таблицей на информационной панели для практики должны быть предусмотрены кнопки **Button:** кнопка для записи значения в таблицу, кнопка для отображения таблицы и кнопка для очистки таблицы (на рисунке ниже для них использованы соответствующие изображения).

Также на информационной панели должно присутствовать текстовое поле **InputText** для ввода полученных значений со шкалы прибора и занесения его в таблицу.

В ходе выполнения лабораторной работы снятые измерения с установки записываются в нужные ячейки таблицы по нажатию кнопки «Записать», для просмотра содержания таблицы используется событие наведение курсора на кнопку «Таблица», а для очистки таблицы от записей - наведение курсора на кнопку «Очистка».

После осуществления всех необходимых заданий из раздела «ПРАКТИКА» выводится соответствующее сообщение о завершении работы с симулятором установки.

Вся таблица – заголовки, названия полей и т.п. строится из текстовых объектов UI соответствующего размера с фиксированным текстом, а для ячеек, куда нужно записывать результаты эксперимента, используется символ подчеркивания или минуса, чтобы было проще находить нужные ячейки таблицы при записи в них значений по нажатию кнопки «Записать».

Для создания мы создаем новый UI элемент для Canvas и в нем , если это Image с рисунком таблицы , мы создаемы текстовые поля для записи в них значений с уже выставленными занаками тире ( “ – “ );

Если Image это лишь обложка то уже в нем помимо полей для записи создаются и поля с названиями столбцов и т.д.

Для записи в текстовые объекты (ячейки таблицы) значений используется, например, для текстового объекта с именем **name1** конструкция

**U1.text.ToString()**,

а для вычислений, использующих записи в текстовых объектах, например, с именем **name1** используется конструкция

**int.Parse(U1.text).**

1. Программирование обработки и расчетов данных, занесенных в таблицу наблюдений

**using System.Collections;  
using System.Collections.Generic;  
using UnityEngine;  
using UnityEngine.UI;  
  
public class TableVal : MonoBehaviour  
{  
//определение используемых переменных**

**[SerializeField]  
    InputField textInput;  
  
    [SerializeField]  
    Text u1;  
    [SerializeField]  
    Text i1;  
 …………**

**public void WriteValue() //функция для кнопки «Записать»  
    {  
        if (u1.text == "-")  
            u1.text = textInput.text.ToString ();  
        else if (i1.text == "-") {  
            i1.text = textInput.text.ToString ();  
            v1.text = (int.Parse (u1.text) \* int.Parse (i1.text)).ToString ();  
        }**

**………………**

**………………   
        textInput.text="";  
    }  
//функция для кнопки «Очистить»  
    public void Clean()  
    {  
        u1.text = "-";**

**…………  
  
    }  
  
}**

Для записи в текстовые объекты (ячейки таблицы) значений используется, например, для текстового объекта с именем **name1** конструкция

**U1.text.ToString()**,

а для вычислений, использующих записи в текстовых объектах, например, с именем **name1** используется конструкция

**int.Parse(U1.text).**

1. Как создать файл анимации в Unity

Для создания анимации в редакторе **Unity** нужно: выбрать в окне проекта **Project** папку **Assets**, выбрать объект, который будет анимироваться, открыть для него вкладку **Window/Animation/Animation,** задать в открывающемся окне имя для файла анимации с типом **\*.anim** и сохранить его в папке проекта **Assets**.

После сохранения файла анимации в нем создается анимация объекта. На временной шкале в нижней части открывшегося окна редактора **Unity** нажать кнопку с красным кружком (на рисунке) и в соответствующих ключевых кадрах записать требуемую анимацию для объекта.

1. Как добавить к объекту контроллер анимации

Затем нужно добавить к объекту, контроллер анимации **Animator** – кликнуть правой кнопкой мыши по свободной области в папке **Assets**, выполнить команду **Create->Animator Controller** и дать имя контроллеру. Дважды кликнув на созданный контроллер, перейти в окно редактора.

В начале есть два блока, это **Entry**, с которого будут начинать проигрываться анимации и **Any State** – особое состояние, которое всегда существует и позволяет перейти от любого состояния к конкретному.

После создания в самом обьекте необходимо добавить компоненту **Animator** и привязать необходимый к свойству **Controller** либо перенести мышкой в **Inspector** обьекта созданный Animator.

1. Как создать и настроить переход к анимации

В начале есть два блока, это **Entry**, с которого будут начинать проигрываться анимации и **Any State** – особое состояние, которое всегда существует и позволяет перейти от любого состояния к конкретному

Создать **Empty State** как состояние по умолчанию, щелкнув правой кнопкой мыши в свободной части окна аниматора и выполнив команду **Create State/Empty,** в качестве состояния покоя у двери и добавить созданную анимацию отдельным состоянием.

Переходы к состоянию анимации открытия двери и обратно должны происходить по булевой переменной **Bolean**, причем обратно - по условию установки режима **Has Exit Time**. При этом после создания анимации двери необходимо *снять зацикливание* в контроллере анимации.

В окне аниматора **создать переменную** открыввкладку **Parameters,** нажать плюс (добавить), выбрать тип создаваемой переменной, затем дать имя переменной, например **run**.

После этого выбрать в окне редактора щелчком мыши переход от анимации ожидания к анимации движения. Затем в Инспекторе для перехода в списке **Conditions** и добавляем в него нашу переменную **run** со значением **true**, нажав кнопку **+**.

Нажать правой кнопкой мыши на анимацию ходьбы и выбрать **Make transition**, чтобы создать обратную связь на анимацию по умолчанию. Выбрать новый переход и опять снять галочку **Has exit time**. Затем в список **Conditions** добавляем нашу переменную **run**, но со значением **false**.

После этого создать скрипт который будет менять состояние анимации при помощи SetBool

1. Как задать скорость анимации

Откройте окно Animator.

Выберите нужный анимационный клип.

В инспекторе измените значение параметра Speed. Значение 1.0 соответствует нормальной скорости. Увеличьте его, чтобы ускорить анимацию, или уменьшите для замедления.

1. Как настроить файл анимации

Для создания анимации в редакторе **Unity** нужно: выбрать в окне проекта **Project** папку **Assets**, выбрать объект, который будет анимироваться, открыть для него вкладку **Window/Animation/Animation,** задать в открывающемся окне имя для файла анимации с типом **\*.anim** и сохранить его в папке проекта **Assets**.

После сохранения файла анимации в нем создается анимация объекта. На временной шкале в нижней части открывшегося окна редактора **Unity** нажать кнопку с красным кружком (на рисунке) и в соответствующих ключевых кадрах записать требуемую анимацию для объекта.

* Выберите файл в Assets.
* В инспекторе откроется вкладка с настройками импорта.
* На вкладке Animations вы можете редактировать:

Loop Time: зациклить анимацию.

Loop Pose: использовать движения корня.

Cycle Offset: добавить события в анимацию.

1. Как создать и использовать переменные для управления анимацией

Для программного управления анимацией непосредственно в окне редактора необходимо создать программный код на языке C# и, прежде всего, в окне аниматора **создать переменную** открыввкладку **Parameters,** нажать плюс (добавить), выбрать тип ( bool , trigger , int , float) создаваемой переменной, затем дать имя переменной, например **run**

После этого выбрать в окне редактора щелчком мыши переход от анимации ожидания к анимации движения. Затем в Инспекторе для перехода в списке Conditions и добавляем в него нашу переменную run со значением true, нажав кнопку +.

Нажать правой кнопкой мыши на анимацию ходьбы и выбрать Make transition, чтобы создать обратную связь на анимацию по умолчанию. Выбрать новый переход и опять снять галочку Has exit time. Затем в список Conditions добавляем нашу переменную run, но со значением false.

1. Скрипт для переключения между состояниями анимируемого объекта

**// Объявляем переменную для хранения компонента Animator**

**Animator anim;**

**void Start()**

**{**

**// Получаем компонент Animator, прикрепленный к объекту**

**anim = GetComponent<Animator>();**

**}**

**void Update()**

**{**

**// Проверяем, нажата ли клавиша Q**

**if (Input.GetKeyDown(KeyCode.Q))**

**{**

**// Устанавливаем параметр "run" в true для начала анимации**

**anim.SetBool("run", true);**

**}**

**// Проверяем, нажата ли клавиша W**

**if (Input.GetKeyDown(KeyCode.W))**

**{**

**// Устанавливаем параметр "run" в false для остановки анимации**

**anim.SetBool("run", false);**

**}**

**}**

1. Как проиграть анимацию из любого состояния

Чтобы проиграть какую-либо анимацию можно было из любого состояния, необходимо:

1. создать переход от **Any State** к этой анимации, например с именем **Hit**. Для возвращения в состояние ожидания необходимо создать переход от новой анимации **Hit** к анимации по умолчанию.
2. создать еще одну переменную типа **Trigger** (например **hitten**), которая будет отвечать за щелчок мышью для запуска анимации. Добавим к переходу от **Any State** эту переменную, а для обратного перехода оставим галочку **Has Exit Time**, чтобы переход совершался автоматически (без использования переменной) один раз после проигрывания анимации.
3. добавить в метод **Update()** код проигрывания новой пользовательской анимации по клику мыши:

**if (Input.GetMouseButtonDown(0))  {  anim.SetTrigger("hitten"); }**