**Приложение к уроку 2**

**1 слайд** – что важно при создании проекта

**2 слайд –** разбираем интерфейс Unity

**Красное поле** – игровая сцена, у него есть вкладка Scene, Game и AssetStore. Говорим про первые две вкладки.

**Синее поле** – свойства текущего выбранного объекта.

GameObject – это любой объект в игре. Следует понимать, что сам по себе объект ничего не делает. Ему требуются специальные свойства, прежде чем он станет персонажем, окружением или визуальным эффектом.

Объекты — это лишь контейнеры. Они могут содержать в себе различные элементы, комбинации которых превращают объект в персонаж, декорацию, спецэффект. Эти элементы называются компонентами (Components). В зависимости от того, что нужно создать, объекту присваиваются различные комбинации компонентов.

Смотрим на примере камеры. У каждого объекта есть компоненты. Главный компонент – это трансформация объекта:

Попробовать поменять и посмотреть.

Сбросить в ноль

Рассмотреть, какие есть еще пункты меню

Можно поменять фон сцены и посмотреть изменения во вкладке Game.

**Зеленое поле** – окно иерархии объектов сцены, пока там только камера.

Кнопка Create, создать пару компонент и продемонстрировать вложение.

**Желтое поле** – окно проекта, его файлы и папки. Посмотрим, что входит в состав пустого проекта. Главная папка Assets, в ней папка Scenes – это наши игровые сцены, пока она одна.

Попробовать создать пару папок и переместить их друг в друга. Лучше это делать так, чем через проводник – там нарушается связь между папками проекта.

Есть еще вкладка Console – она показывает отладочную информацию.

**Фиолетовое поле** – кнопки управления сценой и объектами на ней.

Демонстрация работы кнопок.

Отдельно про кнопки предпросмотра игры. Обратить внимание, что изменения сцены в этом режиме не сохраняются.

**Главное меню** – пункт File на нем нужно заострить внимание.

**3 слайд –** Что нужно, чтобы создать игру? Отвечаем на вопрос и попутно показываем анимированный слайд

**4 слайд -** В чем разница между облаком и монеткой. Объяснить, что спрайт – это то, с чем происходит какое-то взаимодействие.

**5 слайд –** атласы или карты.

**6 слайд –** про звуки.

**7 слайд** – что такое ассет?

 ассетам относятся:

* спрайты — двумерные изображения игровых объектов (деревья, фон, персонажи);
* аудио — музыка и звуковые эффекты;
* визуальные эффекты — взрывы, фильтры, вибрация и прочее;
* модели — трехмерные объекты (заменяют спрайты в трехмерных играх);
* префабы — заготовки объектов (готовые объекты с прикрепленными компонентами);
* текстуры — внешний вид для моделей;
* сцены — локации для игры;
* элементы интерфейса — шрифты, кнопки, изображения;
* анимации — специальные файлы, в которых указывается порядок смены кадров для двумерных игр;
* скрипты — игровой интеллект, код управления персонажем.

**8 слайд** - Спрайты можно нарисовать в Adobe Photoshop или Adobe Illustrator, но подойдет и MS Paint. Для этого нужно владеть векторной графикой или уметь рисовать в стиле pixel art. А еще можно просто сфотографировать реальный предмет, а потом обрезать фото и обработать его для игры.

Модели можно создавать в программах Blender или 3ds Max. Для этого нужно разбираться в трехмерной графике, в том числе и в создании скелетной анимации.

Аудио можно записать на диктофон, а потом обработать в Sony Sound Forge, Adobe Audition или в бесплатном Audacity. Могут получиться очень атмосферные эффекты, если у вас скрипят полы или двери.

**9 слайд** - здорово, когда ты сам сможешь нарисовать и создать эти ассеты, но для этого нужны специальные знания и много времени.

Поэтому существуют магазины платных и бесплатных ассетов.

По умолчанию в Unity есть стандартные ассеты (если вы не отключили их во время установки), но только для 3D-проектов. Для двумерных игр их приходится искать или создавать самостоятельно.

<https://assetstore.unity.com/>

Сначала просто показываем сайты - где требуется регистрация, как найти бесплатные ассеты, фоны, звуки, объекты. Два последних сайта позволяют просто скачивать пакеты.

И отдельно перейти к assetstore.unity.com, открыть среду и показать, как импортируются ассеты в проект.

**10 слайд** - Прежде всего — на их качество и размер. С большими ассетами игра станет более требовательной. Если же, наоборот, сжать спрайты, аудио и модели, то проект станет менее привлекательным. Поэтому нужно соблюдать баланс между производительностью и качеством.

Также вы можете дать игрокам возможность самим выбирать качество — но тогда придется добавлять ассеты в нескольких вариациях, и из-за этого увеличится размер программы.

**11 слайд** – каждый выбирает свою тему и стиль

Можно взять готовые ассеты с гугл-диска.

**14 слайд -** GameObject - самая важная концепция в редакторе Unity.

Каждый объект в вашей игре — это GameObject, от персонажей и коллекционных предметов до источников света, камер и спецэффектов. Однако GameObject ничего не может делать сам по себе; вам нужно присвоить ему свойства, прежде чем он сможет стать персонажем, средой или специальным эффектом.

Чтобы дать GameObject свойства, необходимые для того, чтобы он стал источником света, деревом или камерой, вам нужно добавить к нему компоненты. В зависимости от того, какой объект вы хотите создать, вы добавляете различные комбинации компонентов в GameObject.

Вы можете думать о GameObject как о пустом котелке, а компоненты - как о различных ингредиентах, составляющих рецепт вашей игры. Unity имеет множество различных типов встроенных компонентов, и вы также можете создавать свои собственные компоненты с помощью Unity Scripting API.

GameObject всегда имеет прикрепленный компонент Transform (для представления положения и ориентации), и удалить его невозможно. Другие компоненты, которые придают объекту его функциональность, могут быть добавлены из меню «Компонент» редактора или из сценария. Также есть много полезных готовых объектов (примитивные формы, камеры и т. Д.), Доступных в меню GameObject> 3D Object, см. «Примитивные объекты».

Показать, где находятся GameObject в Unity и как их добавить. Изначально объект без спрайта, но можно добавить компонент Sprite Renderer. Но это просто пустой компонент, чтобы связать его с конкретным спрайтом, нужно перетащить спрайт на свойство компонента. **– провести это на практике**.

Второй вариант - как только вы добавляете спрайт на сцену, Unity генерирует для него пустой Game Object и добавляет компонент Sprite Renderer, который автоматически связывает с добавляемым спрайтом - **провести это на практике**.

Обратите внимание, что у любого GameObject есть имя.

**15 слайд –** самостоятельная работа

Для звукового эффекта самолета задаем флаг “Play on Awake” и «Loop». Запускаем игру и смотрим.

**16 слайд -** В Unity есть множество самых разнообразных компонентов, но вы вскоре можете столкнуться с необходимостью создать собственный компонент для реализации своих алгоритмов. Это можно сделать с помощью скриптов, в которых описана ваша собственная игровая логика и поведение объектов, и которые затем прикрепляются к игровым объектам как компоненты.

Компоненты на основе скриптов позволяют запускать игровые события, проверять объект на предмет столкновений, применять физические свойства, программировать реакцию на управление пользователя и многое другое, например менять позицию объекта в зависимости от того, какую клавишу нажмет пользователь.

Как мы уже говорили, у каждого GameObject есть Transform, поэтому, чтобы переместить наш объект, мы можем менять позицию это объекта. Как нам его изменить – для этого мы должны создать скрипт.

Создаем скрипт, смотрим внутренности.

Каждый скрипт связывается с внутренними механизмами Unity путем реализации класса, производного от встроенного класса MonoBehaviour. Вы можете думать о классе как о своего рода заготовке для создания нового типа компонента, который может быть прикреплен к игровому объекту. Имя класса и имя файла должны быть одинаковыми, для того, чтобы скриптовый компонент мог быть присоединен к игровому объекту.

Основные вещи, достойные внимания, это две функции, определенные внутри класса. Функция **Update** - это место для размещения кода, который будет обрабатывать обновление кадра для игрового объекта. Это может быть движение, срабатывание действий и ответная реакция на ввод пользователя, в основном всё, что должно быть обработано с течением времени во игровом процессе. Чтобы позволить функции Update выполнять свою работу, часто бывает полезно инициализировать переменные, считать свойства и осуществить связь с другими игровыми объектами до того, как будут совершены какие-либо действия. Функция **Start** будет вызвана Unity до начала игрового процесса (т.е. до первого вызова функции Update), и это идеальное место для выполнения инициализации.

**18 слайд –** показать соответствие между названиями компонент в Unity и в коде

**22 слайд** - Возвращает значение виртуальной оси, идентифицированной axisName.

Значение будет в диапазоне -1 ... 1 для устройств ввода с клавиатуры и джойстика.

Значение этого значения зависит от типа элемента управления вводом, например, для горизонтальной оси джойстика значение 1 означает, что джойстик сдвинут полностью вправо, а значение -1 означает, что он полностью влево; значение 0 означает, что джойстик находится в нейтральном положении.

Если ось сопоставлена ​​с мышью, значение будет другим и не будет находиться в диапазоне -1 ... 1. Вместо этого это будет текущая дельта мыши, умноженная на чувствительность оси. Обычно положительное значение означает, что мышь движется вправо / вниз, а отрицательное значение означает, что мышь движется влево / вверх.

Чтобы настроить ввод или просмотреть параметры для axisName, выберите «Правка»> «Настройки проекта»> «Вход». Это вызывает диспетчер ввода. Разверните Axis, чтобы увидеть список ваших текущих входов. Вы можете использовать один из них как axisName. Чтобы переименовать вход или изменить положительную кнопку и т. Д., Разверните один из параметров и измените имя в поле «Имя» или «Положительная кнопка». Также измените Тип на Ось джойстика. Чтобы добавить новый вход, добавьте 1 к числу в поле Размер.

**24 слайд -** В 2D пространстве, координаты определены использованием горизонтальной оси (x) и вертикальной оси (y). Определённая позиция в 2D пространстве записывается как пара значений, таких как (4, 3).

Если вы новичок в компьютерной графике, вам может показаться странным, что положительная ось y указывает вниз, а не вверх, как вас учили в школе. Тем не менее, в приложениях компьютерной графики это встречается повсеместно.

Любая точка на 2D плоскости может быть таким образом определена парой чисел. Вы также можете рассматривать (4, 3) как смещение от точки (0, 0), или точки начала координат. Нарисуем стрелку от начала координат до заданной точки: картинка2

Это вектор. Вектор предоставляет множество полезной информации. Помимо сообщения нам, что точка расположена в (4, 3), мы можем также представить это в виде угла θ и длины (величины, модуля) m. В данном примере стрелка — это вектор позиции — он обозначает позицию в пространстве относительно начала координат.

**25 слайд -** При рассмотрении векторов крайне важно иметь в виду, что они представляют только относительные направление и величину (модуль). Вектору нельзя приписать определённой позиции. Два следующих вектора идентичны:

Оба вектора представляют точку 4 единицами правее и 3 единицами ниже стартовой точки. Неважно, где на плоскости вы нарисуете вектор, он всегда представляет относительное направление и величину.

**26 слайд -** Вектор может представлять любую длину и направление. Обычные примеры это: позиция, скорость, ускорение, и сила. На этом изображении, космический корабль на шаге 1 имеет вектор позиции (1,3) а вектор скорости (2,1). Вектор скорости представляет, как далеко пойдёт корабль на каждом шаге. Вы можете найти его позицию на шаге 2 добавляя скорость к текущей позиции.

Скорость вычисляет изменение позиции за единицу времени. Новая позиция находится добавление скорости к предыдущей позиции.

Давайте попробуем все это объединить в коде:

void Update()

{

float mV = Input.GetAxis("Horizontal");

transform.position = transform.position + new UnityEngine.Vector3(mV, 0, 0);

}

**27 слайд –** рассказать о том, что просто GameObject – это какой-то эфемерный объект, привидение, которым движок Unity не занимается в принципе.

**28 слайд - Твёрдые тела** позволяют вашим **игровым объектам** взаимодействовать с помощью физики. Для реалистичного перемещения твёрдых тел, на последние воздействуют сила вращения и другие силы. Любой игровой объект должен содержать в себе твёрдое тело, чтобы быть подверженным гравитации, действовать согласно назначенным путём скриптинга силам, или взаимодействовать с другими объектами через физический движок NVIDIA PhysX.

**29 слайд –** Unity есть смысл следить только за реальными объектами, поскольку две стены никогда не столкнутся, если это не предусмотрено у вас в игре. Тогда зачем тратить ресурсы?

Добавляем машине. Запускаем игру и смотрим.

Машина упала, почему? Как исправить? Добавить земле компонент Rigidbody. Но тогда земля тоже упадет.

**30 слайд** - Динамический тип: динамическое тело сталкивается с любым другим типом телосложения и является наиболее интерактивным из всех типов телосложения. Это тип тела по умолчанию для Rigidbody 2D, потому что это наиболее распространенный тип тела для вещей, которые нужно перемещать. Это также самый дорогой тип тела из-за его динамичного характера и интерактивности со всем, что его окружает. Все 2D-свойства Rigidbody доступны для этого типа тела.

Static Rigidbody 2D вообще не движется при моделировании; если с ним что-то сталкивается, статическое твердое тело 2D ведет себя как неподвижный объект (как если бы он имел бесконечную массу). Это также наименее ресурсоемкий тип телосложения. Статическое тело сталкивается только с 2D-объектами Dynamic Rigidbody. Столкновение двух статических жестких тел 2D не поддерживается, поскольку они не предназначены для перемещения.

Kinematic Rigidbody 2D предназначен для перемещения в режиме моделирования, но только под явным контролем пользователя. В то время как на Dynamic Rigidbody 2D действуют сила тяжести и силы, на Kinematic Rigidbody 2D - нет. По этой причине он работает быстро и требует меньше системных ресурсов, чем Dynamic Rigidbody 2D.

Kinematic Rigidbody 2D по-прежнему движется со своей скоростью, но на скорость не влияют силы или гравитация. Kinematic Rigidbody 2D не сталкивается с другими Kinematic Rigidbody 2D или со статическим Rigidbody 2D; он сталкивается только с 2D-объектами Dynamic Rigidbody. Подобно Static Rigidbody 2D (см. Ниже), Kinematic Rigidbody 2D ведет себя как неподвижный объект (как если бы он имел бесконечную массу) во время столкновений. Свойства, связанные с массой, недоступны для этого типа тела.

**32 слайд -** Тип Dynamic – по умолчанию

Не используйте компонент Transform для задания положения или поворота Dynamic Rigidbody 2D. Положение нужно изменять с помощью velocity. То есть мы работаем не с самим компонентом, а с физическими силами, которые воздействуют на объект.

Пробуем.

void Update()

{

float mV = Input.GetAxis("Horizontal");

rb.velocity = new UnityEngine.Vector2(mV,0);

Но машина все равно падает. В чем же дело?

**33 слайд –** пытаемся ответить на вопрос слайда. Наводящие вопросы – если у нас сложный герой?

**34 слайд -** Компоненты коллайдера определяют форму GameObject для целей физических столкновений. Коллайдер, который является невидимым, не обязательно должен иметь ту же форму, что и сетка GameObject. Грубая аппроксимация сетки часто более эффективна и неотличима в игровом процессе.

Самые простые (и наименее требовательные к процессору) коллайдеры относятся к примитивным типам коллайдеров. В 3D это Box Collider, Sphere Collider и Capsule Collider. В 2D вы можете использовать Box Collider 2D и Circle Collider 2D. Вы можете добавить любое количество из них к одному GameObject для создания составных коллайдеров.

Пробуем добавить всем нашим объектам коллайдеры. Смотрим, что получилось.

**35 слайд –** Для создания пола, стен и других неподвижных элементов сцены, к объектам без компонента Rigidbody можно добавлять коллайдеры. Они называются **static** (статичными) коллайдерами. Обычно вам не следует перемещать статичные коллайдеры изменяя положение их трансформации, т.к. это сильно повлияет на производительность физического движка. Коллайдеры на объекте, у которого *есть* Rigidbody, известны как *динамичные* коллайдеры. Статичные коллайдеры могут взаимодействовать с динамичными коллайдерами, но, т.к. у них нету Rigidbody, они не сдвинутся в ответ на коллизии (столкновения).

**36 слайд –** коллайдер не работает без Ridgidbody!!!

**37 слайд –** отслеживаем момент столкновения

На кактус должен стоять AudioSource.

void OnCollisionEnter2D(Collision2D collision)

{

}

**38 слайд –** кратко, что такое отладочное сообщение, откуда оно берется и где отображается

Debug.Log("OnCollisionEnter2D");

Смотрим, что лог выводится всегда, как только происходит соприкосновение двух коллайдеров.

Как отследить кактус?

**39 слайд –** Тег может использоваться для идентификации игрового объекта.

Показываем как добавить тэг. Сначала добавляем новый тэг, сохраняем его, затем назначаем!

if (collision.gameObject.CompareTag("Cactus"))

collision.gameObject.GetComponent<AudioSource>().Play();

Kinematic Rigidbody 2D предназначен для явного изменения положения с помощью Rigidbody2D.MovePosition или Rigidbody2D.MoveRotation. Используйте физические запросы для обнаружения столкновений и сценарии, чтобы решить, куда и как следует перемещать Rigidbody 2D.

**15 слайд** -

**13 слайд** – Если вы хотите определить столкновения, но не ходите, чтоб столкнувшийся объект остановился, вам требуется использовать триггер (trigger).

У компонента box collider есть чекбокc «Is Trigger». Если включить  
его и запустить сцену снова, то куб провалиться через платформу.

В случае столкновения, система скриптинга может это обнаружить и выполнить действия, указанные в функции OnCollisionEnter. Однако вы также можете использовать физический движок просто для обнаружения того, что один коллайдер входит в пространство другого, без создания коллизии. Коллайдер, настроенный как **триггер** (с помощью свойства **Is Trigger**), не ведёт себя как твёрдый объект и просто будет пропускать другие коллайдеры сквозь себя. Когда другой коллайдер войдёт “на территорию” этого коллайдера, триггер вызовет функцию OnTriggerEnter в скриптах объекта, к которому присоединён триггер.

Звук добавляем все вместе, нужно добавить компонент препятствию AudioSource. В этот компонент в свойство AudioClip перетащить нужный звуковой ассет.

Создать скрипт и в нем написать:

void OnTriggerEnter2D(Collider2D col)

{

GetComponent<AudioSource>().Play();

}