**КОНТРОЛЬНЫЕ ВОПРОСЫ**

**ПО ТЕМЕ «ГУРИН ПИДОРАС»**

***(все строки программного кода должны быть отхуячены и забыты)***

1. **Программа для обхода камерой вокруг установки**

|  |
| --- |
| [SerializeField] // Позволяет редактировать переменные в инспекторе Unity  Transform targetPos; // Целевая позиция, вокруг которой будет вращаться камера  [SerializeField]  int sensitivity = 3; // Чувствительность вращения камеры  void Update() // Метод, вызываемый каждый кадр  {  // Проверяем, удерживается ли правая кнопка мыши  if (Input.GetMouseButton(1))  {  // Получаем движение мыши по оси X, умножаем на чувствительность  float y = Input.GetAxis("Mouse X") \* sensitivity;  // Если движение мыши не равно нулю, вращаем камеру вокруг целевой позиции  if (y != 0)  {  transform.RotateAround(targetPos.position, Vector3.up, y); // Вращаем камеру вокруг целевой позиции по оси Y  }  } |

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

|  |
| --- |
| [SerializeField]  float scrollSpeed = 10f; // Скорость приближения/удаления камеры  [SerializeField]  int maxDistance = 20; // Максимальное расстояние, на которое может удалиться камера  [SerializeField]  int minDistance = 1; // Минимальное расстояние, на которое может приблизиться камера  // Получаем движение по горизонтали (клавиши A и D)  float x = Input.GetAxis("Horizontal") / sensitivity;  **float y = Input.GetAxis("Vertical"); // Получение ввода для вертикального движения**  **if (x != 0 || y != 0)** {  // Вычисляем новую позицию камеры с учетом текущего направления  Vector3 newPos = transform.position + transform.TransformDirection(new Vector3(x, 0, 0) +Vector3.up \* y);  // Проверяем, находится ли новая позиция в допустимых пределах  if (ControlDistance(Vector3.Distance(newPos, targetPos.position)))  {  transform.position = newPos; // Обновляем позицию камеры  }  }  // Метод для проверки, находится ли расстояние в допустимых пределах  bool ControlDistance(float distance)  {  // Возвращает true, если расстояние больше минимального и меньше максимального  return distance > minDistance && distance < maxDistance;  } |

1. **Программа приближения и удаления камеры относительно установки**

|  |
| --- |
| // Получаем прокрутку мыши  float z = Input.GetAxis("Mouse ScrollWheel") \* scrollSpeed;  if (z != 0) // Если прокрутка не равна нулю  {  // Вычисляем новую позицию камеры для приближения или удаления  Vector3 newPos = transform.position + transform.TransformDirection(Vector3.forward \* z);  // Проверяем, находится ли новая позиция в допустимых пределах  if (ControlDistance(Vector3.Distance(newPos, targetPos.position)))  {  transform.position = newPos; // Обновляем позицию камеры  }  } |

1. **Правила создания и размещения на экране элементов интерфейса CANVAS**

Холст - 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.

Привязка (Anchor) объекта - 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 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.

Точки привязки позволяют подстраиваться не только под изменение положения, но и под изменение размера.

Привязку можно отредактировать таким образом, что при увеличении размера экрана изображения будут растягиваться вместе с ним.

Способ наложения элементов UI друг на друга определяется порядком их следования на вкладке Hierarchy. Чтобы расположить всплывающее окно поверх остальных элементов UI необходимо перетащить окно в самый низ иерархического списка дочерних элементов холста, чтобы оно отображалось поверх всего остального.

1. **Принцип обработки щелчка мышью по кнопке.**

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.

КОД ДЛЯ ВЫВОДА СООБЩЕНИЯ В ТЕКСТОВОЕ ОКНО ПРИ ЩЕЛЧКЕ КУРСОРОМ МЫШИ ПО КНОПКЕ:

**using UnityEngine;**

**using UnityEngine;**

**using UnityEngine.UI;**

**public class InText : MonoBehaviour {**

**[SerializeField]**

**Text message;**

**public void OnPointer () {** // МЕТОД, ВЫЗЫВАЕМЫЙ ПОЛЬЗОВАТЕЛЕМ ЩЕЛЧКОМ ПО КНОПКЕ

**message.text="Установка начальных значений";**

**}**

**}**

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**.

1. **Принцип обработки надвижения и ухода курсора мыши с кнопки.**

По умолчанию у кнопки доступно событие щелчок мышью - Click(), для использования других событий, например, надвижение курсора мыши PointerEnter(), необходимо к кнопке добавить компоненту Event/Event Trigger и затем выбрать из предложенного списка нужное событие.

**public void OnExit() {**// МЕТОД, ВЫЗЫВАЕМЫЙ ПОЛЬЗОВАТЕЛЕМ ПРИ УХОДЕ КУРСОРА МЫШИ С КНОПКИ **message.text = "Симулятор предназначен для проведения лабораторного практикума в виртуальном режиме с установкой, представленной на экране компьютера";   
    }**

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**

**}**

**}**

**}**

**}**

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

|  |
| --- |
| **using System.Collections.Generic; using UnityEngine;  public class ChangeColor : MonoBehaviour {          public void ChangCol() //на красный     {         GetComponent<Renderer> ().material.color = new Color (1, 0, 0);        }      public void ChangCol1()// на белый        {         GetComponent<Renderer> ().material.color = new Color (1, 1, 1);    } }** |

Затем необходимо повторить процедуру вызова из скрипта функций **ChangCol()** и **ChangCol1()** соответственно для событий наведения курсора мыши на кнопку **Pointer Enter** и **Pointer Exit.**

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

Создать UI-объект – окно, которое должно всплывать и расположить его на CANVAS.

Задать следующий скрипт для всплывающего окна.

**using UnityEngine;**

**using System.Collections;**

**public class SettingsPopup : MonoBehaviour {**

**void Start() {    
        Close();**//  ЗАКРЫТЬ ВСПЛЫВАЮЩЕЕ ОКНО ПРИ ЗАПУСКЕ ПРОГРАММЫ**}   
  
    public void Open() {  
        gameObject.SetActive(true);**//   АКТИВИРОВАТЬ ОБЪЕКТ, ЧТОБЫ ОТКРЫТЬ ОКНО.**}    
     public void Close() {  
        gameObject.SetActive(false);**// ДЕАКТИВИРОВАТЬ ОБЪЕКТ, ЧТОБЫ ЗАКРЫТЬ ОКНО.**}**

**}**

Создать на CANVAS две кнопки, соответственно для открытия и закрытия окна при щелчке по кнопкам мышью.

Назначить кнопкам общий объект обработки – всплывающее окно и соответственно выбрать в связанном с окном скрипте **SettingsPopup** в списке доступных в скрипте функций функцию **Open()**для кнопки открытия окна и **Close()**для кнопки закрытия окна.

1. **Настройка проекта для обработки щелчков мышью по 3D-объектам сцены.**

необходимо добавить скрипт на этот объект с добавлением в скрипт директивы *using UnityEngine.EventSystems* и класса *IpointerClickHandler* к базовому классу *MonoBehaviour.*

1. **Правила создания таблицы результатов эксперимента на симуляторе установки.**

Таблица состоит из массива текстовых полей, в которые записываются результа-ты работы с симулятором.

Ячейки таблицы заполняются последовательно частично вручную, частично ав-томатически по формулам.

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.

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

В ходе выполнения лабораторной работы снятые измерения с установки записы-ваются в нужные ячейки таблицы по нажатию кнопки «Записать», для просмотра со-держания 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» выво-дится соответствующее сообщение о завершении работы с симулятором установки.

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

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

**name1.text.ToString()**,

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

**int.Parse(name1.text).**

**using System.Collections;  
using System.Collections.Generic;  
using UnityEngine;  
using UnityEngine.UI;  
  
public class TableVal : MonoBehaviour  
{**

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

**public void WriteValue()     {  
        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="";  
    }  
    public void Clean()  
    {  
        u1.text = "-";  
  
    }  
}**

1. **Как создать файл анимации в Unity.**

выбрать в окне проекта 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. На временной шкале в нижней части открывшегося окна редактора Unity нажать кнопку с красным кружком и в соответствующих ключевых кадрах записать требуемую анимацию для объекта.

1. **Как добавить к объекту контроллер анимации**

кликнуть пкм по свободной области в папке Assets, выполнить команду Create->Animator Controller и дать имя контроллеру. Дважды кликнув на созданный контроллер, перейти в окно редактора. 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 пкм 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 пкм 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(гпт)**

* 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.

В окне Animation вы можете добавлять ключевые кадры, изменять свойства объектов (например, позицию, вращение, масштаб) и настраивать интерполяцию между ключевыми кадрами.

1. **Как создать и использовать переменные для управления анимацией**

в окне редактора необходимо создать в окне аниматора **переменную** открыввкладку **Parameters,** нажать плюс, выбрать тип переменной, затем дать имя.

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 со значением true, нажав кнопку +.

Нажать пкм 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, но с false.

1. **Скрипт для переключения между состояниями анимируемого объекта**

**Animator anim;** //переменная типа **Animator** для ссылки на анимацию

**void Start()**

**{**

**anim = GetComponent<Animator>();** //инициализация контроллера анимации

**}**

**void Update()**

**{**

**If(Input.GetKeyDown(KeyCode.Q))** //если нажата клавиша **q**

**{**

**anim.SetBool(“run”, true);** // устанавливаем параметр в тру для начала анимации

**}**

**If(Input.GetKeyDown(KeyCode.W))** //если нажата клавиша **w**

**{**

**anim.SetBool(“run”, false);** // устанавливаем параметр в тру для остановки анимации **}**

**}**

1. **Как проиграть анимацию из любого состояния**
2. 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.
3. 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.
4. добавить в метод **Update()** код проигрывания новой пользовательской анимации по клику мыши:

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