****

**МОСКОВСКИЙ АВТОМОБИЛЬНО-ДОРОЖНЫЙГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ (МАДИ)** Кафедра “Высшая математика”  
  
  
**Курсовой проект**

По дисциплине: Компьютерная графика**на тему:**

**«Разработка 3D сцен в Unity3D»**

Выполнил:

Студент группы 3бПМ

Нетовканая А.С.Проверил:

Кутейников И.А

г. Москва, 2020

Содержание

Введение …………………………………………………………………………………………………………….3

1. Знакомство с Unity ………………………………………………………………………………………..4
2. Практическая часть ……………………………………………………………………………………….5

2.1 Управление сферой …………………………………………………………………………………5

2.2 Работа с камерой ……………………………………………………………………………………..7

2.3 Объекты из Unity Asset Store ………………………………………………………………….9

1. Вывод …………………………………………………………………………………………………………..14

Список литературы …………………………………………………………………………………………..15

Введение

Самым популярным видом досуга среди молодёжи на сегодняшний момент являются компьютерные игры, они стали неотъемлемой частью нашей жизни. Виртуальные миры открывают нам огромные возможности, они позволяют выбрать себе различные роли. Разработкой видеоигр может заниматься как один человек, так и фирма (коллектив разработчиков). Создание игры — это продолжительный и трудоёмкий процесс, состоящий из самых разнообразных этапов, включающий в себя как технические, так и творческие моменты. Сначала обдумывают цель и средство (среду) разработки игры, затем во время творчества игры создаю игровую механику, уровни, сюжет, графику и звук.

Целью работы является разработка 3D объектов на Unity3D.

Актуальность создания состоит в том, что создатели игр могут создавать свои миры, людей, флору и фауну, различных существ и связь всего этого друг с другом. Они должны просчитывать всё до мельчайших подробностей, и любая оплошность может привести к багам и поломкам.

И в связи с поставленной целью в курсовом проекте решается следующие задачи:

- Создание физического объекта;

- Присвоение объекту движение;

- Работа с камерой;

- Добавление графики, ассетов из Unity Asset Store

1. Знакомство с Unity

Unity – больше, чем движок, это среда для разработки компьютерных игр, в которой объединены различные программные средства, используемые при создании ПО – текстовый редактор, компилятор, отладчик и так далее. При этом, благодаря удобству использования, Unity делает создание игр максимально простым и комфортным, а мультиплатформенность движка позволяет игроделам охватить как можно большее количество игровых платформ и операционных систем.

Преимущества:

1. Здесь используется компонентно-ориентированный подход, в рамках которого разработчик создает объекты (например, главного героя) и к ним добавляет различные компоненты (например, визуальное отображение персонажа и способы управления им). Благодаря удобному Drag & Drop интерфейсу и функциональному графическому редактору движок позволяет рисовать карты и расставлять объекты в реальном времени и сразу же тестировать получившийся результат.
2. Наличие огромной библиотеки ассетов и плагинов, с помощью которых можно значительно ускорить процесс разработки игры. Их можно импортировать и экспортировать, добавлять в игру целые заготовки – уровни, врагов, паттерны поведения ИИ и так далее. Никакой возни с программированием. Многие ассеты доступны бесплатно, другие предлагаются за небольшую сумму, и при желании можно создавать собственный контент, публиковать его в Unity Asset Store и получать от этого прибыль.
3. Поддержка огромного количества платформ, технологий, API. Созданные на движке игры можно легко портировать между ОС Windows, Linux, OS X, Android, iOS, на консоли семейств PlayStation, Xbox, Nintendo, на VR- и AR-устройства. Unity поддерживает DirectX и OpenGL, работает со всеми современными эффектами рендеринга, включая новейшую технологию трассировки лучей в реальном времени.
4. Unity доступен бесплатно, что открывает перед независимыми разработчиками дверь в игровую индустрию. Конечно, существуют ограничения: бесплатная версия движка демонстрирует лого Unity перед запуском игры, а проект, созданный с ее помощью, не должен приносить разработчику больше $100 тысяч в год.

Недостатки:

1. Если команда захочет разработать что-нибудь сложнее простого кликера или платформера, то ей придется искать хорошего программиста на C#, который напишет скрипты и компоненты, внедрит их в игру и заставит работать.
2. Медлительность. Создание масштабных, сложных сцен с множеством компонентов может негативно повлиять на производительность игры, в результате чего разработчикам придется потратить дополнительное время и ресурсы на оптимизацию, а возможно – и удаление некоторых элементов из проекта.
3. Приложения, созданные на Unity, довольно «тяжеловесны»: даже самая простая пиксельная игра может занимать несколько сотен мегабайт на ПК. Да, для жесткого диска компьютеров это небольшой объем, но, если проект разрабатывается и для мобильных платформ, следует задуматься об оптимизации его размера.
4. Практическая часть

2.1 Управление сферой

Многие игры включают в себя персонажей, которые должны передвигаться, чтобы достичь какой-то цели. Экшн-игры дают вам прямое управление, управляя персонажем, обычно нажимая клавиши или поворачивая курок джойстика. В играх жанров квест или стратегия указывается позиция цели, и персонаж перемещается туда автоматически.

В этом курсовом проекте мы будет управлять сферой, как персонажем в 3D игре.

Для начала создаем новый 3D проект по умолчанию. В проекте уже имеется сцена SampleScene, которая уже имеет основную камеру и свет, которые понадобятся позже. В этой сцене мы создает плоскость, на которой и будет находится наша сфера (Рис.1).

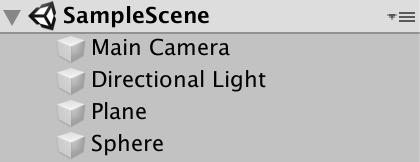


Рис.1 SampleScene

Устанавливает радиус и расположение сферы, которое нам необходимо. Так же можно задать материалы для сферы и плоскости, на начальной этапе мы сделали сферу черной, а плоскость серой. Рис.2. Настраиваете материалы, как нравится вам.

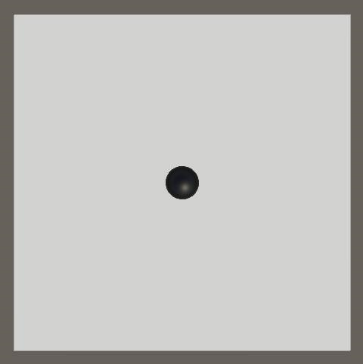


Рис. 2. Заданная сфера и плоскость

Для реализации движения используем скрипт MovingSphere. И начинаем с простого, перемещая сферу вокруг определенной плоскости. И как только мы настроем нужное для нас движение, мы перейдем к более сложному. И если мы хотим, чтобы наша сфера двигалась в трехмерной среде, мы должны поддерживать ее взаимодействие с произвольной геометрией. И будем использовать существующий физический движок Unity, которым является NVIDIA PhysX.

Существует два основных способа управления персонажем в сочетании с физическим движком. Во-первых, это подход с твердым телом (rigidbody) , который заключается в том, чтобы персонаж вел себя как обычный физический объект, применяя силы или изменяя его скорость. Вторым является кинематический подход, который заключается в прямом управлении, когда физический движок запрашивается только для выполнения обнаружения столкновений.

Мы же в этом курсовом проекте используем первый метод и добавляем компонент Ragidbody. Для использования физический движок, мы должны позволить ему контролировать положение нашей сферы. Непосредственная корректировка положения была бы телепортацией, а это не то, чего мы хотим. Вместо этого мы должны косвенно контролировать сферу, применяя силы к ней или регулируя ее скорость.

После того, как сфера начнет перемешаться в трехмерной физике, мы дадим ей возможность прыгать.

Мы можем использовать Input.GetButtonDown (“Jump”), чтобы определить, нажал ли игрок кнопку прыжка для этого кадра, которая по умолчанию является клавишей пробела.

В настоящий момент мы можем прыгнуть и на земле, и в воздухе. Чтобы этого не было, мы используем метод OnCollisionEnter и метод OnCollisionExit. И добавляем оба метода в MovingSphere, установив для нового логического поля onGround значение true в первом и значение false во втором.

Теперь мы можем прыгать только тогда, когда мы на земле, что на данный момент подразумевает тот случай, когда мы касаемся чего-то. Если мы ни с чем не контактируем, то желаемый прыжок следует игнорировать. Рис.3.

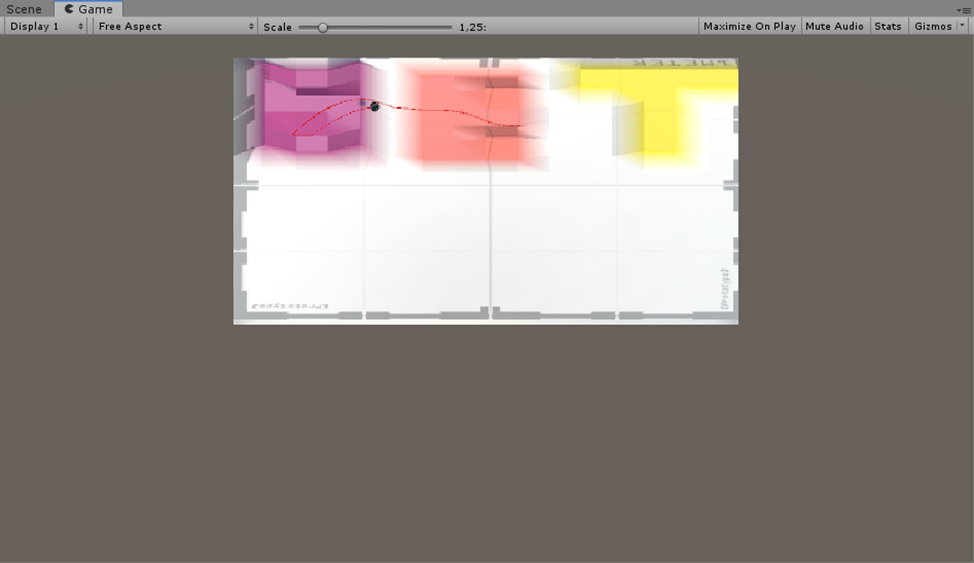


Рис. 3. Прыжок сферы

* 1. Работа с камерой

Фиксированная точка обзора работает только тогда, когда сфера ограничена полностью видимой областью. Но обычно персонажи в играх могут бродить по большим территориям. Типичные способы сделать это возможным - использовать вид от первого лица или заставить камеру следовать за аватаром игрока в режиме просмотра от третьего лица. Существуют и другие подходы, например, переключение между несколькими камерами в зависимости от положения аватара.

Мы создаем простую вращающуюся камеру, которая будет следовать за нашей сферой в режиме от третьего лица. Определите для нее тип компонента OrbitCamera, присвоив ему атрибут RequireComponent для принудительного подключения к игровому объекту, который также имеет регулярный компонент Camera.

Чтобы камера была сфокусирована на сфере, нам нужно указать ей, на чем фокусироваться. Это может быть что угодно, поэтому добавьте настраиваемое поле Transform для фокуса. Также добавьте параметр для расстояния до точки орбиты, по умолчанию оно установлено на пять единиц.

Камера не всегда будет оставаться на одном и том же расстоянии и ориентации, но поскольку PhysX регулирует положение сферы с фиксированным шагом по времени, то же самое делает и наша камера. Если этот процесс не будет соответствовать частоте кадров, это приведет к дрожанию камеры.

Самый простой и надежный способ исправить это - установить для свойства Rigidbody сферы параметр интерполяции ее положения. Это избавит от дрожащего движения сферы и камеры. Обычно это требуется только для объектов, на которых фокусируется камера.

На данный момент у нас есть простая камера орбиты. Теперь мы собираемся сделать движение игрока относительно точки обзора камеры.

Входные данные могут быть определены в любом пространстве, а не только в мировом пространстве или камере орбиты. Это может быть любое пространство, определенное компонентом Transform. Для этого добавьте в MovingSphere поле конфигурации пространства ввода игрока. Присвойте камеру орбиты этому полю. Это специфичная для сцены конфигурация, поэтому она не является частью префаба сферы, хотя она может быть настроена на себя, что будет порождать движение относительно ее собственной ориентации.

Наша камера заботится только о своем положении и ориентации относительно ее фокуса. Он ничего не знает об остальной части сцены. Таким образом, она идет прямо через другую геометрию, что вызывает несколько проблем. Во-первых, это некрасиво. Во-вторых, это может привести к тому, что геометрия будет препятствовать нашему взгляду на сферу, что затруднит навигацию. В-третьих, отсечение геометрии может выявить области, которые не должны быть видны. Мы начнем с рассмотрения только случая, когда расстояние фокусировки камеры установлено на ноль.

Существуют различные стратегии, которые могут использоваться для обеспечения правильного обзора камеры. Мы применим самое простое, а именно: потянуть камеру вперед по направлению ее взгляда, если что-то окажется между камерой и точкой фокусировки.

Самый очевидный способ обнаружить проблему - направить луч из точки фокусировки туда, где мы хотим разместить камеру. Сделайте это в OrbitCamera.LateUpdate, как только у нас будет направление взгляда. Если мы ударяемся о что-то, мы используем расстояние до точки удара вместо настроенного расстояния.

Притянув камеру ближе к точке фокусировки, вы можете приблизить ее к сфере. Когда сфера пересекает ближнюю плоскость камеры, она может частично или даже полностью обрезаться. Вы можете установить минимальное расстояние, чтобы избежать этого, но это будет означать, что камера остается внутри другой геометрии. Идеального решения для этого не существует, но его можно уменьшить, ограничив вертикальные углы орбиты, не делая геометрию уровня слишком узкой и уменьшив расстояние до плоскости отсечения камеры. Рис. 4.

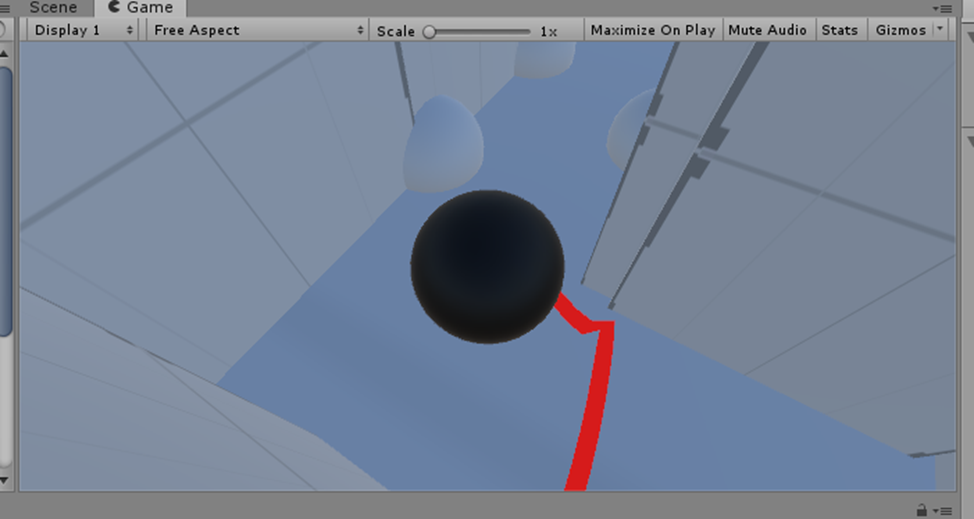


Рис. 4. Пример работы камеры

* 1. Объекты из Unity Assets Store

Чтобы разнообразить нашу плоскость. Мы будет использовать объекты, которые можно скачать из Unity Assets Store. Для этого нужно перейти в Unity Window/Asset Store. (Рис. 5)

После этого появляется окно

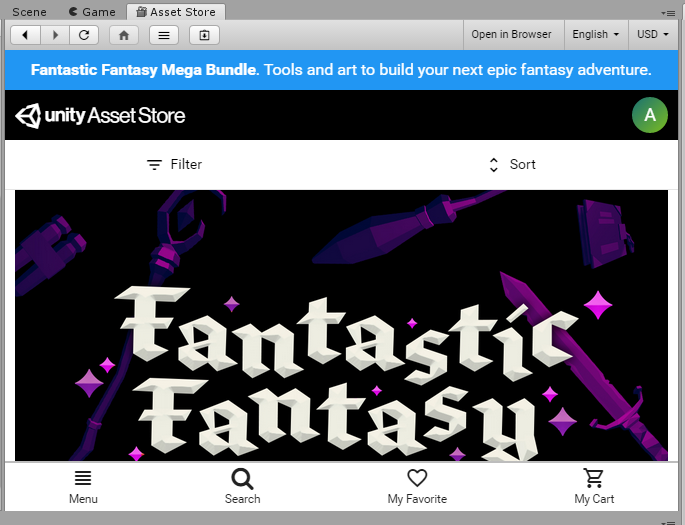


Рис. 5. Магазин Unity AssetStore

Для того, чтобы скачивать ассеты, мы должны быть зарегистрированны на официальном сайте Unity. Далее, мы в Search набираем то, что нам нужно, не забыв указать в Filter, что нам нужны только бесплатные ассеты. (Рис. 6)

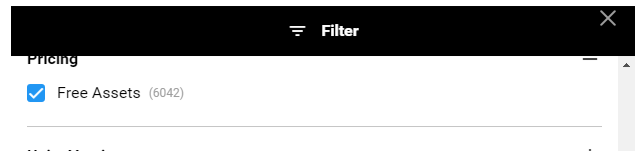


Рис. 6. Filter

После этого выбираем нужные нам ассеты, скачиваем и импортируем в наш проект.

Для этой курсовой работы я использовала несколько ассетов: Darth\_Artisan (деревья), KajamansRoads (дорога и материал зелени), Low Poly Houses Pack (дома), с (цветы) и Meshes (кусты). На рис. 7 показан пример импортированного ассета

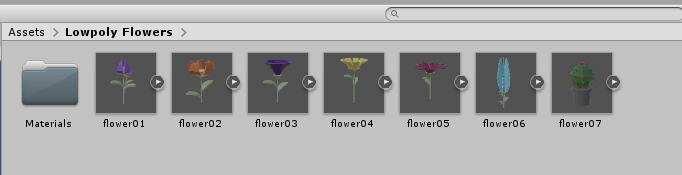


Рис. 7. Импортируемый ассет Lowpoly Flowers

После того, как все ассеты, которые нам нужны, импортированы, мы добавляем их на нашу Scene. (Рис. 8-12)

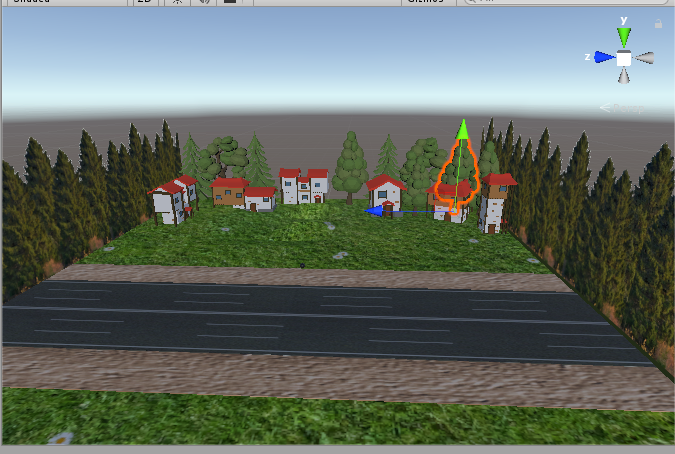


Рис. 8

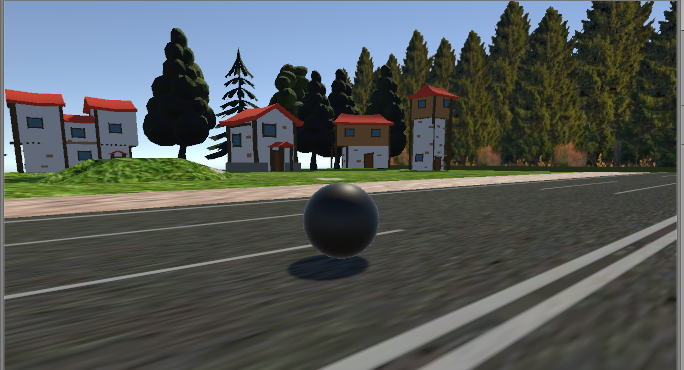


Рис. 9

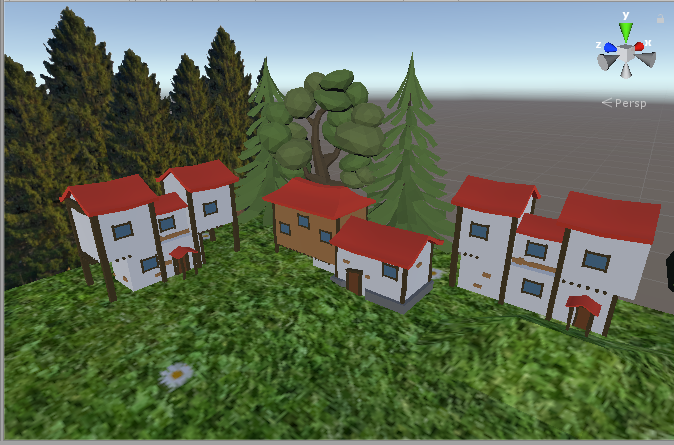


Рис. 10

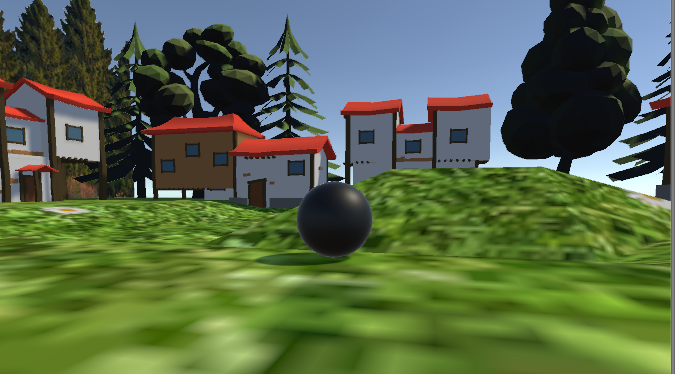


Рис. 11

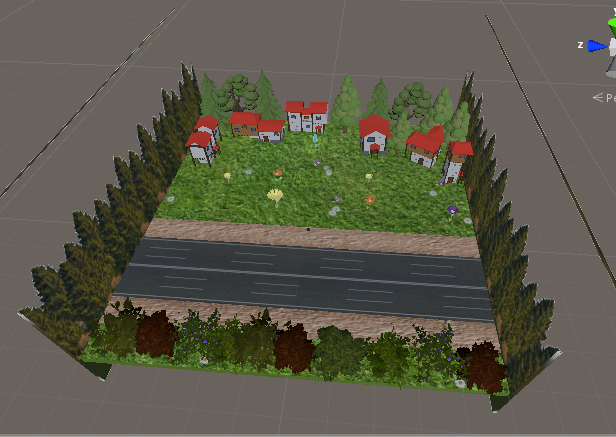


Рис. 12. Полностью обставленная объектами сцена

1. Вывод

С помощью этого курсового я много узнала о Unity и разработке 3D объектов. Здесь были рассмотрены преимущества и недостатки движка, управление объектов и настройка камеры. А также было рассмотрено, как с помощью asset store можно украсить свою сцену ассетами, что уж очень облегчает работу при создании игр.

На нем уже написано сотни игр и приложений и в дальнейшем будет еще не мало написано. Unity3d — очень гибкий движок, предоставляющий большую свободу действий пользователю. Чтобы упростить жизнь разработчику и потребителю, можно предпринять некоторые шаги для оптимизации своего проекта.

Список литературы

1. Unity в действии, мультиплатформенная разработка на С #/ Джозеф Хокинг: Питер, 2016, 336 стр.;
2. Unity3d. Начало работы, практические советы. Рецензия / Хабр [Электронный ресурс] URL: <https://habr.com/ru/post/161463/>
3. Unity User Manual (2018.3) [Электронный ресурс] URL: <https://docs.unity3d.com/ru/2018.4/Manual/UnityManual.html>
4. Движок Unity – особенности, преимущества и недостатки [Электронный ресурс] URL: https://cubiq.ru/dvizhok-unity/