Разработка комплекса учебных проектов по созданию приложений с графическим интерфейсом на платформе Unity.

**Введение**

Программный интерфейс не только решает проблему взаимодействия с приложением, но и делает это взаимодействие максимально комфортным. Интерфейс необходим для удобного взаимодействия пользователя с приложением.

Поэтапная разработка интерфейса приложений и сложных систем экономит время и структурирует работу. Очень важно уметь разработать интерфейс, в котором пользователь найдет ключевые функции программы за минимальное время. Время, затрачиваемое на разработку, и количество поддерживаемых платформ также является важным фактором.

Кроссплатформенная разработка позволяет создавать приложения сразу для нескольких платформ. Кроссплатформенное приложение требует гораздо меньше ресурсов, позволяет тратить меньше времени на разработку. Unity – это мультиплатформенный инструмент для разработки 2D и 3D приложений. Основными преимуществами Unity являются наличие визуальной среды разработки, межплатформенной поддержки и модульной системы компонентов. Созданные с помощью Unity приложения работают под операционными системами Windows, Linux, OS X, Android, IOS.

Современные приложения часто нуждаются в поддержке широкого спектра различных разрешений экрана, и особенно в данной возможности нуждаются их интерфейсы. Адаптивный дизайн является необходимым элементом кроссплатформенного приложения. Причиной этому служит большое количество различных устройств и платформ, которые отличаются разными разрешениями. Адаптивная версия должна работать на всех платформах с любым разрешением. Это позволяет потратить меньше усилий для реализации программы под смартфоны, планшеты, ноутбуки, ПК.

Система создания интерфейсов в Unity снабжена рядом различных инструментов для адаптивного дизайна, которые также можно комбинировать между собой массой различных способов.

Таким образом, платформа Unity является отличным инструментом для кроссплатформенной разработки приложений с пользовательским интерфейсом под широкий набор разрешений экранов на различных устройствах.

**1 Постановка задачи**

Задачей данной работы является разработка комплекса учебных проектов по созданию кроссплатформенных графических приложений с адаптивных дизайном. Демонстрация работы приложений в мобильных ОС на примере Android и настольных ОС на примере Windows.

Данная работа включает в себя:

1. Установка платформы Unity и дополнений для сборки под ОС Android.
2. Знакомство с редактором и основными особенностями разработки пользовательских интерфейсов на Unity.

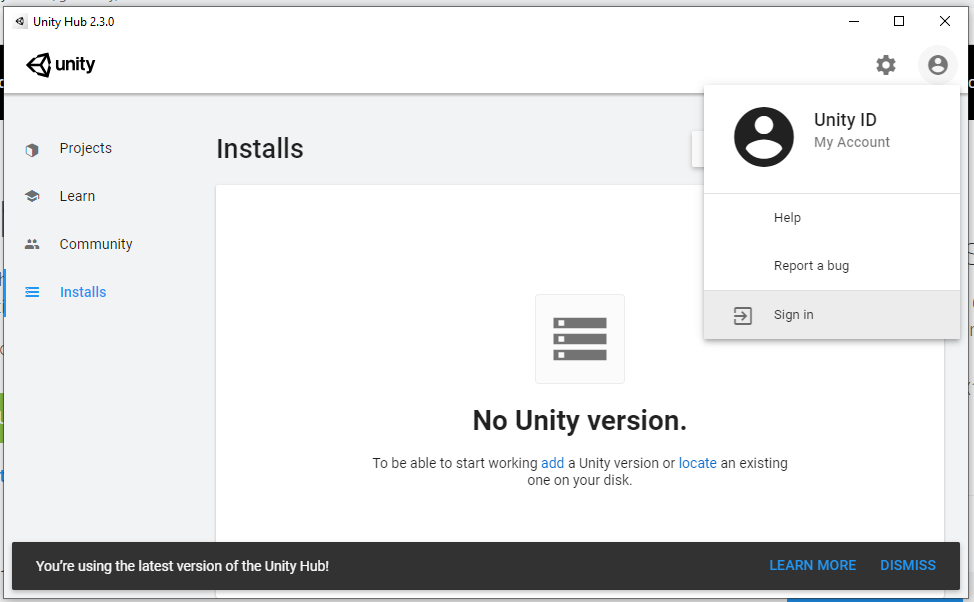
**2 Особенности разработки приложений на платформе Unity**

**2.1 Отличие разработки приложений на Unity от нативной разработки приложения**

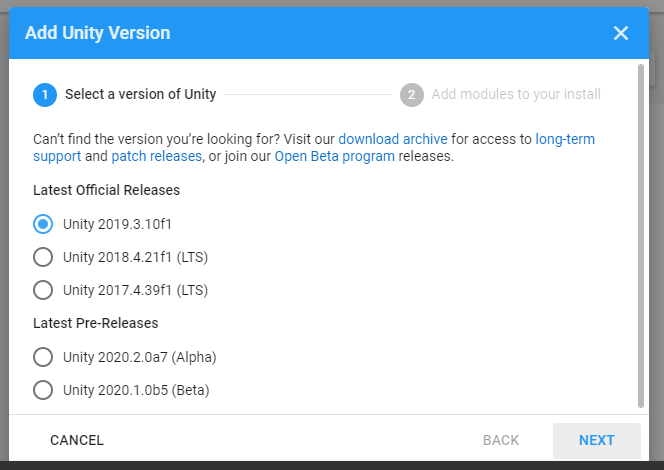
**2.2 Установка и настройка среды разработки**

Для начала необходимо установить Unity Hub по ссылке <https://unity3d.com/get-unity/download>. Он позволяет управлять проектами, работать с различными версиями Unity и устанавливать дополнения для сборки под различные платформы.

Для работы с редактором необходимо создать Unity ID.



Далее на вкладке Installs нажать кнопку Add и установить последнюю стабильную версию Unity.

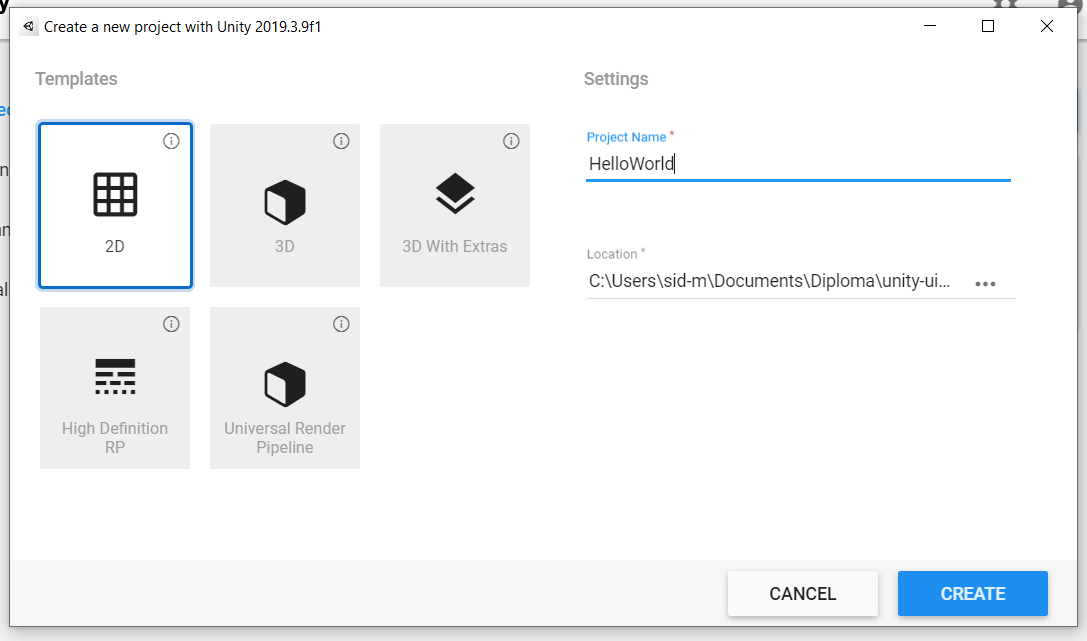


На следующем шаге поставить галочку напротив опции Android Build Support.

**3. Обзор среды разработки. Создание проекта**

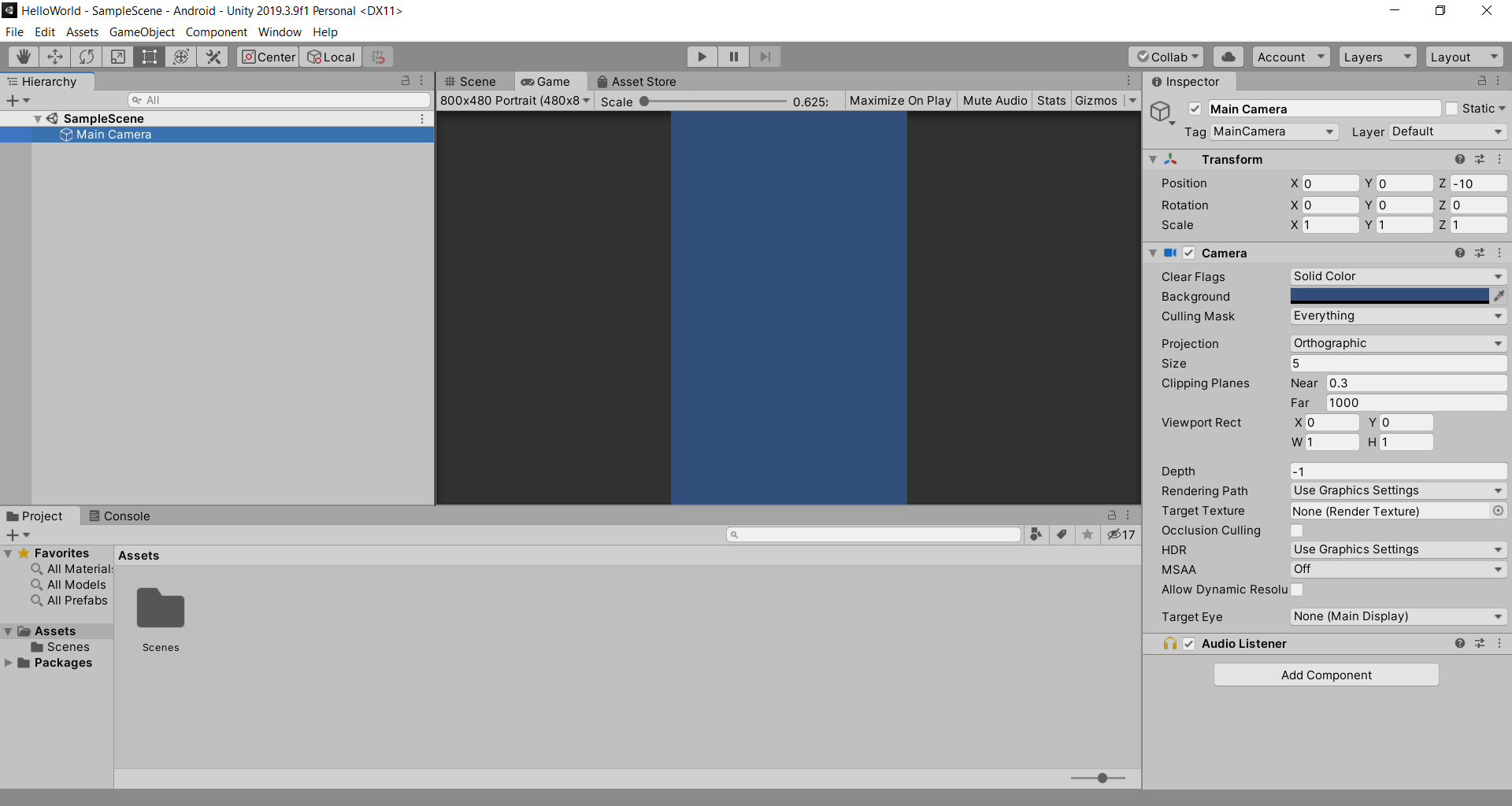
**3.1 Создание проекта**

Зайти в Unity Hubs, во вкладке projects нажать на кнопку New. Далее выбрать тип проекта 2D, указать название проекта и выбрать место его расположения.

****

В первую очередь в верхнем меню File, далее Build Settings, изменим текущую платформу на Android. Этот шаг был выполнен с целью добавления большего количества разрешений экранов в предпросмотре редактора.

Рассмотрим основные области редактора Unity. По центру располагается вкладка Game. Здесь есть выпадающий список для изменения разрешения. Вверху кнопка запуска симуляции работы приложения в редакторе. Также здесь есть вкладка Scene, в которой можно посмотреть отображение UI элементов. Для перемещения по сцене необходимо выбирать инструмент Hand Tool в левом верхнем углу. Для приближения и отдаления можно использовать колесико мыши. Снизу располагается вкладки консоль, куда можно выводить отладочную информацию. А также вкладка Project со структурой проекта. В папке Assets есть папка Scenes, где хранится автоматически созданная сцена. В левой области располагается вкладка Hierarchy, где располагаются сцены и их объекты. Изначально у сцены есть только объект камеры. Во вкладке Inspector, расположенной справа осуществляется редактирование свойств объектов.

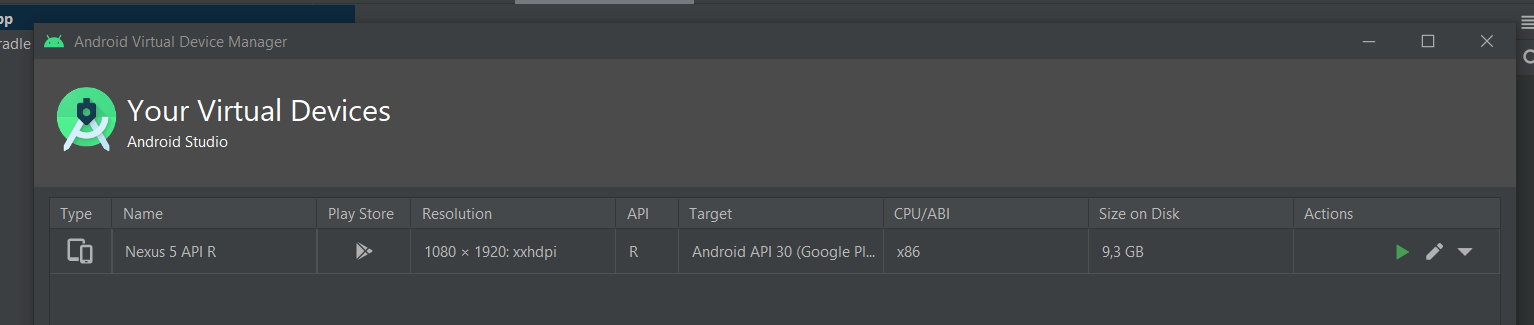


Изменим Background камеры на значение CCCCCC. Для начала на сцене нужно создать элемент Canvas, так как все UI элементы должны быть дочерними к элементу Canvas. Для этого нужно во вкладке Hierarchy нажать правой кнопкой по области под Main Camera и выбрать UI, далее Canvas. Также автоматически будет создан объект EventSystem, отвечающий за обработку пользовательского ввода.

Render Mode для Canvas стоит Screen Space – Overlay. Элементы отображаются поверх сцены, а если меняется размер экрана или его разрешение, то Canvas автоматически примет нужный размер вместе с ним. У Canvas в Inspector установим Pixel Perfect чекбокс. Включение Pixel Perfect может сделать элементы более четкими и предотвратить размытость. Следующим шагом будет добавление элемента текста внутрь Canvas. Для этого во вкладке Hierarchy нажать правой кнопкой по Canvas > UI > Text. Выбрать элемент Text во вкладке Hierarchy, далее во вкладке Inspector отредактировать значение Text на Hello World, установить параметр Alignment по центру и установить чек бокс Best Fit для автоматической подстройки размера шрифта. Элемент текст располагается в центре элемента Canvas так как Rect Transforms > Anchor Presets установлены значения middle center. Anchor Presets задает расположение элемента относительно своего родителя отдельно по горизонтали и вертикали. Также Anchor Presets позволяет растянуть элемент на всю ширину и высоту родительского элемента.

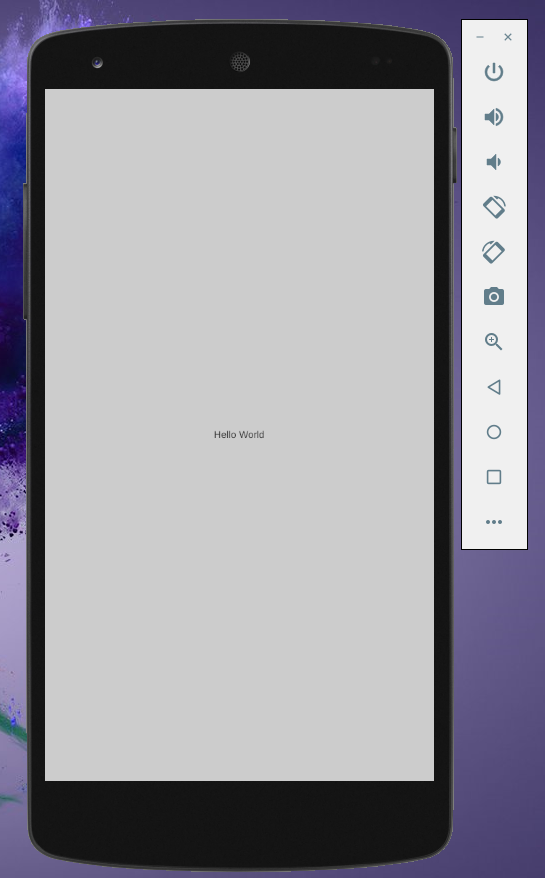
Далее запустим симуляцию нажав на кнопку Play и посмотрим, как будет выглядеть текст при различных разрешения экрана.

Осуществим сборку и запуск приложения на Android эмуляторе или настоящем Android устройстве. Если у вас установлена Android Studio, в ней можно создать и запустить эмулятор в AVD Manager.



Также можно подключить Android устройство с включенным режимом разработчика и отладки по USB.

Для сборки и запуска нажать File > Build And Run, указать название .apk. Обратите внимание, что если в пути к проекту присутствует кириллица во время сборки возникнет ошибка.



Далее поменяем тип сборки File > Build Settings изменим платформу на PC, Mac & Linux Standalone. Target Platform выбираем систему, в которой вы работаете. Галочку Create Visual Studio Solution можно снять. Открыть Player Settings. Вкладка Settings For PC > Resolution And Presentation. Fullscreen Mode выбрать Windowed, Default Screen Width 800, Default Screen Height 600, Resizable Window выбрать чек бокс. Закрыть настройки, нажать Build And Run.

Если вы, например, работаете под Windows и у вас появится необходимость собрать приложение под Mac или Linux. Для этого в Unity Hub > Install > три точки по версии Unity > Add Modules. Выбрать опции Linux Build Support, Mac Build Support и кнопка Done. Дождаться окончания установки. Изменить Target Platform в Build Settings.

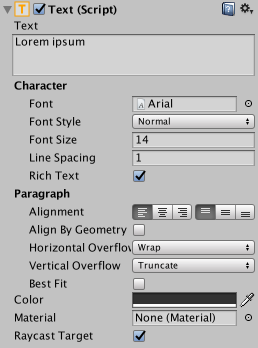
**3.2 Начальные сведения для начала разработки**

**3.2.1 Элементы пользовательского интерфейса.**

Canvas – это область, внутри которой находятся все элементы UI. Когда создаётся новый элемент UI вместе с ним автоматически создается и Canvas, если до этого на сцене его еще не было. Canvas использует объект EventSystem, чтобы использовать систему обмена сообщениями и обрабатывать события пользовательского ввода. Элементы UI на Canvas появляются в том же порядке, в каком они расположены в иерархии. Первый дочерний элемент отрисовывается первым, второй – за ним и так далее. Если два элемента UI накладываются друг на друга, добавленный позднее будет поверх того, что был добавлен ранее. Чтобы изменить то, какой элемент будет находиться поверх остальных, просто поменяйте местами элементы в иерархии путем перетаскивания. Порядком также можно управлять при помощи кода, используя следующие методы компонента Transform: SetAsFirstSibling, SetAsLastSibling и SetSiblingIndex.

Рассмотрим основные компоненты UI, которые призваны упростить создание и использование характерных функции GUI:

## Text



Компонент **Text**, также известный как Label, имеет область для ввода текста, который будет отображен. Есть возможность задать шрифт, его стиль, размер и виды отображения.

## Button



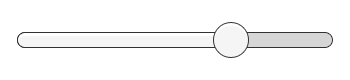
Кнопка предназначена для запуска действия, когда пользователь щелкает и отпускает её. Кнопка имеет одно событие OnClick, которое реагирует, когда пользователь завершает щелчок.

**Toggle**



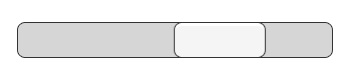
Элемент управления Toggle позволяет пользователю включать или отключать опцию. Также можно объединить несколько переключателей в ToggleGroup в случаях, когда одновременно должен быть включен только один из набора параметров. Toggle имеет одно событие под названием *OnValueChanged*.

## Slider



Slider имеет Value - десятичное число, которое пользователь может устанавливать между минимальным и максимальным значением. При изменении значения вызывается событие *OnValueChanged*.

**Scrollbar**



Полосы прокрутки используются вместе с другими компонентами для создания эффекта прокрутки. Полоса прокрутки имеет Size – диапазон от 0 до 1, оно определяет значение прокрутки. При изменении значения вызывается событие *OnValueChanged*.

## Dropdown



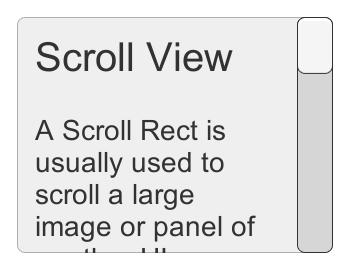
Раскрывающийся список позволяет выбрать одну опцию из списка опций. При изменении значения вызывается событие *OnValueChanged*.

**Input Field**



Поле ввода используется, чтобы сделать текст элемента Text редактируемым пользователем.

**Scroll Rect**



Scroll Rect может использоваться, когда контент, занимающий много места, должен отображаться в небольшой области.

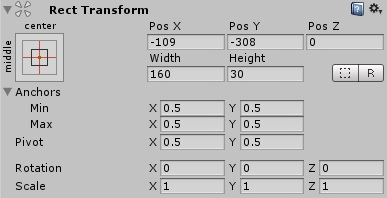
**Image**

Источник может быть добавлен к Image в поле Source Image, а его цвет задан в поле Color. Также к компоненту Image может быть добавлен материал.

## 3.2.2 Расположение элементов пользовательского интерфейса

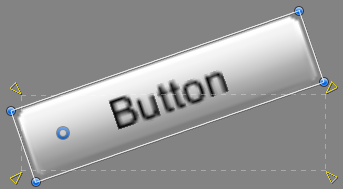
## Rect Transform

Rect Transform – это компонент преобразования, который используется для всех элементов пользовательского интерфейса.



Rect Transform имеет значения положения, вращения и масштаба, а также ширину и высоту, используемые для указания размеров области.

### Pivot



Pivot - точка, относительно которой происходят преобразования при повороте, изменении размера и масштабировании.

**Anchors**

Anchors отображаются в виде четырех маленьких треугольных ручек в окне просмотра сцены. Они могут быть привязаны к любой точке внутри родительского прямоугольника. Закрепление позволяет ребенку растягиваться вместе с шириной или высотой родителя. Каждый угол прямоугольника имеет фиксированное смещение относительно соответствующей привязки. Верхний левый угол прямоугольника имеет фиксированное смещение относительно верхнего левого якоря и т. д. Таким образом, различные углы прямоугольника могут быть привязаны к различным точкам в родительский прямоугольник.

Нажатие на Anchor Presets вызывает выпадающий список. Отсюда вы можете быстро выбрать один из самых распространенных вариантов привязки. Вы можете привязать элемент пользовательского интерфейса к сторонам или середине родительского элемента или растянуть вместе с родительским размером. Горизонтальное и вертикальное закрепление является независимым.



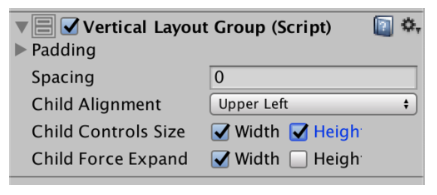
Система автоматической разметки предоставляет способы размещения элементов во вложенных группах разметки, таких как горизонтальные группы, вертикальные группы или сетки. Это также позволяет автоматически выбирать размеры элементов в соответствии с содержимым. Например, кнопка может быть динамически изменена, чтобы точно соответствовать ее текстовому содержимому плюс некоторый отступ.

# **Auto Layout**

Система автоматической компоновки на основе концепции элементов макета и контроллеры макета. Элементом макета является игровой объект с трансформацией Rect, а также, при необходимости, другие компоненты. Элемент макета имеет определенные знания о том, какой размер он должен иметь. Элементы макета непосредственно не устанавливают свой собственный размер, но другие компоненты, которые функционируют как контроллеры макета, могут использовать информацию, которую они предоставляют, чтобы вычислить размер, который будет использоваться для них.

Элемент макета имеет свойства, которые определяют его собственные: Minimum width, Minimum height, Preferred width, Preferred height, Flexible width, Flexible height.

Примерами компонентов контроллера макета, которые используют информацию, предоставляемую элементами макета, являются Content Size Fitter и различные компоненты Layout Group.



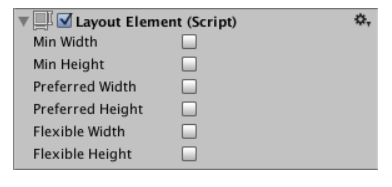
Основные принципы определения размера элементов макета в группе макетов следующие:

Первые minimum sizes выделяются.

Если свободного места достаточно, выделяются preferred sizes.

Если есть дополнительное доступное пространство, выделяется flexible size.

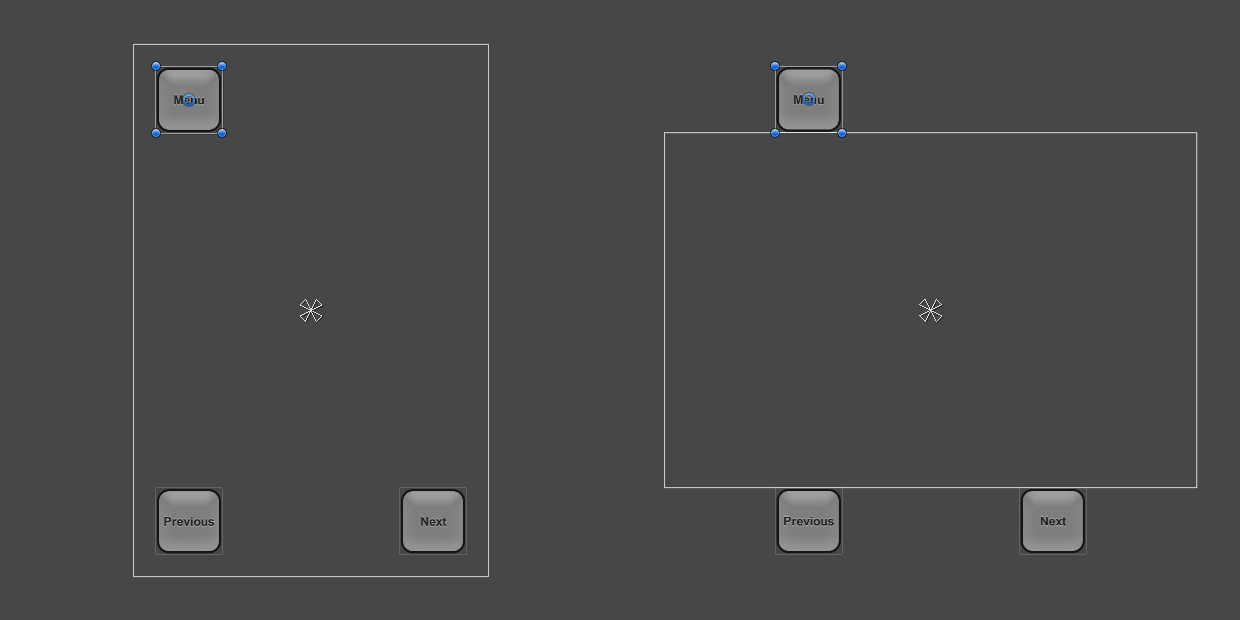
Если вы хотите переопределить minimum, preferred или flexible size у Layout Element, вы можете сделать это, добавив компонент Layout Element к Game Object.



**3.2.3 Адаптивный дизайн**

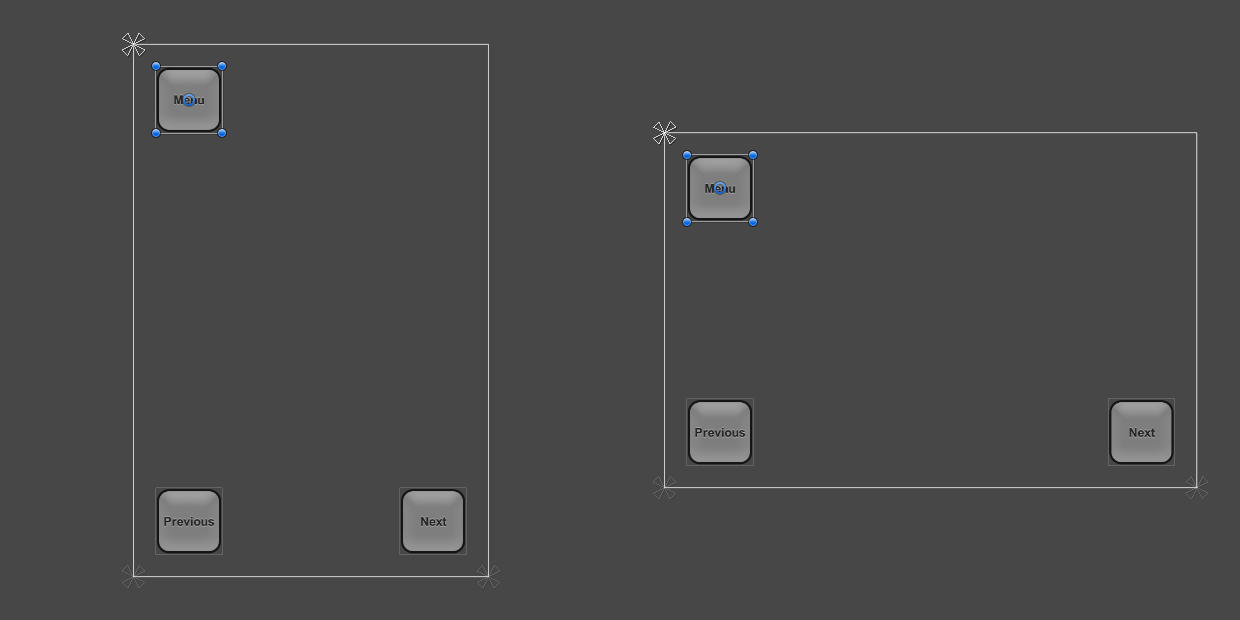
Элементы интерфейса по умолчанию привязаны к центру родительского прямоугольника. Это означает, что они сохраняют постоянное смещение относительно центра.

Если с данной настройкой разрешение было изменено на альбомное соотношение сторон, кнопки могут выпасть из своих прямоугольных областей, где они изначально должны располагаться.

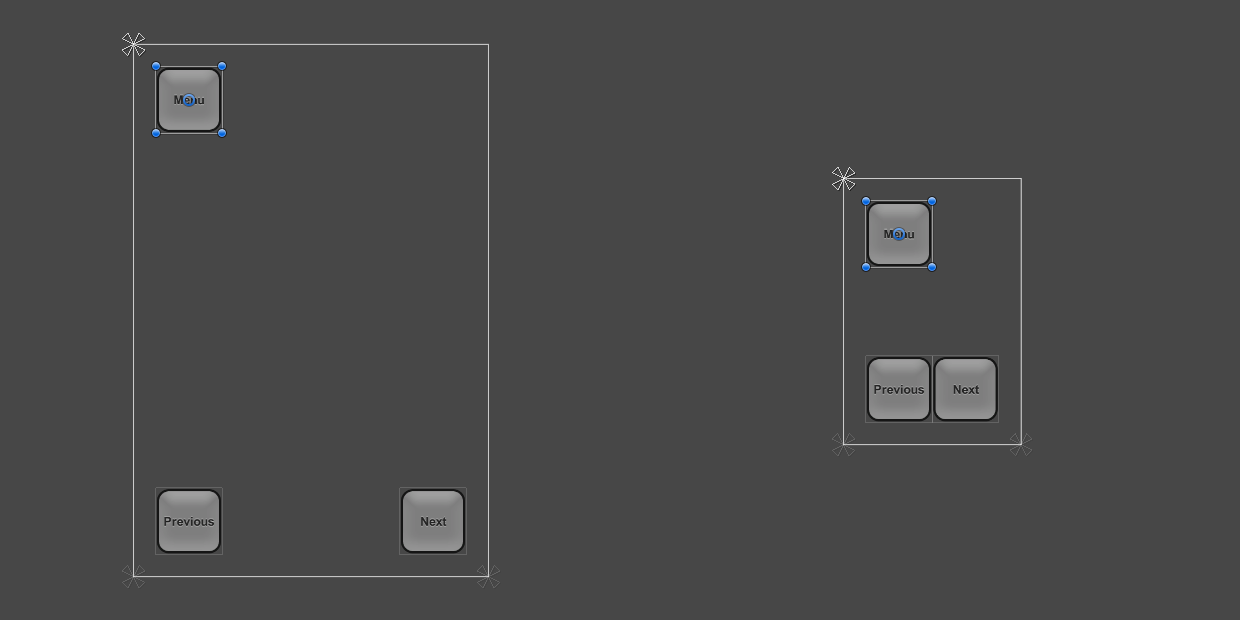


Одним из способов сохранить расположение кнопок в области экрана является изменение компоновки таким образом, чтобы места их расположения были связаны с соответствующими углами на экране. Привязка левой верхней кнопки может быть установлена в левом верхнем углу при использовании в инспекторе выпадающего списка Anchors Preset (наборы привязок), или путём перетаскивания треугольных ручек привязок в видовом окне сцены (Scene View). Аналогично привязки для левой нижней и правой нижней кнопок могут быть выставлены в левый нижний и правый нижний угол соответственно.

После привязки кнопок к своим углам, при дальнейших изменениях разрешений экрана и соотношений сторон будут сохраняться позиции кнопок относительно углов.

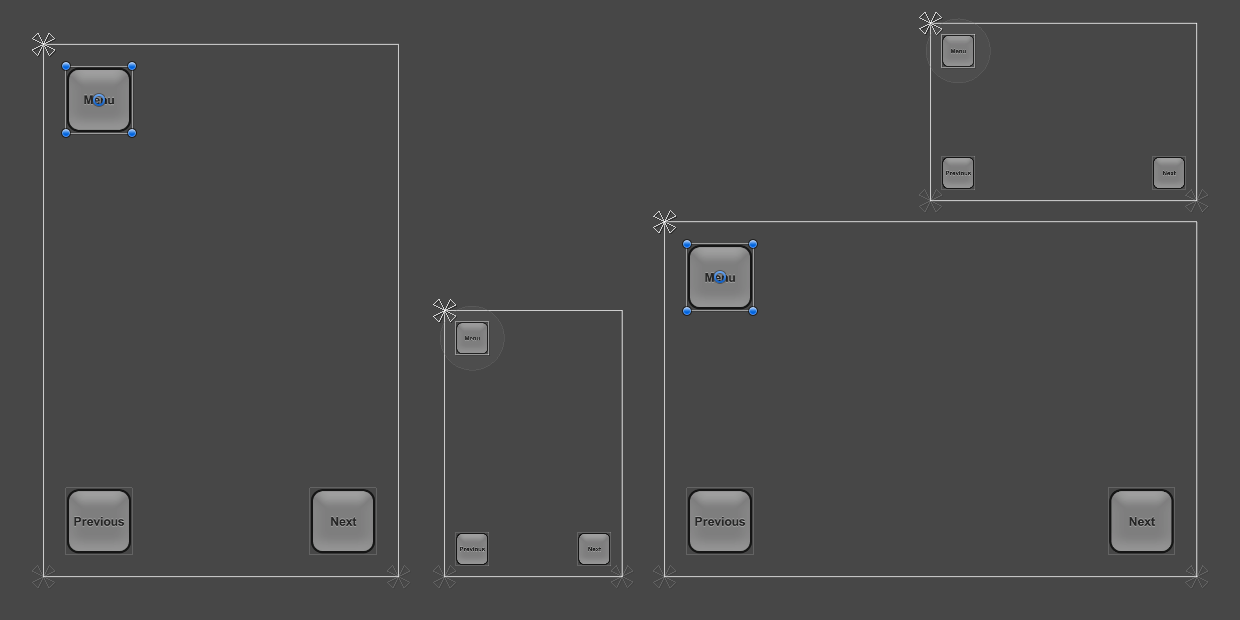


Когда разрешение экрана изменяется на большее или меньшее относительно текущего, кнопки должны по-прежнему сохранять своё изначальное расположение относительно углов, к которым они привязаны. Однако, сохраняя своё оригинальное разрешение, заданное в пикселях, они могут становиться как больше, так и меньше, соответствуя пропорциям текущего разрешения экрана. Кнопки должны становиться меньше настолько же, насколько в процентном соотношении становиться меньше сам экран.



Для реализации масштабирования компонент Canvas Scaler может быть добавлен в корень Canvas. В компоненте Canvas Scaler можно установить его UI Scale Mode в Scale With Screen Size. В данном режиме масштабирования можно определить какое разрешение использовать в качестве базового. Если текущее разрешение больше или меньше базового, фактор масштабирования компонента Canvas устанавливается соответственно так, чтобы все элементы интерфейса масштабировались в большую или меньшую сторону вместе с разрешением экрана.

После данной настройки компоновка должна будет масштабироваться пропорционально размерам экрана. При этом уменьшаться размеры кнопок, их расстояние от краёв экрана, графическая составляющая и текстовые элементы. Это означает, что компоновка, которая была в портретном разрешении отобразиться на экране также как и при альбомном разрешении, лишь с разницей в плотности пикселей.



**Проект 1: Events**

Темы, рассматриваемые в проекте:

* создание обработчиков событий
* использование UI элемента Button
* добавление и удаление обработчиков событий

Цель проекта: создать приложение, в котором есть две кнопки. Кнопка закрыть либо изменяет свое расположение на сцене, либо закрывает приложение. Действие этой кнопки определяется количеством нажатий на кнопку приручить. При нажатии на кнопку приручить меняется обработчик события кнопки закрыть. Также меняется заголовок кнопки приручить с приручить на сброс и наоборот.

В папке Assets создадим две директории Scripts и Scenes. Далее во вкладке Hierarchy выберем Main Camera. Во вкладке Inspector откроются настройки для объекта камеры. Установим Background в серый цвет(#CCCCCC).

Создадим пустой холст. Щёлкнем правой кнопкой по сцене, выберем GameObjects > UI > Canvas. Для только что созданного Canvas в Inspector установим Pixel Perfect чекбокс. В разделе Canvas Scaler UI Scale Mode выберем Scale With Screen Size. Значение Screen Match Mode = Match Width Or Height, Match = 0.5, Reference Resolution установим по x в 800, по у в 600.

Добавим кнопки на сцену. Во вкладке Hierarchy правой кнопкой по Canvas GameObjects > UI > Button. Дадим кнопкам названия Button1 и Button2. Значения Rect Transform для Button1: Pos X = 5, Pos Y = -5, Width = 120, Height = 40. Для вложенного элемента Text в Button2 изменим значение Text на Приручить. Значения Rect Transform для Button1: Pos X = 140, Pos Y = -5, Width = 120, Height = 40. Для вложенного элемента Text в Button2 изменим значение Text на Закрыть.

Создадим обработчики события. Папка Assets > Scripts, правой кнопкой, Create C# Script. При создании скрипта дадим ему название MainController. Добавим этот скрипт к элементу Canvas. Вкладка Inspector, спуститься вниз, нажать на кнопку Add Component, далее Scripts > MainController.

Исходный код MainController.cs:

using UnityEngine;

using UnityEngine.UI;

public class MainController : MonoBehaviour {

bool isWild;

Canvas canvas;

Button btn1;

Button btn2;

private void Start()

{

canvas = GameObject.Find("Canvas").GetComponent<Canvas>();

btn1 = GameObject.Find("Button1").GetComponent<Button>();

btn2 = GameObject.Find("Button2").GetComponent<Button>();

isWild = true;

btn1.onClick.AddListener(ChangeState);

btn2.onClick.AddListener(WildClick);

}

private void OnRectTransformDimensionsChange()

{

Vector3 pos = btn2.transform.localPosition;

Rect btnRect = btn2.GetComponent<RectTransform>().rect;

pos.x = 0 - btnRect.width / 2;

pos.y = 0 - btnRect.height / 2;

btn2.transform.localPosition = pos;

}

public void ChangeState ()

{

Debug.Log("ChangeState");

Text labelBtn1 = btn1.GetComponentInChildren<Text>();

if (isWild)

{

isWild = false;

labelBtn1.text = "Сброс";

btn2.onClick.RemoveListener(WildClick);

btn2.onClick.AddListener(QuitApplication);

}

else

{

isWild = true;

labelBtn1.text = "Приручить";

btn2.onClick.RemoveListener(QuitApplication);

btn2.onClick.AddListener(WildClick);

}

}

public void WildClick()

{

Debug.Log("WildClick");

Vector3 pos = btn2.transform.localPosition;

Rect btnRect = btn2.GetComponent<RectTransform>().rect;

Rect canvasRect = canvas.GetComponent<RectTransform>().rect;

pos.x = Random.Range(-1f, 1f) \* (canvasRect.width / 2 - btnRect.width);

pos.y = Random.Range(-1f, 1f) \* (canvasRect.height / 2 - btnRect.height);

Debug.Log("pos.x = " + pos.x);

Debug.Log("pos.y = " + pos.y);

btn2.transform.localPosition = pos;

}

public void QuitApplication()

{

Debug.Log("QuitApplication");

Application.Quit();

}

}

Подключим пространства имён UnityEngine и UnityEngine.UI. В классе MainControoler создадим приватные переменные:

* isWild типа bool для определения состояния кнопки приручить
* canvas типа Canvas для работы с элементом холста
* btn1 типа Button для работы с кнопкой приручить
* btn2 типа Button для работы с кнопкой закрыть

В приватном методе Start, который вызывается до отрисовки первого кадра, проинициализируем значения переменных и добавим обработчики событий для кнопок. Для поиска UI элемента по его имени используется статический метод класса GameObject с названием Find. После этого осуществляется преобразование типа GameObject к нужному типу при помощи метода GetComponent. Для добавления обработчика события нажатия на кнопку нужно у кнопки обратиться к свойству onClick и вызвать метод AddListener и в качестве аргумента передать ссылку на метод обработчик.

Для кнопки btn1 в качестве обработчика события добавляется метод ChangeState. При его вызове в зависимости от значения флага isWild меняется обработчик события кнопки btn2, а также её заголовок. Для изменения заголовка необходимо работать с вложенным элементом Text в элемент Button. Для его получения используется метод GetComponentInChildren переменной btn1 - осуществляет поиск вложенного элемента с заданным типом. Изменение обработчика события для переменной btn2 осуществляется удалением текущего обработчика и добавлением другого. Для удаления обработчика события у кнопки нужно обратиться к свойству onClick и вызвать метод RemoveListener.

Если для btn2 установлен обработчик QuitApplication, то при его срабатывании происходит вызов статического метода Quit класса Application, что приводит к закрытию приложения. Если для btn2 установлен обработчик WildClick, то кнопка случайно меняет своё расположение на сцене. Через свойство transform кнопки обращаемся к свойству localPosition и получаем вектор координат элемента на сцене. С помощью метода GetComponent получаем элемент RectTransform и его свойство rect у canvas и btn2 для определения их ширины и высоты. Изменяем значения координат вектора положения кнопки на случайное число в допустимом диапазоне размера холста с учётом размера кнопки.

Также у класса MainController был добавлен метод OnRectTransformDimensionsChange, который вызывается при изменении размера холста. Это может произойти, например, при смене ориентации экрана на смартфонах. В таком случае кнопка закрыть может оказаться за пределами сцены и нужно осуществить перерасчёт её координат. В данном случае устанавливается позиция центра сцены.

**Проект 2: Calc**

Темы, рассматриваемые в проекте:

* повторное использование обработчиков событий
* использование и валидация UI элемента InputField
* обработка ввода с клавиатуры в приложении
* использование Horizontal Layout Group
* фокусировка на поле ввода

Цель проекта: создать приложение калькулятор с двумя полями ввода, между которыми отображается текст применяемой операции. Ввод осуществляется с клавиатуры, изменить операцию и получить ответ также можно при помощи кнопок на сцене. Необходимо осуществить валидацию - разрешить вводить только числовые значения в поля ввода.

В папке Assets создадим две директории Scripts и Scenes. Далее во вкладке Hierarchy выберем Main Camera. Во вкладке Inspector откроются настройки для объекта камеры. Установим Background в серый цвет(#CCCCCC).

Создадим пустой холст. Щёлкнем правой кнопкой по сцене, выберем GameObjects > UI > Canvas. Для только что созданного Canvas в Inspector установим Pixel Perfect чекбокс. В разделе Canvas Scaler UI Scale Mode выберем Scale With Screen Size. Значение Screen Match Mode = Match Width Or Height, Match = 0, Reference Resolution установим по x в 600, по у в 100.

Добавим два пустых GameObject на сцену, которые будут выполнять роль строк для полей ввода и кнопок операций. Во вкладке Hierarchy правой кнопкой по Canvas > Create Empty. Во вкладке Inspector для только что созданного GameObject изменим название на Display. Установим Anchor Presets по горизонтали stretch, по вертикали top, Pos Y = -10, Height = 60. Далее добавим Horizontal Layout Group, для этого во вкладке Inspector спуститься вниз, нажать на кнопку Add Component, далее Layout > Horizontal Layout Group. Дадим следующие настройки для Padding: Left = 5, Right = 5, Top = 10, Bottom = 5. Child Alignment оставим значение по умолчанию Upper Left. Для автоматического расчёта высоты вложенных элементов поставим галочки на следующих чекбоксах: Height для Control Child Size, Height для Child Force Expand. Значения других чекбоксов необходимо поставить в выключенное состояние. Во вкладке Hierarchy скопируем объект Display. Во вкладке Inspector изменим название на Controls. Pos Y изменим на значение -80. Изменим у Horizontal Layout Group Padding: Left = 0, Right = 5, Top = 5, Bottom = 10. Поставим Spacing = 3, Child Alignment =Upper Right.

Добавим поля ввода, текст операции и ответа внутрь объекта Display. Для этого во вкладке Hierarchy правой кнопкой по Display > UI > Input Field, правой кнопкой по Display > UI > Text. Сначала один раз скопируем InputField, затем два раза Text. Дадим созданным объектам следующие названия: InputField1, OperatorText, InputField2, EqualText, AnswerText. Установим следующие значения Width по порядку: 160, 40, 160, 40, 190. Для OperatorText значение Text изменим на +, включим Best Fit. Для EqualText значение Text изменим на =, включим Best Fit. Для AnswerText значение Text изменим на пустую строку, включим Best Fit.

Добавим кнопки операций внутрь объекта Controls. Для этого во вкладке Hierarchy правой кнопкой по Controls > UI > Button. Установим ей Width = 60. Четыре раза скопируем созданную кнопку. Дадим кнопкам следующие названия: Button1, Button2, Button3, Button4, Button5. Для вложенных в Button Text сделаем значения Text следующими: +, -, x, /, =.

Создадим обработчики события для кнопок, валидацию ввода и обработку нажатий с клавиатуры. Папка Assets > Scripts, правой кнопкой, Create C# Script. При создании скрипта дадим ему название MainController. Добавим этот скрипт к элементу Canvas. Вкладка Inspector, спуститься вниз, нажать на кнопку Add Component, далее Scripts > MainController.

Исходный код MainController.cs:

using UnityEngine;

using UnityEngine.UI;

public class MainControl : MonoBehaviour {

Text operatorLabel;

Text answerLabel;

InputField input1;

InputField input2;

public void Start()

{

operatorLabel = GameObject.Find("OperatorText").GetComponent<Text>();

answerLabel = GameObject.Find("AnswerText").GetComponent<Text>();

input1 = GameObject.Find("InputField1").GetComponent<InputField>();

input2 = GameObject.Find("InputField2").GetComponent<InputField>();

input1.onValidateInput += CharValidate;

input2.onValidateInput += CharValidate;

for (int i = 1; i <= 4; i++)

{

Button btn = GameObject.Find("Button" + i).GetComponent<Button>();

btn.onClick.AddListener(() => onOperatorClick(btn));

}

Button btn5 = GameObject.Find("Button5").GetComponent<Button>();

btn5.onClick.AddListener(onAnswerClick);

}

private char CharValidate(string input, int charIndex, char c)

{

string newVal = input.Insert(charIndex, c.ToString());

Debug.Log("OpValidate newVal " + newVal);

double res;

if (!double.TryParse(newVal, out res))

{

c = '\0';

}

return c;

}

public void onOperatorClick(Button btn)

{

Text label = btn.GetComponentInChildren<Text>();

Debug.Log("onOperatorClick " + label.text);

operatorLabel.text = label.text;

}

public void onAnswerClick()

{

Debug.Log("onAnswerClick");

try

{

double answer = 0;

double num1 = double.Parse(input1.text);

double num2 = double.Parse(input2.text);

string op = operatorLabel.text;

switch (op)

{

case "+":

answer = num1 + num2;

break;

case "-":

answer = num1 - num2;

break;

case "x":

answer = num1 \* num2;

break;

case "/":

answer = num1 / num2;

break;

default:

break;

}

answerLabel.text = answer.ToString();

}

catch

{

Debug.Log("Parse Error");

answerLabel.text = "Error";

}

}

private void Update()

{

if (Input.GetKeyDown(KeyCode.Return) || Input.GetKeyDown(KeyCode.KeypadEnter) ||

Input.GetKeyDown(KeyCode.Equals) || Input.GetKeyDown(KeyCode.KeypadEquals))

onAnswerClick();

if (Input.GetKeyDown(KeyCode.Plus) || Input.GetKeyDown(KeyCode.KeypadPlus))

operatorLabel.text = "+";

if (Input.GetKeyDown(KeyCode.Minus) || Input.GetKeyDown(KeyCode.KeypadMinus))

operatorLabel.text = "-";

if (Input.GetKeyDown(KeyCode.KeypadMultiply))

operatorLabel.text = "x";

if (Input.GetKeyDown(KeyCode.KeypadDivide))

operatorLabel.text = "/";

if (Input.GetKeyDown(KeyCode.Tab))

{

if (input1.isFocused)

input2.ActivateInputField();

else

input1.ActivateInputField(); }

}

}

Подключим пространства имён UnityEngine и UnityEngine.UI. В классе MainControoler создадим приватные переменные:

* operatorLabel типа Text для отображения текущей операции
* answerLabel типа Text для отображения результата вычислений
* input1, input2 типа InputField для валидации ввода и наведения фокуса

В приватном методе Start, который вызывается до отрисовки первого кадра, проинициализируем значения переменных и добавим обработчики событий для кнопок. Для поиска UI элемента по его имени используется статический метод класса GameObject с названием Find. После этого осуществляется преобразование типа GameObject к нужному типу при помощи метода GetComponent. Для добавления обработчика события нужно обратиться к свойству события и вызвать метод AddListener (или оператор +=) и в качестве аргумента передать ссылку на метод обработчик. Сначала находим метки OperatorText, AnswerText и поля ввода InputField1, InputField2 по их названиям. Далее навешиваем обработчики ввода символа CharValidate для полей ввода. Можно обратить внимание на повторное использование метода onOperatorClick в качестве обработчика события для кнопок операций. Для сокращения кода был использован цикл for от 1 до 4, так как кнопки операций имеют названия Button1 – Button4. Для определения текущей кнопки операции в обработчик нажатия onOperatorClick передается ссылка на кнопку. На кнопку с названием Button5 навешивается метод onAnswerClick.

Метод CharValidate принимает на вход строку до ввода, позицию курсора и вводимый символ. На основании параметров получаем строку после ввода. Если она не преобразуется к типу double функция возвращает пустой символ. Если проблем с преобразованием нет, тогда возвращается вводимый символ.

Метод onOperatorClick принимает на вход текущую кнопку операции и получает вложенный в неё элемент Text. Для его получения используется метод GetComponentInChildren переменной btn - осуществляет поиск вложенного элемента с заданным типом. Далее свойство text переменной operatorLabel изменяется на значение текущей операции.

Метод onAnswerClick преобразует к числам значения в полях ввода input1, input2. Далее получает операцию по значению text operatorLabel и производится математическую операция с двумя числами. Ответ записывается в свойство text answerLabel. В случае возникновения ошибки будет записано значение Error.

Приватный метод Update вызывается при перерисовке кадров. В нём с помощью Input.GetKeyDown проверяется нажатия на клавиатуре. При нажатии на равно и Enter будет вызываться метод onAnswerClick. При нажатии на плюс, минус, делить, умножить меняться значение operatorLabel. При нажатии на Tab, если input1 не в фокусе, он станет активен, иначе фокус получит input2.