МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Основы геометрического моделирования в Unity3d

Методические указания к выполнению лабораторных работ
для бакалавров, специалистов и магистров по направлениям
200100 – Приборостроение,
230400 – Информационные системы и технологии

Составители: З. В. Степчева,
О. С. Ходос

УлГТУ
УлГТУ
2012

УДК 004.925.8 (076)
ББК 32.973.26-018.2я7
О-75

Рецензент
Доцент кафедры ВТ УлГТУ, канд. техн. наук Святов К. В.

Одобрено секцией методических пособий
научно-методического совета университета

Основы геометрического моделирования в Unity3d :
О-75 методические указания / сост.: З. В. Степчева, О. С. Ходос. – Ульяновск : УлГТУ, 2012. – 33 с.

Методические указания написаны в соответствии с рабочими программами курсов «Компьютерная графика», «Машинная графика», «Компьютерная геометрия и графика» для бакалавров по направлениям 200100 – Приборостроение, 230400 – Информационные системы и технологии, а также специалистов по направлению 23020165 – Информационные системы и технологии.

В методических указаниях описаны три лабораторные работы, посвященные основам работы в Unity3d. Предназначены студентам дневной формы обучения для выполнения лабораторных работ по дисциплинам «Компьютерная графика», «Машинная графика», «Компьютерная геометрия и графика». Могут быть также использованы студентами других специальностей.

Указания подготовлены на кафедре «Измерительно-вычислительные комплексы».

УДК 004.925.8 (076)
ББК 32.973.26-018.2я7

Оглавление

Введение 4

1. Введение в Unity3d 5

1.1. Общие сведения о Unity3d 5

1.2. Концепция Game Engine в Unity3d 6

1.3. Терминология Unity3d 9

2. Интерфейс Unity3d. Создание простейших моделей твердых тел. Добавление массы, гравитации к твердому телу 11

3. Основы взаимодействия (столкновения) между объектами. Применение скриптов на языке C#. 21

4. Префабы. Копирование и удаление объектов среды в Unity3D. Создание префабов с применением скриптов C#. 28

Рекомендуемая литература 33

Введение

Важное место в программе подготовки бакалавров по направлениям 200100 «Приборостроение», 230400 «Информационные системы и технологии» и выпускников по специальности 23020165 «Информационные системы и технологии» занимают проблемы современных аппаратно-программных средств визуализации интерактивной трехмерной графики в дисциплинах «Компьютерная графика», «Компьютерная геометрия и графика», «Машинная графика», содержание которых определяется современными государственными образовательными стандартами.

В методических указаниях раскрываются современные методы и средства разработки Unity3d – профессионального мультиплатформенного инструмента для геометрического моделирования и программирования интерактивной трехмерной графики.

В указаниях описаны визуальные инструменты для разработки и визуализации интерактивных 3D-проектов в Unity3d в режиме реального времени, в том числе с использованием скриптов на языке C#, а также описаны пути решения задачи экспорта и импорта данных.

Приводятся конкретные примеры типичных ситуаций в Unity3d с использованием программирования на языке C# с подробным их разбором.

Указания предназначены для использования студентами наряду с учебниками, учебными пособиями и лекционным материалом.

1. Введение в Unity3d

1.1. Общие сведения о Unity3d

Unity3d – это мощный мультиплатформенный инструмент для разработки и программирования интерактивных браузерных и настольных приложений с двух- и трехмерной графикой, обрабатываемой в реальном времени.

Проект Unity3d основан в 2005 году в Дании компанией Unity Technologies, имеет штаб-квартиру в San Francisco и рабочие группы в Копенгагене, Лондоне, Стокгольме, Вильнюсе, Сеуле, Токио.

Все версии проекта Unity3d содержат интегрированный редактор проектов, поддерживают импорт графических и неграфических ресурсов (моделей, в том числе анимированных, текстур, скриптов и т. д.), содержат встроенные ландшафты, шейдерную систему, сочетающую простоту использования, гибкость и производительность. Программирование графики в Unity3d осуществляется средствами JavaScript, Boo (диалект Phyton) и C# на основе .NET; реализована работа с сетью, используется физический движок Ageia PhysX, смешивание 3D-графики реального времени с потоковым аудио и видео. Сервер ресурсов Unity обеспечивает контроль версий в Unity.

Unity3d поддерживает широкий диапазон *платформ*: Windows (XP / Vista / W7), MacOs X, iPhone, iPod, iPad, Xperia PLAY, PS3, Flash 3D player. В 2012 дошла до финальной стадии версия Unity 4.0, в которой список поддерживаемых платформ расширился до iOS, Android, Wii, XBox 360, PlayStation 3, Linux. Поддерживаемые *браузеры*: IE, FireFox, Chrome, Opera, Safari.

Разработчиками выпускаются две версии программного продукта: обычная версия и платная версия Unity3d Pro. Первая отличается ограниченным функционалом, вторая позволяет осуществить все этапы графического конвейера, включая рендер в текстуру, эффекты пост-процесса, удаление из процесса рендеринга невидимых вершин и полигонов.

Инструментарий Unity3d построен на использовании для разработки интерактивных приложений с двух- и трехмерной графикой, обрабатываемой в реальном времени, концепции игрового движка (Game Engine).

Несмотря на специфичность названия, игровые движки широко используются в других типах интерактивных приложений, использующих 3D-графику в реальном времени, таких, как демонстрационные рекламные ролики, архитектурные визуализации, обучающие симуляторы и среды моделирования.

1.2. Концепция Game Engine в Unity3d

Игровой движок (Game Engine) – это центральный программный компонент интерактивных приложений с трехмерной графикой, обрабатываемой в реальном времени, в том числе компьютерных и видеоигр. Он обеспечивает основные технологии моделирования и 3D-визуализации, упрощает процесс разработки проектов, обеспечивает возможность их запуска на нескольких платформах, таких как игровые консоли и настольные операционные системы, например, GNU/Linux, Mac OS X и Microsoft Windows.

Игровой движок обеспечивает основную функциональность пакета Unity3d. Он включает в себя многократно используемые программные компоненты: графический движок («визуализатор»), физический движок, звуковой движок, систему скриптов, анимацию, искусственный интеллект, сетевой код, управление памятью и многопоточность.

В дополнение к многократно используемым программным компонентам, игровые движки, как правило, предоставляют набор визуальных инструментов для разработки проектов. Эти инструменты обычно составляют интегрированную среду для упрощенной, быстрой разработки интерактивных приложений на манер поточного производства, предоставляя гибкую и многократно используемую программную платформу со всей необходимой функциональностью для разработки приложения, сокращая затраты, сложность и время разработки.

Игровые движки обычно платформо-независимы и позволяют разрабатываемому проекту запускаться на различных платформах, включая иг-

ровые консоли и персональные компьютеры, с некоторыми внесенными в исходный код изменениями (или вообще без них).

Часто игровые движки имеют компонентную архитектуру, позволяющую заменять или расширять некоторые подсистемы движка более специализированными (и часто более дорогими) компонентами, например, для симуляции физической природы взаимодействия (*Havok*), звука (*FMOD*) или рендеринга (*SpeedTree*). Однако некоторые игровые движки, такие как *RenderWare*, проектируются как набор слабосвязанных компонентов, которые могут выборочно комбинироваться для создания собственного движка, вместо более традиционного подхода, который заключается в расширении или настройке гибкого интегрируемого решения.

Рассмотрим основные компоненты любого игрового движка – графический и физический движки.

Графический 3D-движок (англ. *3D-graphics engine*; *система рендеринга* или «*визуализатор*») – промежуточное программное обеспечение (ППО), основной задачей которого является визуализация (рендеринг) двух- или трехмерной компьютерной графики. Может существовать как отдельный продукт или в составе игрового движка и использоваться для визуализации отдельных изображений или компьютерного видео. Графические движки, использующиеся в программах по работе с компьютерной графикой (таких, как 3ds Max, Maya, Cinema 4D, Zbrush, Blender), обычно называются «рендерерами», «отрисовщиками» или «визуализаторами». При этом основное и важнейшее отличие «игровых» графических движков от «неигровых» состоит в том, что первые должны обязательно работать *в режиме реального времени*, тогда как вторые могут тратить по несколько десятков часов на вывод одного изображения. Вторым существенным отличием является то, что, начиная приблизительно с 1995-1997 года, графические движки производят визуализацию с помощью графических процессоров (GPU) видеокарт. Программные графические движки используют только центральные процессоры (CPU).

Чаще всего графические 3D-движки или системы рендеринга в игровых движках построены на графическом API, таком как Direct3D или OpenGL, который обеспечивает программную независимость. Использование низкоуровневых библиотек, например, DirectX, SDL и OpenAL позволяет обеспечить аппаратно-независимый доступ к другому аппаратному

обеспечению компьютера, например, устройствам ввода (мышь, клавиатура и джойстик), сетевым и звуковым картам.

Физический движок (англ. *physics engine*) – программный движок, который производит компьютерное моделирование физических законов реального мира в виртуальном пространстве 3D-сцены с той или иной степенью аппроксимации. Чаще всего физические движки используются не как отдельные самостоятельные программные продукты, а как составные компоненты других программ. Выделяют игровые и научные физические движки.

Первый тип используется в компьютерных играх как компонент игрового движка. В этом случае он также должен работать в режиме реального времени, то есть воспроизводить физические процессы в игре с той же самой скоростью, в которой они происходят в реальном мире. Вместе с тем от игрового физического движка не требуется точности вычислений. Поэтому в таких движках используются очень сильные аппроксимации, приближенные модели взаимодействия.

Научные физические движки используются в научно-исследовательских расчетах и симуляциях, где крайне важна именно физическая точность вычислений, а скорость вычислений не играет существенной роли.

Современные физические движки симулируют не все физические законы реального мира, а лишь некоторые, причем с течением времени и прогресса в области информационных технологий и вычислительной техники список «поддерживаемых» законов увеличивается. Современные физические движки, как правило, могут симулировать следующие физические явления и состояния: динамику абсолютно твердого и деформируемого тела, динамику жидкостей и газов, поведение тканей и веревок (тросы, канаты и т. д.).

На практике физический движок позволяет наполнить виртуальное 3D-пространство статическими и динамическими объектами – телами (англ. *body*), указать некие общие законы взаимодействия тел и среды в этом пространстве, в той или иной мере приближенные к физическим, задавая при этом характер и степень взаимодействий (импульсы, силы и т. д). Собственно расчет взаимодействия тел движок и берет на себя. Когда простого набора объектов, взаимодействующих по определенным законам в виртуальном пространстве, недостаточно, в силу неполного приближения

физической модели к реальной, возможно добавлять к телам связи (англ. *joint*, «*соединение*»). Связи представляют собой ограничения объектов физики, каждое из которых может накладываться на одно или два тела. Рассчитывая взаимодействие тел между собой и со средой, физический движок приближает физическую модель получаемой системы к реальной, передавая уточненные геометрические данные в графический движок. Unity3d в качестве физического движка использует Nvidia's PhysX engine.

1.3. Терминология Unity3d

Основной концепцией Unity3d является использование в сцене легко управляемых объектов, которые, в свою очередь, состоят из множества компонентов. Создание отдельных игровых объектов и последующее расширение их функциональности с помощью добавления различных компонентов позволяет бесконечно совершенствовать и усложнять проект. Влияние компонента на поведение или положение того или иного объекта в сцене (*свойства* компонента) определяется с помощью *переменных* компонента.

Ресурсы (Assets) проекта – это строительные/составные блоки всех проектов Unity, в качестве которых могут быть использованы файлы изображений (текстур), 3D-моделей, звуковые файлы, которые будут использоваться при создании в качестве ресурсов. Поэтому в любой папке проекта Unity всегда существует подкаталог с именем Assets, где хранятся все файлы ресурсов.

Когда какой-либо ресурс (например, геометрическая 3D-модель) используется в сцене игры, он становится в терминологии Unity *игровым объектом* (Game Object). Все эти объекты изначально имеют хотя бы один компонент, задающий его положение в сцене и возможные преобразования (компонент Transform). Переменные компонента Transform определяет положение (position), поворот (rotation) и масштаб (scale) объекта в его локальной декартовой прямоугольной системе координат X, Y, Z. Наличие переменных у каждого компонента обусловливает возможность обращения к ним из соответствующей программы (скрипта).

Компоненты (components) в Unity3d имеют различное назначение – они могут влиять на поведение, внешний вид и многие другие функции объектов, к которым прикрепляются (attaching). Unity предоставляет множество компонентов различного назначения.

Для обеспечения интерактивности различных 3D-приложений в Unity3d используются *скрипты*, которые также рассматриваются средой как компоненты. Помимо JavaScript, Unity3d также предоставляет возможность использовать для написания скриптов языки C# и Boo (производный от языка Python). Для написания скриптов можно воспользоваться встроенным редактором Unity3d *MonoDevelop*.

2. Интерфейс Unity3d. Создание простейших моделей твердых тел. Добавление массы, гравитации к твердому телу

Для создания нового проекта необходимо запустить среду Unity3d и в открывшемся окне создать новый проект (***File → New Project***, рис. 1).

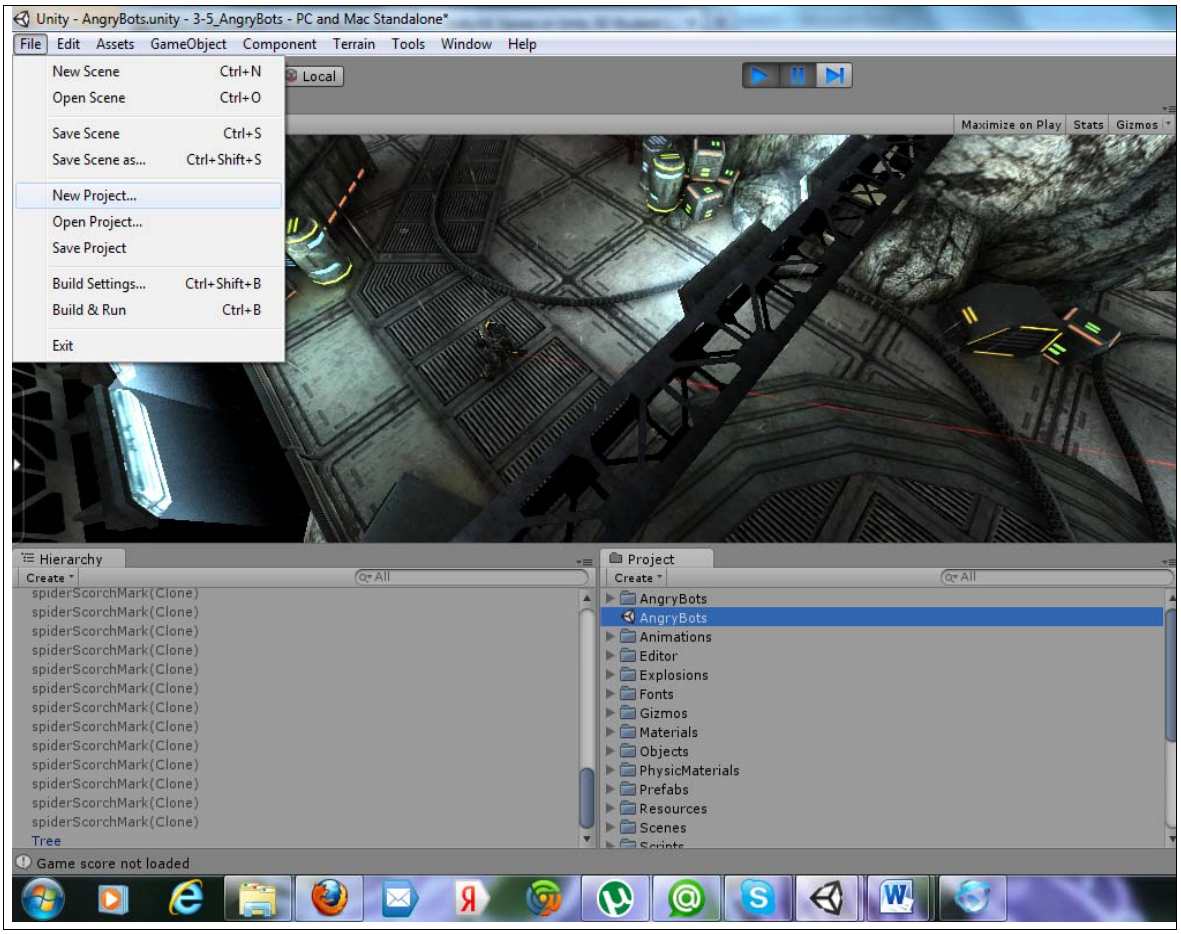


Рис. 1. Окно создания нового проекта в Unity3d

Unity3d содержит набор базовых пакетов с различными физическими свойствами. Для создания стандартных геометрических моделей (куб, сферы и т. д.) и затем задания гравитационных свойств этим объектам нам потребуются пакеты *PhysicMaterials.UnityPackage* и *StandartAssets (Mobile).UnityPackage*. Для добавления пакетов в новый проект в открывшемся окне (рис. 2) выбираем вкладку *Create New Project*, указываем путь к текущей рабочей папке с проектом и отмечаем необходимые пакеты.

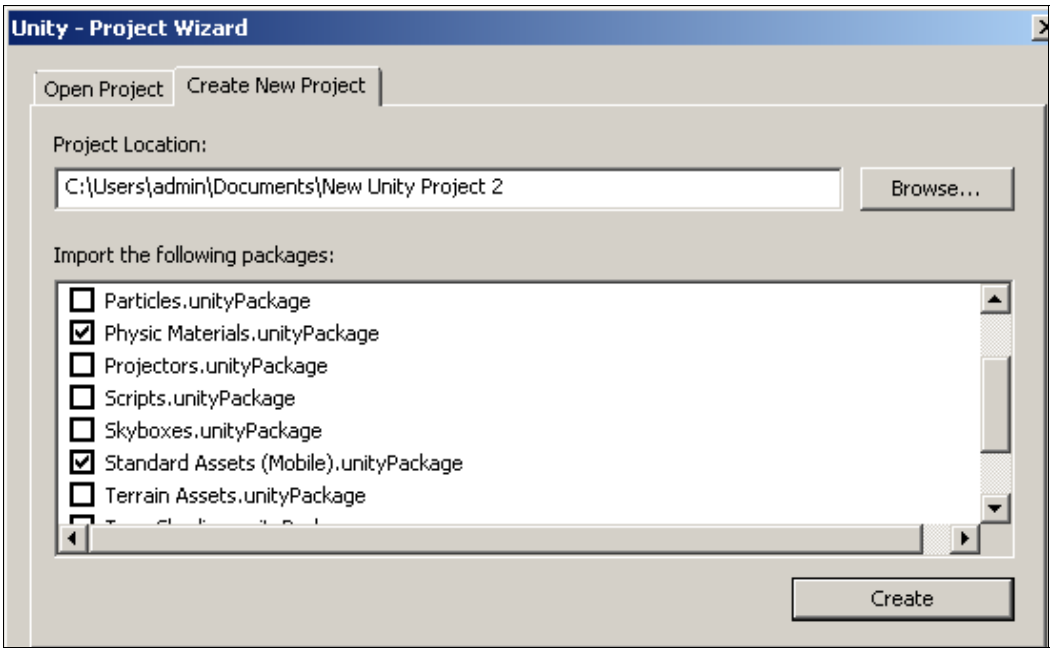


Рис. 2. Окно выбора стандартных пакетов

Рассмотрим окно пустого проекта в Unity3d и его основные элементы (рис. 3).

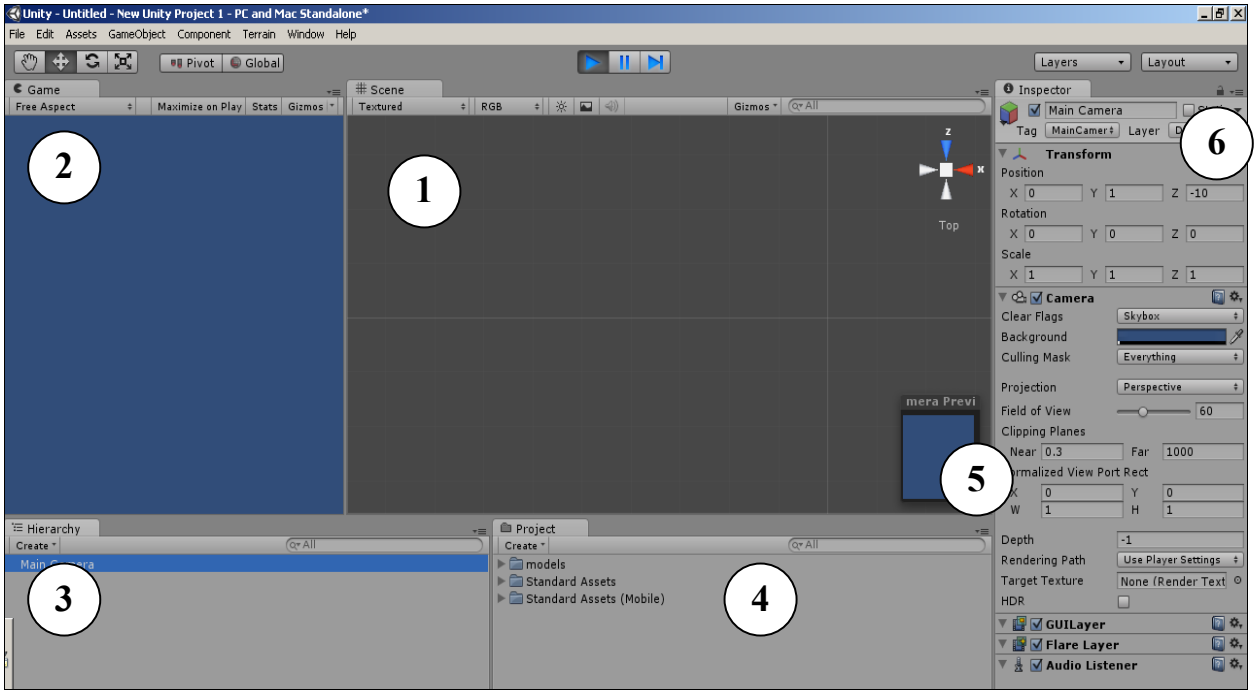


Рис. 3. Окно пустого проекта в Unity3d: 1 – окно сцены (*Scene*); 2 – окно запуска и просмотра игры (*Game*); 3 – окно иерархии объектов (*Hierarchy*); 4 – окно инспектора префабов и ресурсов (*Project*); 5 – окно предварительного просмотра сцены (*Camera Preview*); 6 – окно инспектора компонентов и их свойств (*Inspector*)

Интерфейс Unity3d является гибко настраиваемым под нужды разработчика. Так, при выделении в любом элементе окна текущей вкладки с помощью контекстного меню, она может быть дополнена новой (*Add Tab*) или удалена (*Close Tab*) (рис. 4).

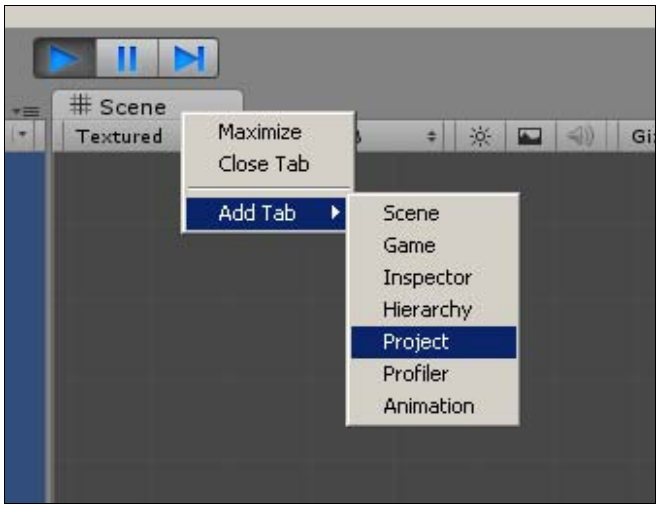


Рис. 4. Настройка вкладок Unity3d в рабочем окне проекта

Инспектор префабов и используемых ресурсов (Project) содержит ссылки на файлы добавленных в проект моделей, текстур, звуков, скриптов, а также специальные объекты – префабы, предназначенные для дальнейшего повторного использования объектов в проекте.

Иерархия объектов (Hierarchy) – это список всех объектов на текущем уровне, показывающий также их отношения наследования (*Parent-Child*).

Инспектор компонентов и их свойств (Inspector) характеризует выделенный в данный момент объект – модель, текстуру, префаб и т. д., отображаемый в окне иерархии объектов (*Hierarchy*).

Добавление к проекту стандартного геометрического объекта Unity3d осуществляется на вкладке ***GameObject*** в главном меню. В качестве простейших объектов в сцену проекта могут быть добавлены геометрические объекты (куб, сфера, цилиндр и т. д.), источники освещения, камера.

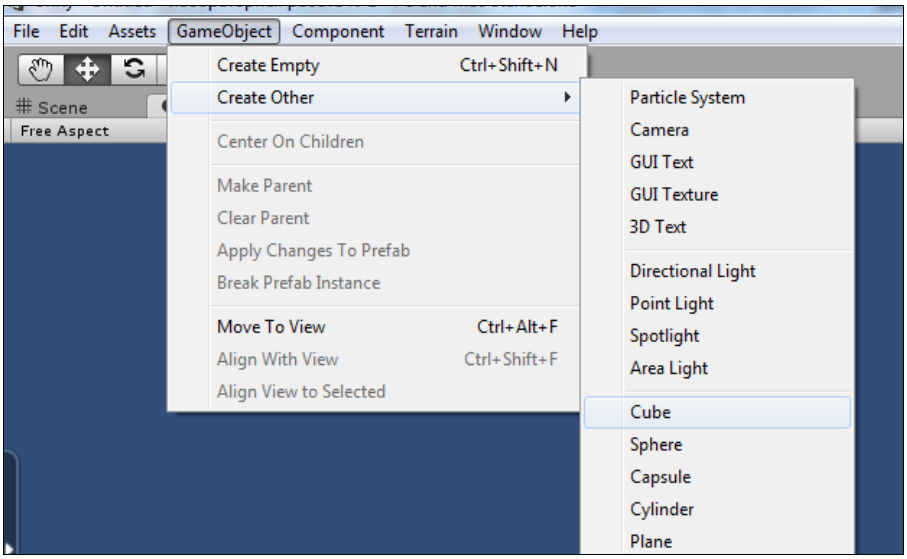
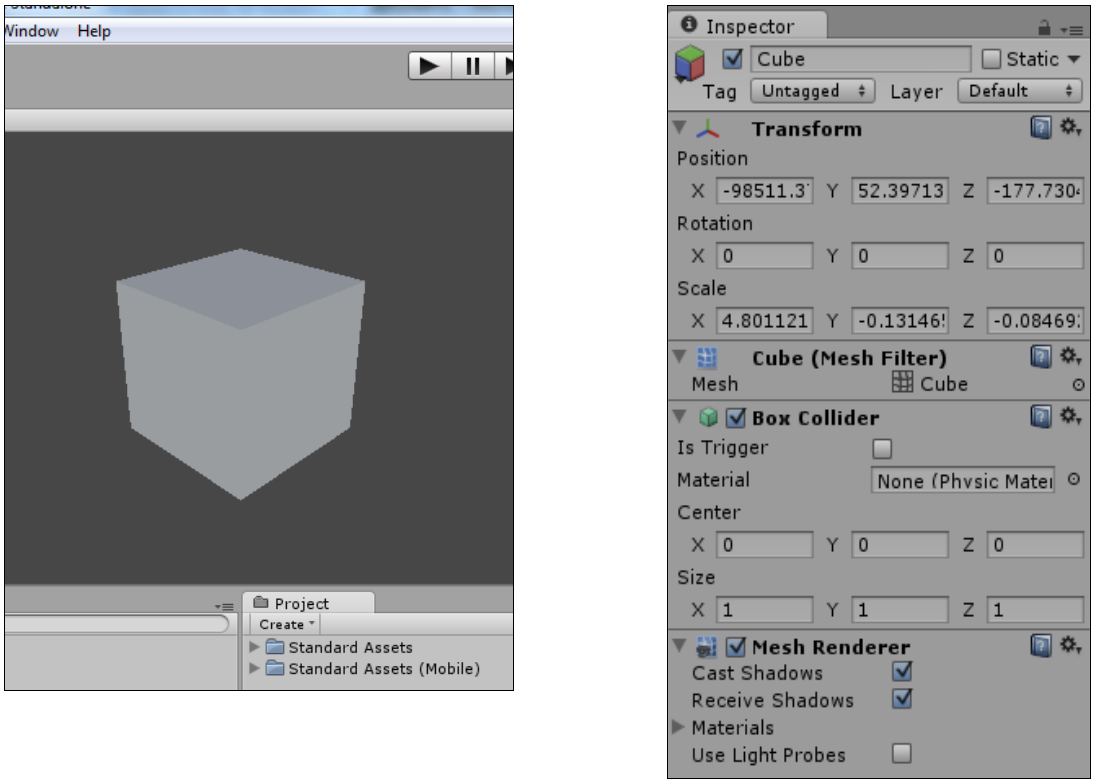


Рис. 5. Добавление к проекту простейшего геометрического объекта

Выберем на этой вкладке пункт ***Cube*** для создания в текущей сцене объекта «куб» (***GameObject → Create Other → Cube***, рис. 5), изображение которого в трехмерной системе координат показано на рис. 6, а.



а) б)

Рис. 6. Создание простейшего объекта: а) отображение объекта в трёхмерной системе координат; б) панель инспектора свойств объекта со списком компонентов

Рассмотрим, что представляет собой только что созданный геометрический объект. Если выделить его в окне иерархии объектов или в окне сцены, то в окне инспектора мы увидим список связанных с объектом компонентов (рис. 6, б).

Transform – это компонент, отвечающий за положение объекта в пространстве в мировой системе координат и имеющийся у любого игрового объекта. Он хранит в себе *переменные* – координаты объекта в трехмерном пространстве *x, y, z*; данные о повороте его относительно осей координат, а также *методы* для изменения этих параметров. Также он отвечает за иерархию объектов на сцене: свойство *parent* может содержать ссылку на другой аналогичный компонент Transform, являющийся родительским по отношению к данному. Если родительский компонент при этом двигается или вращается, вместе с ним двигается и вращается и дочерний. Дочерний же может свободно двигаться и вращаться, при этом его перемещения не оказывают влияние на родительский компонент. Чтобы лучше понять этот момент, посмотрите на свою руку: когда вы двигаете ладонью, пальцы перемещаются вместе с ней, а шевеление пальцами ни в коей мере не затрагивает ладонь. В этой ситуации ладонь является родительским элементом для пальцев.

Mesh filter – компонент, хранящий в себе трехмерную модель объекта, в данном случае – это кубический примитив (рис. 6, б).

Mesh Renderer – компонент, который дополняет геометрические данные о трехмерной модели из вышеуказанного компонента ***Mesh filter*** текстурой (текстурные координат) и применяет шейдеры, в результате чего мы получаем на экране полноценную модель такой, как она должна выглядеть. Также этот компонент позволяет включить/отключить свойство отбрасывания теней объектом и на объект.

Box Collider – компонент, хранящий в себе трехмерную модель «коллизий», то есть ту модель, по которой физический движок рассчитывает столкновения между объектами или со средой. Модель, указанная в данном случае как *Box Collider* (рис. 6, б), отображается на сцене в виде зеленого каркаса. Варьируя переменными *x,y,z* центра масс (*Center*) и его размера (*Size*), возможно существенным образом менять характер взаимодействия объектов.

Для того чтобы созданный куб имел под собой основание в виде плоской поверхности, необходимо создать для него объект «плоскость» (***GameObject → Create Other → Plane***) (рис. 7).

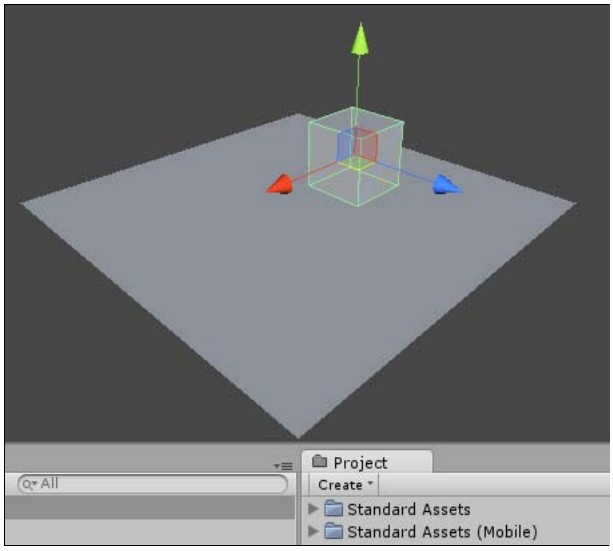


Рис. 7. Создание плоскости

Как видно из рисунка, цвет куба теряется в пространстве сцены, т. е. сливается по цвету с вновь созданным объектом плоской поверхности. Для исключения этого необходимо создать для объекта освещение. Добавление к проекту точечного освещения объектов также осуществляется в том же меню (***GameObject → Create Other → Point Light***) (рис. 8).

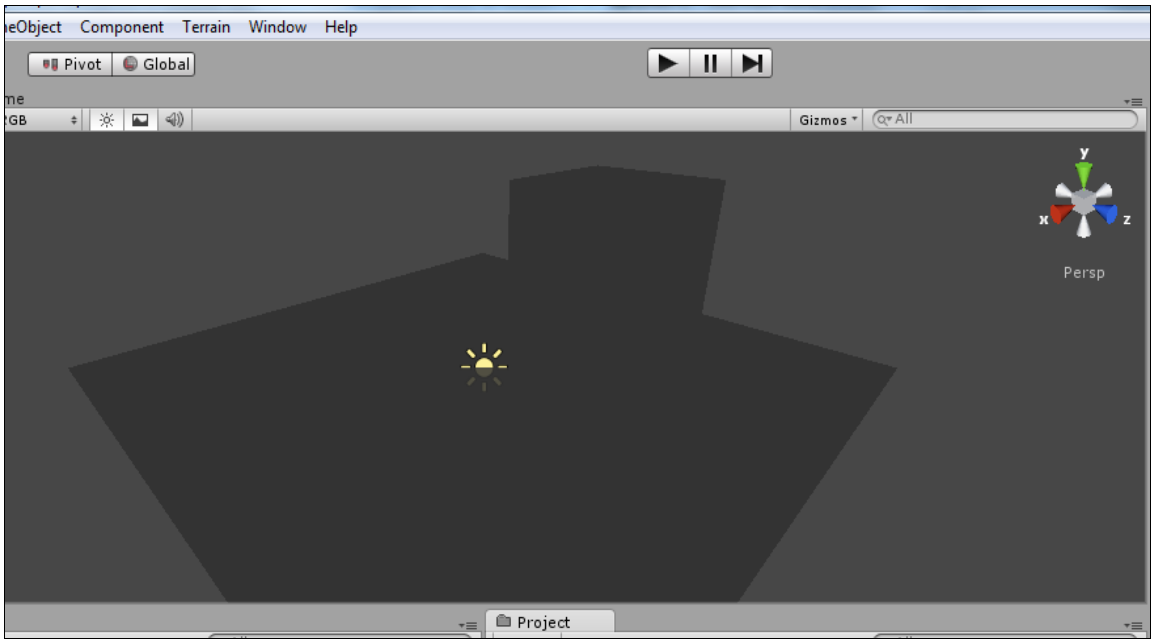


Рис. 8. Добавление источника света в пространство сцены

Теперь при нажатии управляющей кнопки *Play* в окне среды Game мы пока не сможем увидеть созданные ранее объекты. Для того чтобы воплотить наш проект в виртуальную жизнь, в сцене не достает еще одного объекта – камеры.

Для того чтобы разместить камеру не просто в «пространстве сцены», а на конкретной плоскости, необходимо в окне иерархии созданных объектов проекта указать созданную нами ранее плоскость, после чего создать камеру (***GameObject → Create Other → Camera***, рис. 9):

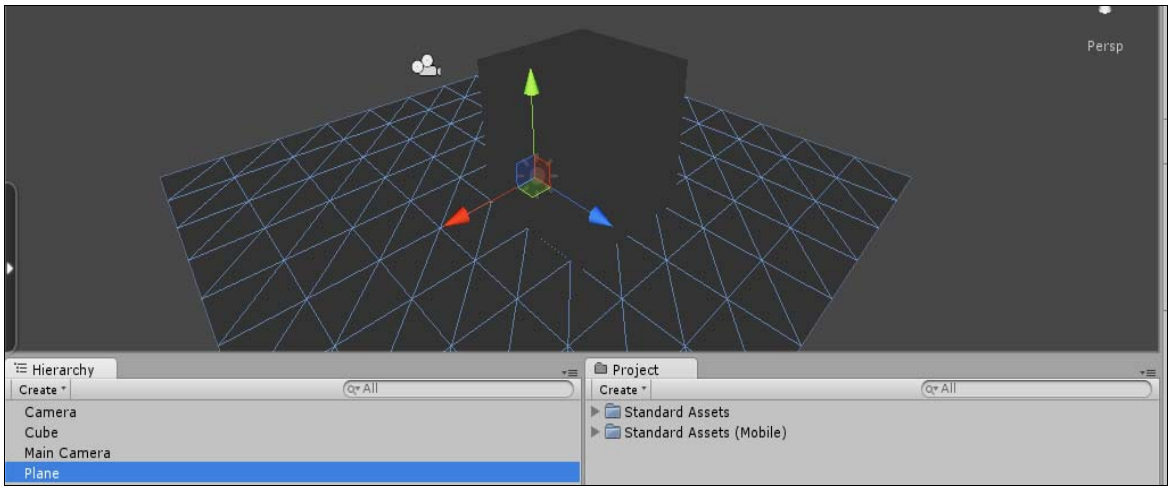


Рис. 9. Добавление камеры

Воспользовавшись кнопкой перетаскивания объектов  , расположим их примерно так, как это показано на рис. 10. При этом предварительное изображение с камеры сцены соответствует изображению на рис. 11.

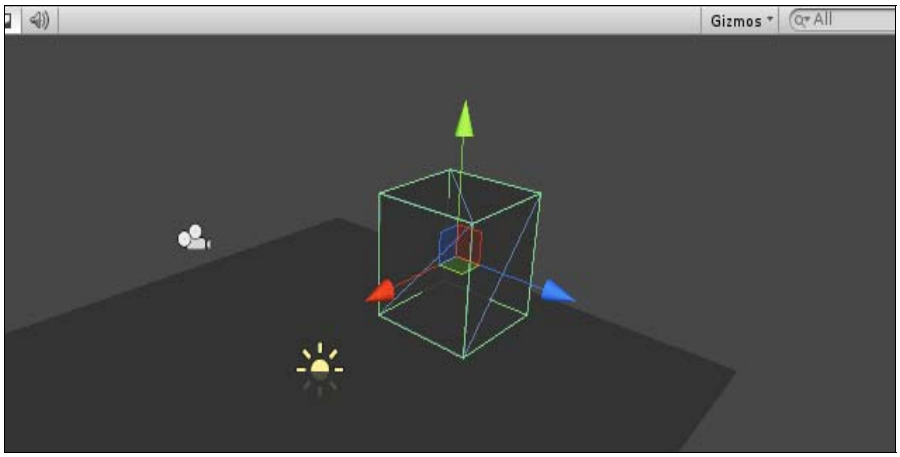


Рис. 10. Расположение объектов на сцене

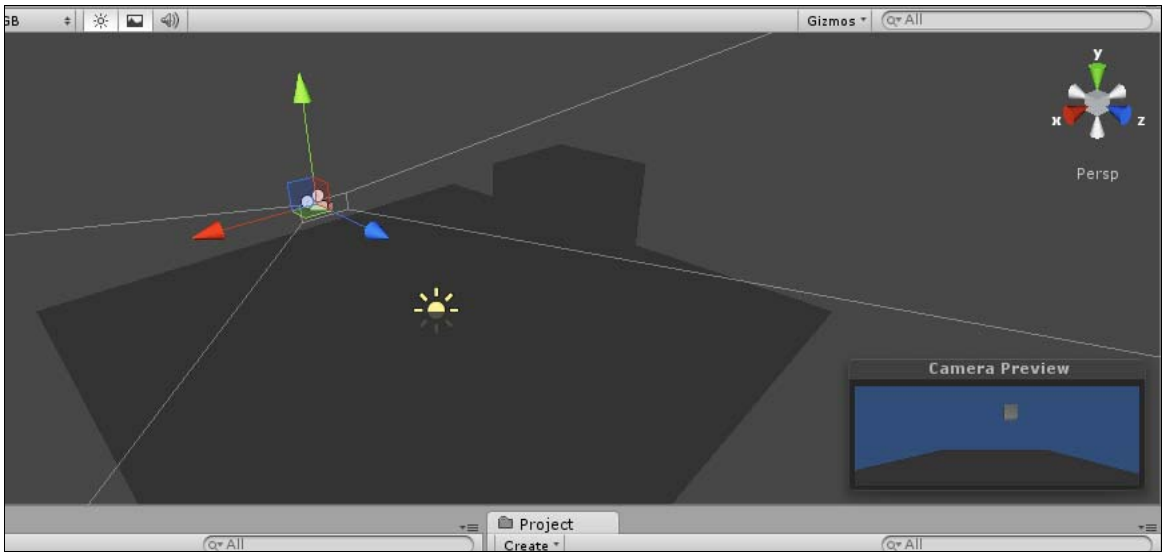


Рис. 11. Предварительное расположение камеры сцены

Теперь при нажатии управляющей кнопки «*Play*» в окне проекта *Game* отобразится статическая сцена, в которой кубический объект нависает над плоскостью (рис. 12).

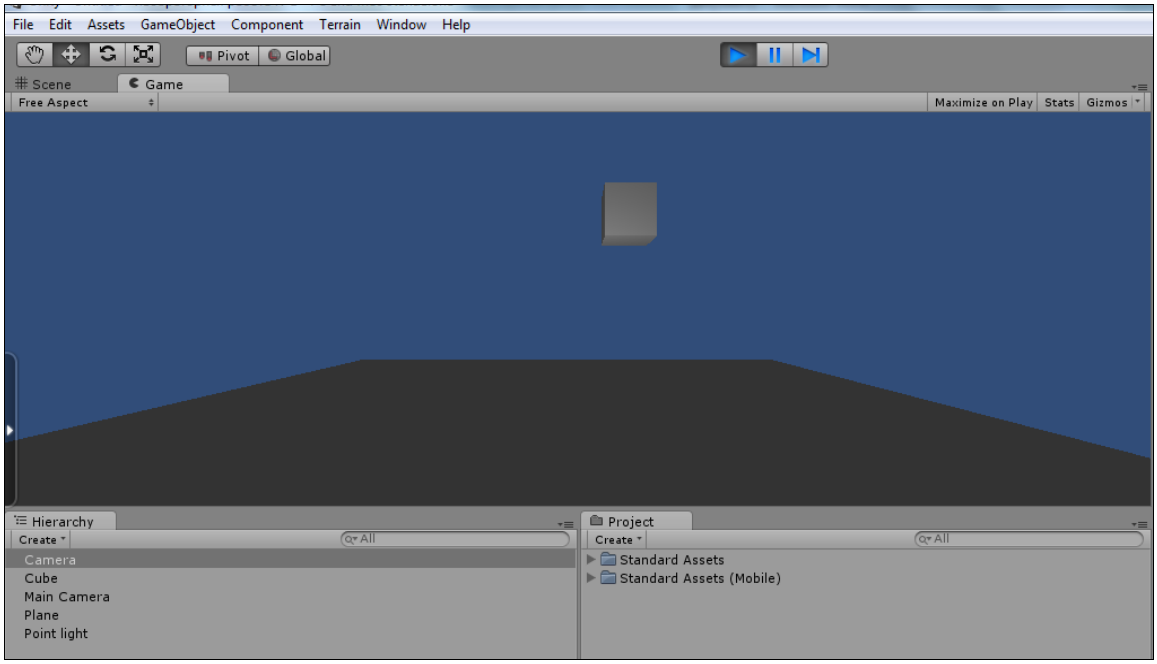


Рис. 12. Результат запуска в окне проекта

Для установления взаимодействия между созданными геометрическими объектами необходимо добавить к ним соответствующие физические свойства. Физический движок среды используют систему динамики твердых тел для создания реалистичного движения. Это означает, что вме-

сто статичных объектов, находящихся в виртуальном пространстве сцены, мы имеем объекты, у которых могут быть такие свойства, как масса (*Mass*), гравитация (*Use Gravity*) и другие. Так, свойства массы или гравитации добавляются к объекту с помощью компонента ***Rigidbody*** (***Component → Physics → Rigidbody)***. В соответствующих полях переменных этого компонента *mass* и *drag* в окне инспектора компонентов выставляются величины массы и скорости падения (рис. 13), при этом в окне иерархии куб должен оставаться выделенным.

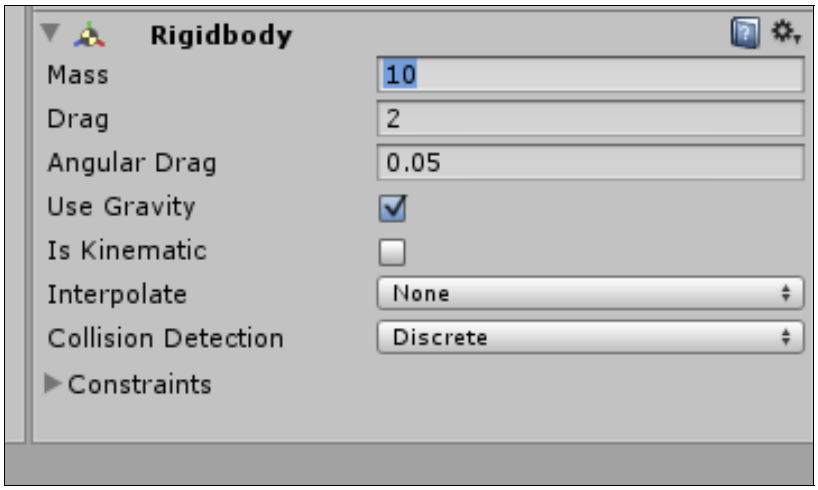


Рис. 13. Настройка свойств компонента Rigidbody

Теперь при запуске проекта во вкладке *Game* можно увидеть, как с высоты на созданную плоскость падает куб под действием силы тяжести. Причем падение это после соприкосновения с плоскостью прекращается, и куб остается на плоскости.

Изменение физических свойства куба можно осуществить в компоненте *BoxCollider*, одним из параметров которого является материал (*Material*). По умолчанию значение этого параметра выставляется в значении *None*.

Поскольку при создании проекта был подключен стандартный набор ресурсов *PhysicMaterials.UnityPackage*, то в проекте появляется возможность использования нескольких стандартных физических материалов Unity3d. Указание в качестве параметра *упругого* материала (*Bouncy*) позволяет добиться отскока куба от плоской поверхности за счет указанного физического свойства.

Лабораторная работа № 1

Цели работы:

– знакомство с особенностями моделирования физических свойств трехмерных объектов в среде Unity3d;

– изучение физических свойств трехмерного тела (модели) в среде с помощью компонента Rigidbody (твердое тело);

– освоение приемов организации взаимодействия объектов за счет добавления гравитации и массы к 3d-объекту.

Задание к лабораторной работе

1) Познакомьтесь с особенностями моделирования простейших трехмерных геометрических моделей в среде Unity3d (по п. 2 методических указаний).

2) Освойте приемы моделирования трехмерных геометрических объектов и их физических свойств с помощью компонентов Transform, Rigidbody, BoxCollider в новом проекте в среде Unity3d.

3) Освойте приемы организации взаимодействия объектов за счет добавления гравитации и массы к 3d-объектам в этом проекте.

4) Сохраните файл и проекта по п. 2 и предоставьте на проверку преподавателю.

Контрольное задание

Создать физическую модель падающего шара, скатывающегося по наклонной поверхности. Последовательность моделирования описать в плане-отчете по лабораторной работе.

Контрольные вопросы

1) Каковы основные элементы рабочего окна среды Unity3d?

2) В чем сущность использования принципа родительских и дочерних компонентов?

3) Как изменится взаимодействие объектов в сцене при увеличении переменной *Size* компонента *BoxCollider*?

4) Объясните назначение параметра *Is Kinematic* компонента *Rigidbody.*

5) Объясните назначение параметра *Angular Drag* компонента *Rigidbody.* Для чего предназначено свойство *Cast Shadows* компонента *Mesh Renderer?*

3. Основы взаимодействия (столкновения) между объектами. Применение скриптов на языке C#

Большое значение при организации взаимодействия 3d-объектов в пространстве имеет обнаружение столкновений (*Collision detection*). *Collision detection* – это способ, с помощью которого анализируется 3D-пространство сцены на предмет столкновений между объектами. Присваивая объекту компонент *Collider*, мы фактически размещаем вокруг него невидимую сетку – так называемый коллайдер, который имитирует форму объекта и информирует о наличии столкновения с другим объектом.

Например, в игре-симуляторе боулинга шары будут иметь простую сферическую форму коллайдера (*Sphere collider*), в то время как у объектов-кеглей коллайдер будет иметь форму цилиндра/капсулы или, для большей реалистичности столкновений, будет использовать меш (*mesh*), который является не чем иным, как описанием геометрии 3d-модели. Информация о столкновении коллайдеров поступает в физический движок, который сообщает столкнувшимся объектам их дальнейшую реакцию на это столкновение, основанную на направлении и силе удара, скорости и других факторах. Отметим, что использование коллайдеров, повторяющих форму меша модели, с одной стороны, дает более точное определение столкновений, но в то же время приводит к увеличению затрат на их вычисление.

Рассмотрим особенности столкновений объектов как с использованием непосредственного функционала *Collision detection*, предоставляемого Unity3d, так и с помощью программирования такого взаимодействия на языке C#.

Для этого добавим в созданную ранее сцену новый кубический объект, выполняющий роль некоторого препятствия для падающего куба, смоделированного ранее (***GameObject → Create Other → Cube***), и придадим ему форму параллелепипеда (рис. 14).

Для изменения его формы можно воспользоваться свойствами компонента *Transform* на панели инспектора компонентов, либо с помощью инструмента масштабирования «Scale»  , предварительно выбрав геометрический объект. При этом в сцене можно увидеть три разноцветных

квадратных куба по разным сторонам объекта (рис. 15), выполняющих роль узлов для изменения его размера в направлении, соответствующем осям координат сцены. Центральный куб позволяет изменять размер одновременно по всем осям координат (рис. 15).

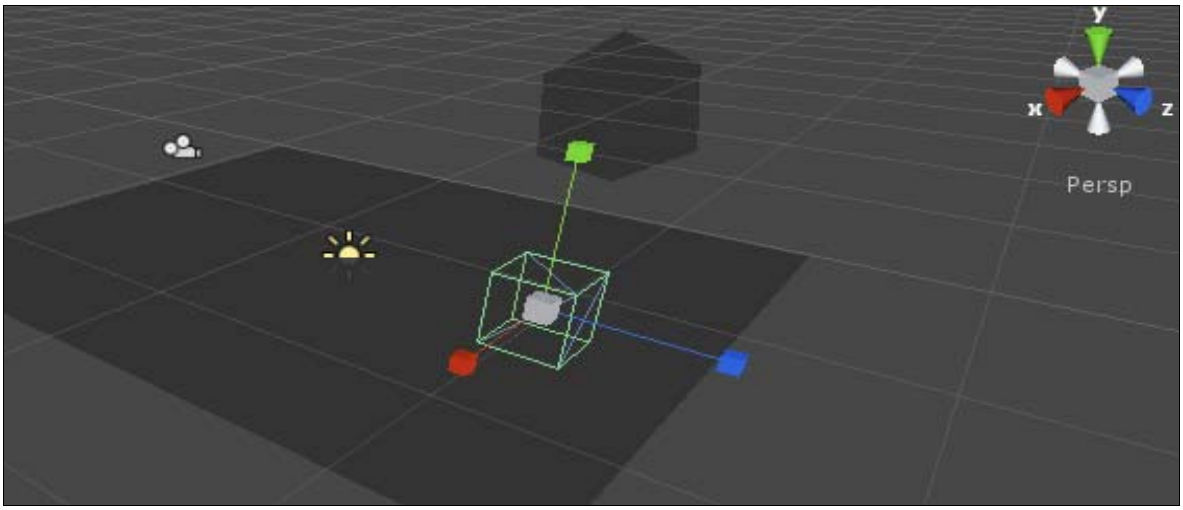


Рис. 15. Изменение размеров объекта в Unity3d

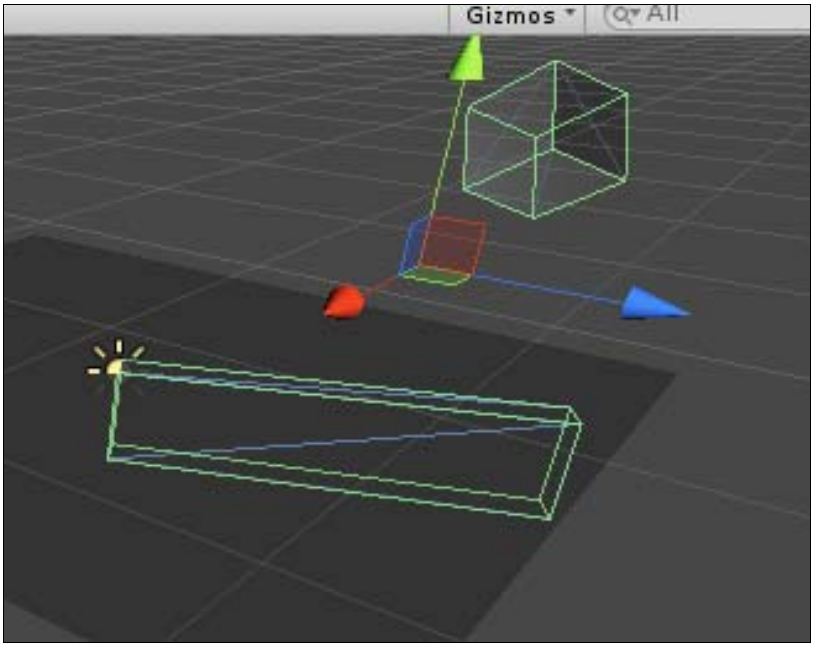


Рис. 14. Создание параллелепипеда на основе объекта «куб»

Чтобы лучше увидеть картину взаимодействия куба и прямоугольного препятствия, необходимо развернуть исходное положение падающего куба – на ребро. Вращение выбранного объекта (куба) осуществляется в

Unity3d с помощью инструмента «Rotate»  . При активации этого инструмента вокруг объекта появляется своеобразная сфера, определяющая углы его вращения в трехмерном пространстве. Захватывая и перемещая одну из ее сторон (рис. 16), можно вращать объект произвольным образом.

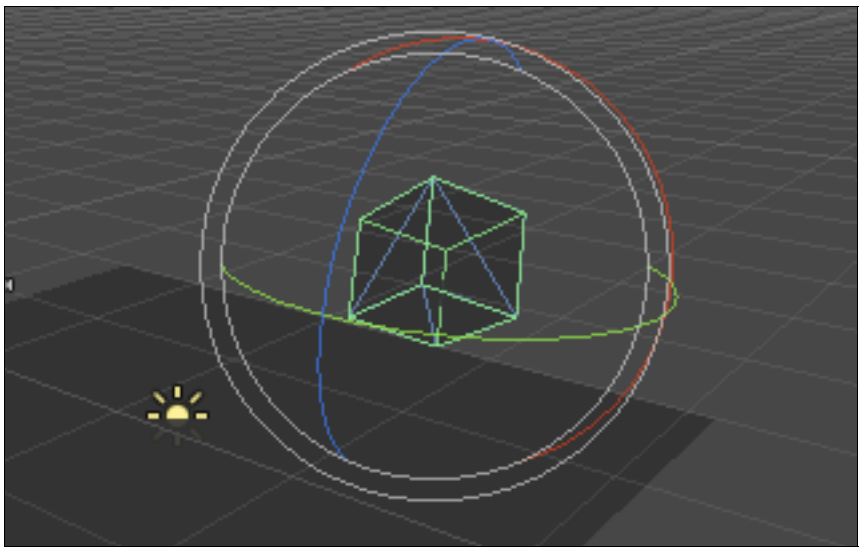


Рис. 16. Вращение объектов в Unity3d

Далее переключившись в режим просмотра *Game*, можно наблюдать сцену взаимодействующих в ней объектов (рис. 17).

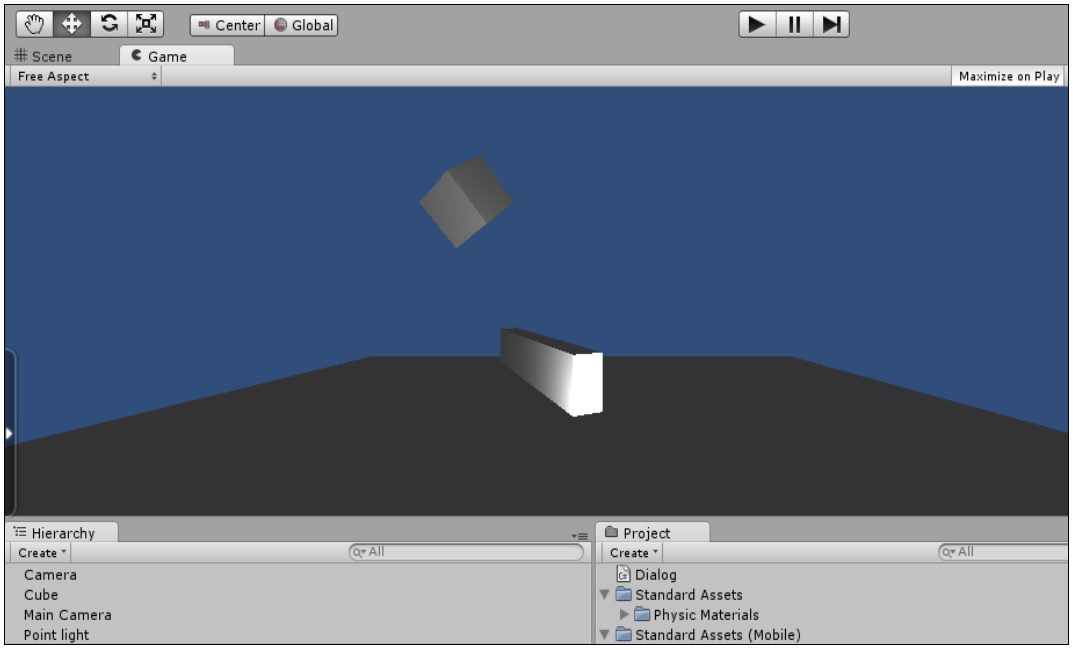


Рис. 17. Взаимодействующие объекты в сцене

Для определения факта столкновения объектов в Unity3d необходимо отличать эти объекты по их названию. Переименовать объект «Plane» (Плоскость) в объект «Zemlya», а параллелепипед, представляющий стену (препятствие), в объект «Stena», можно непосредственно выбрав объект в окне иерархии и применив к нему команду «Rename» из контекстного меню (рис. 18), после чего задать новое имя объекта. Отметим, что при разработке проектов в Unity3d разрешается ввод русскоязычных символов, однако при работе со скриптами необходимо использовать латиницу.

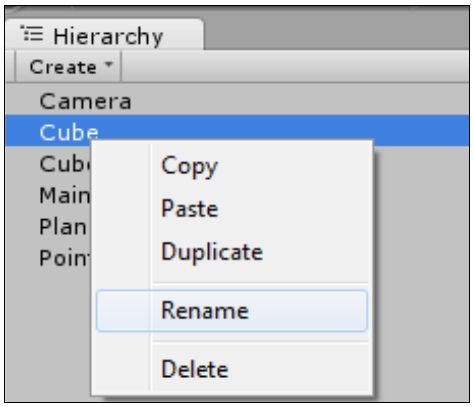


Рис. 18. Переименование объектов

Для того чтобы заставить взаимодействовать между собой имеющиеся в сцене трехмерные модели, создадим скрипт на языке программирования C# и назовем его «Dialog» (***Project → Create → C# Script)***.
Двойной щелчок позволяет открыть скрипт в редакторе скриптов *MonoDevelop*
(рис. 19).

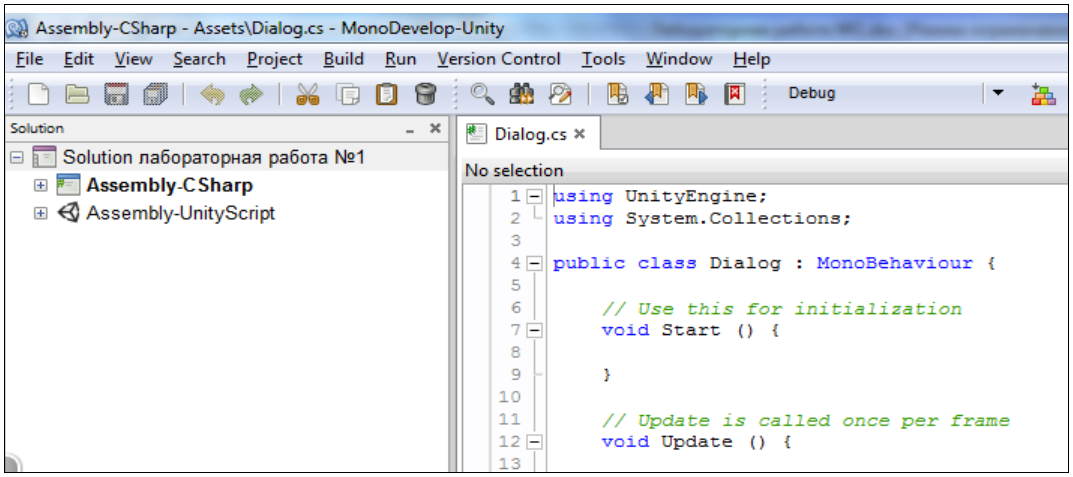


Рис. 19. Окно редактора скриптов Unity3d *MonoDevelop*

Создание скриптов – один из наиважнейших моментов в разработчике игр. Код, написанный для использования в Unity, опирается на ряд готовых встроенных классов, о которых вы должны думать как о библиотеках команд или поведения. При написании скриптов вы будете создавать свои собственные классы, на основании существующих в Unity Engine.

При создании нового C#-скрипта в Unity3d автоматически создается его каркас, который на первых порах программирования может не понадобиться, и все его содержимое можно удалить.

Рассмотрим следующий код:

```
using UnityEngine;
using System.Collections;

public class Dialog : MonoBehaviour {
// Метод-функция, вызываемая при столкновении объектов
void OnCollisionEnter() {
  Debug.Log("Hit Something");  // Передаем сообщение в кон-
соль Unity
  }
}
```

Первые две строки подключают к скрипту используемые пространства имен. Далее необходимо запомнить, что главным классом в Unity3d является MonoBehaviour. Любой пользовательский скрипт (в описанном случае это *Dialog)* должен быть его наследником, и не спроста – ведь именно этот класс реализует интеграцию всех объектов в основной цикл программы. Именно это наследование позволяет пользовательскому скрипту (классу) выполнять роль компонента и быть привязанным к игровому объекту.

Здесь метод «OnCollisionEnter» определяет столкновение объекта с другими объектами. А статический метод «Log» класса «Debug» пишет сообщение "`Hit Something`" в консоль Unity.

После сохранения скрипта добавляем его в качестве компонента для падающего куба.

Для этого необходимо сначала выбрать соответствующий объект в окне иерархии и перетащить на него вновь созданный скрипт «Dialog».

При этом необходимо обратить внимание на то, что добавленный скрипт также отображается внизу в окне «Inspector» в качестве компонента объекта, к которому он привязан

Теперь, перейдя в режим Play, можно наблюдать, что в тот момент, когда созданный объект куб коснется плоской поверхности, в консоли среды Unity3d (***Window → Console***) появляется соответствующее сообщение. Заметьте, что такое сообщение будет выдаваться при каждом столкновении объектов. Причем последнее консольное сообщение отображается в статус (внизу окна) (рис. 20).

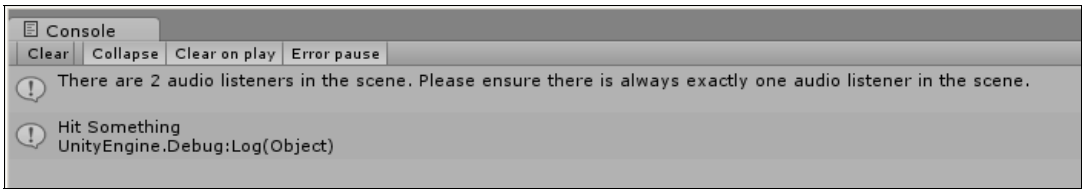


Рис. 20. Просмотр сообщений консоли Unity3d

Для выяснения того, с какими именно объектами столкнулся исходный объект, необходимо использовать значение параметра класса «Collision», которое будет принимать метод «OnCollisionEnter».

Открываем редактор скрипта и вставляем в него следующий код:

```
// Теперь метод принимает объект класса Collision, с кото-
рым происходит столкновение
void OnCollisionEnter(Collision myCollision) {
  // определение столкновения с двумя разноименными объек-
тами
  if (myCollision.gameObject.name == "Zemlya") {
      // Обращаемся к имени объекта, с которым столкнулись
      Debug.Log("Stolknulsya s Zemlei");
    }
    else if (myCollision.gameObject.name == "Stena") {
      Debug.Log("Stolknulsya so Stenoi ");
    }
  }
```

После открытия консоли (***Window → Console***) мы увидим, с какими именно объектами в сцене столкнулся куб (рис. 21).Таким образом, Uni-

ty3d позволяет нам оценить возможности взаимодействия объектов внутри среды.

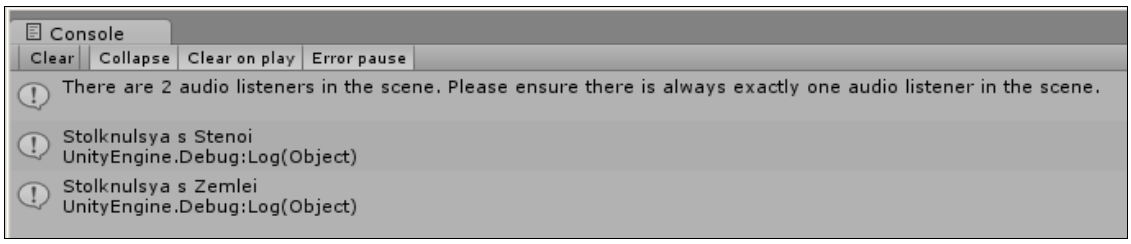


Рис. 21. Просмотр сообщений о столкнувшихся объектах в консоли Unity3d

Лабораторная работа № 2

Цели работы:

– знакомство с особенностями разработки и внедрения скриптов на языке программирования C#;

– изучение основ взаимодействия трехмерных моделей с использованием функций и применением скриптов;

– освоение приемов организации взаимодействия объектов за счет столкновений (collisions) между 3d-объектами.

Задание к лабораторной работе

1) Познакомьтесь с особенностями разработки и внедрения скриптов на языке программирования C# в Unity3d.

2) Изучите способы организации взаимодействия трехмерных моделей с использованием функций и скриптов Unity3d.

3) Освойте приемы организации взаимодействия объектов за счет столкновений между объектами на основе прикрепления скрипта на языке C#.

4) Сохраните файл проекта по п. 3 и представьте преподавателю.

Контрольное задание

Измените скрипт *Dialog* на языке C# так, чтобы при столкновении куба со стеной происходило разрушение стены. Составить план-отчет по лабораторной работе.

Контрольные вопросы

1. Чем отличаются функции *Destroy (gameObject)* от *Destroy (collision.gameObject)*?

2. Объясните назначение процедуры *OnCollisionEnter*?

3. Каким образом установить связь «родитель-потомок» для произвольных объектов?

4. Префабы. Копирование и удаление объектов среды в Unity3D. Создание префабов с применением скриптов C#

Префаб (*Prefabs*) – это конструкция подготовленных объектов и компонентов, предназначенная для их многократного использования в проекте. Экземпляр префаба может быть добавлен в любое количество сцен, а также многократно в одну сцену. Все экземпляры являются ссылками на оригинальный префаб и, фактически, его «клонами»; имеют те же свойства и компоненты, что и оригинальный объект.

Префабы полезны, когда нужно создать несколько экземпляров сложного объекта. Альтернативным способом реализации многократного использования объектов и компонентов в проекте является создание новых объектов с помощью скрипта (рис. 22):

one ▸ No selection

```
1  using UnityEngine;
2  using System.Collections;
3
4
5  public class one : MonoBehaviour {
6
7      void Start() {
8        for (var y = 0; y < 3; y++) {
9         for (var x = 0; x < 3; x++) {
10        GameObject cube = GameObject.CreatePrimitive(PrimitiveType.Cube);
11        cube.transform.position = new Vector3(x, y, 0);
12            }
13    }
14    }
15 }
```

Рис. 22. Простейший скрипт для создания 9 кубических объектов

Описанный выше код добавляет объекты без учета их компонентов, таких как, например, Rigidbody. Рассмотрим, каким образом можно создать множество таких кубов с уже добавленными компонентами, используя конструкцию префаба.

Прежде всего необходимо добавить префаб в проект (***Project → Create → Prefab***). В результате на панели *Project* в окне проекта появится префаб с именем «*New Prefab*». Переименуем его в «*UprugostCube*». Обратите внимание, что на данном этапе ссылка на префаб на панели *Project* отображается серым цветом. Для того, чтобы добавить объекты к префабам, достаточно выбрать объект в окне иерархии объектов и перетянуть его на соответствующий префаб («*UprugostCube*») на панели *Project*. Свойства созданного префаба «UprugostCube» описываются на панели инспектора, а его предварительный вид доступен в окне «Preview» (рис. 23).

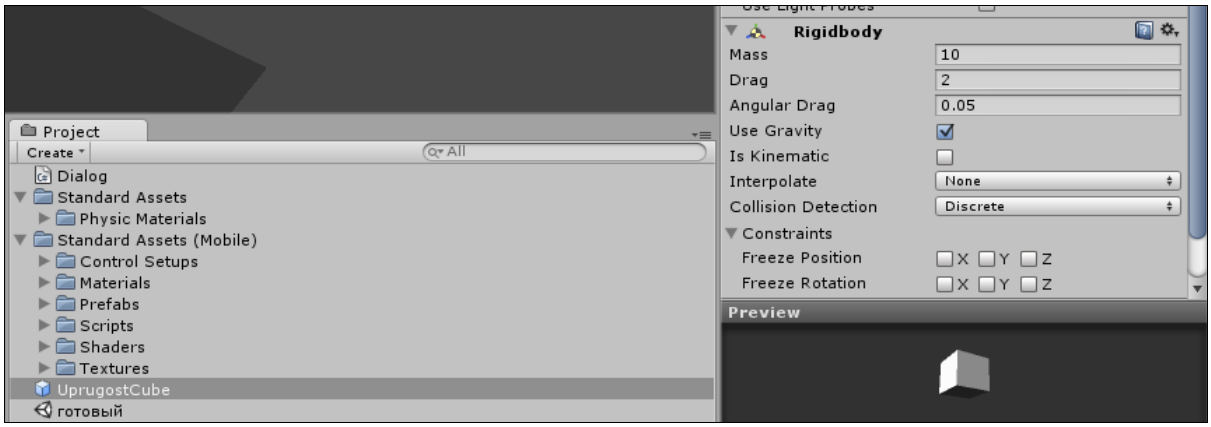


Рис. 23. Просмотр свойств префаба «UprugostCube»

Для создания пары экземпляров префаба на плоскости необходимо просто его перетащить. В результате на плоскости появятся два куба, а в окне «иерархии» добавится объект с именем соответствующего префаба «*UprugostCube*» (рис. 24).

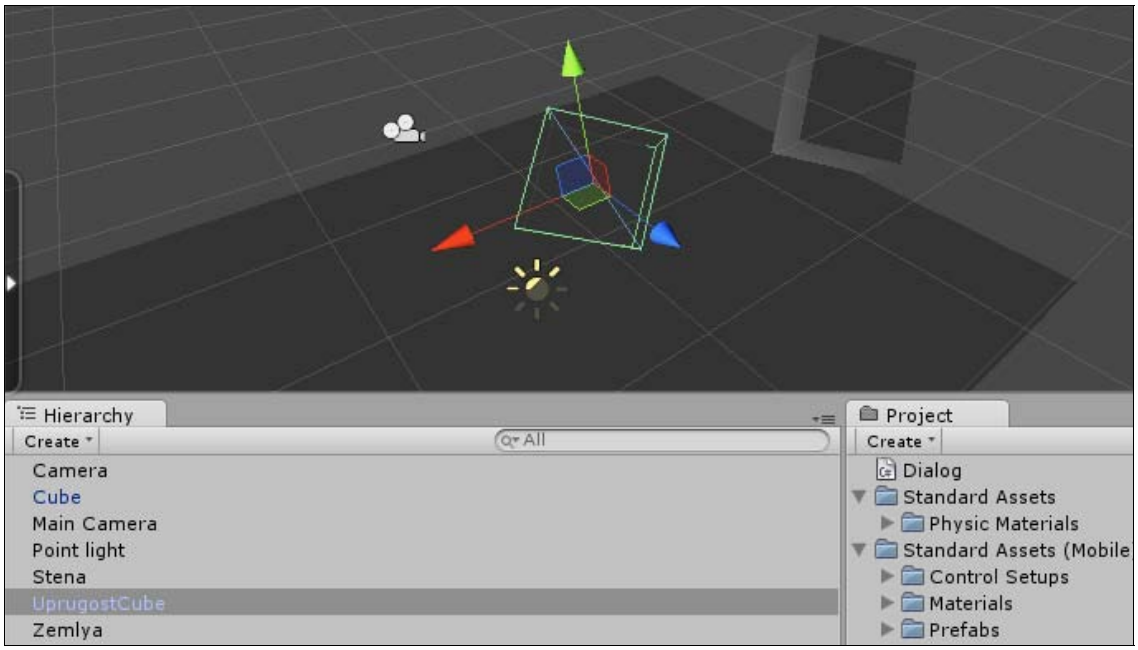


Рис. 24. Добавление экземпляров префаба в сцену

Переключившись в режим просмотра «Game» и запустив сцену, можно увидеть, что добавленный куб ведет себя точно так же, как и оригинальный, обладая одинаковыми физическими свойствами (рис. 25).

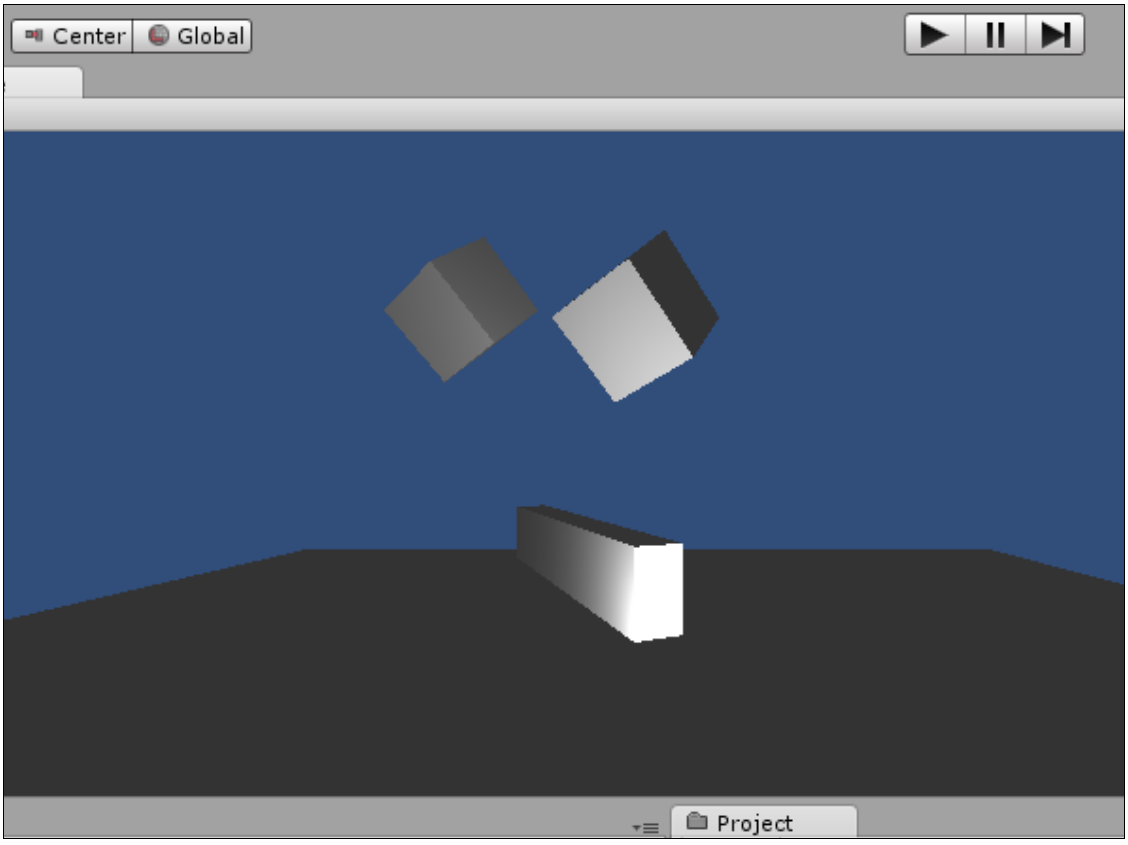


Рис. 25. Сцена с экземплярами префаба

Рассмотрим, каким образом можно удалять объекты в Unity3d в скриптах на языке C#. Для этого в окне «Project» создадим новый скрипт с именем *«Destroy»* и откроем его в редакторе скриптов *MonoDevelop.*

Как уже говорилось, при создании C#-скрипта Unity создает некий каркас, состоящий из подключенных библиотек и основного класса (используемого скриптом) с методами *Start()* и *Update()*.

В предыдущих случаях нами рассматривался метод *Update*, который вызывается каждый раз в новом кадре для каждого компонента всех объектов на сцене.

В данном случае мы воспользуемся методом *Start()*, который выполняется единожды для каждого компонента сразу после нажатия на кнопку «*Play*» и, соответственно, должен использоваться для инициализации переменных и придания им каких-либо начальных значений. Добавим в тело метода *Start()* функцию *Destroy()* и передадим в нее gameObject, указав таким образом, что скрипт должен уничтожить объект, компонентом которого он является:

```
// метод Start() выполняется единожды, сразу после окончания
загрузки сцены
Start() {
  // уничтожить объект, к которому прикреплен данный скрипт
   Destroy(gameObject);
}
```

Добавим этот скрипт к кубическому объекту, который должен удаляться, с помощью меню компонентов (рис. 26). Теперь после запуска сцены можно убедиться, что добавленный куб при запуске программы сразу пропадает.

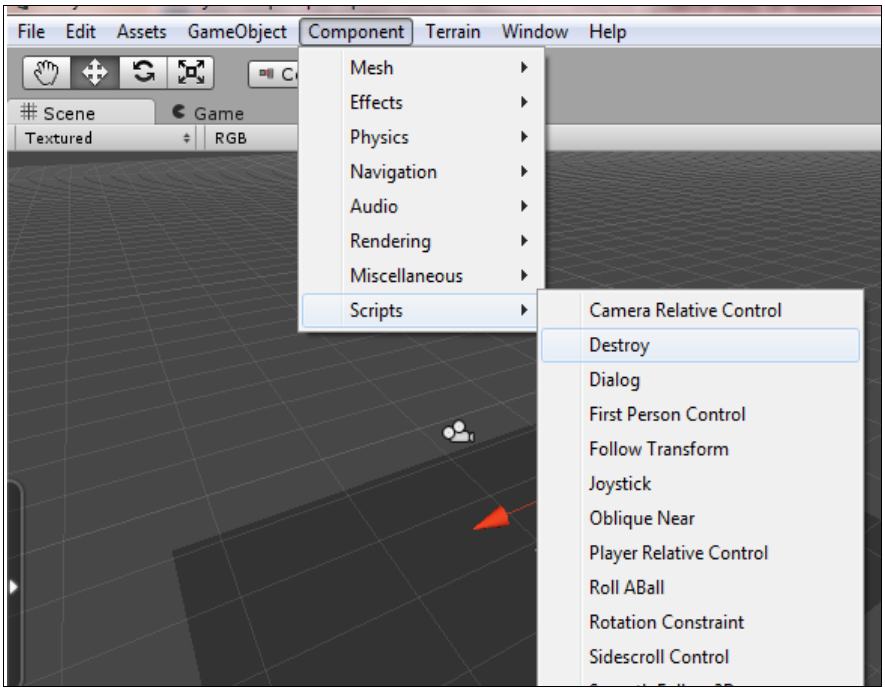


Рис. 26. Добавление C#-скрипта к объектам

Теперь попробуем уничтожить другой объект с помощью его поиска в среде. Для этого воспользуемся статическим методом *Find()* основного класса *GameObject*:

```
// ищем объект с именем Stena и если таковой есть – уничтожаем его
Destroy(GameObject.Find("Stena"));
```

В случае необходимости уничтожения объекта не сразу, а спустя какое-то время, можно задать значение во второй параметр функции Destroy:

```
// Теперь уничтожаем стену, спустя две секунды после загрузки сцены
Destroy(GameObject.Find("Stena"), 2);
```

Лабораторная работа № 3

Цели работы:

– знакомство с особенностями создания, назначением префабов в среде Unity3D, а также основам их взаимодействия;

– изучение основами взаимодействия трехмерных моделей с применением скриптов Unity3D;

– освоение приемов организации взаимодействия объектов за счет их удаления, разрушения (destroy).

Задание к лабораторной работе

1) Познакомиться с назначением префабов в среде Unity3d, особенностями их создания и взаимодействия, в частности, с целью их последующего удаления (разрушения).

2) Разработать в среде Unity3d сцену по материалу п. 4, сохранить и представить преподавателю.

3) Добавить в проект новый объект – сферу с физическими свойствами твердого тела из металлического материала. Наклонить поверхность и расположить объект (сферу) так, чтобы при падении с высоты происходил накат сферы на объекты среды с их последующем удалением.

4) Изменить материал сферы на упругий. Описать разницу.

5) Составить план-отчет по лабораторной работе.

Контрольное задание

Разработать интерактивное приложение «Арканоид».

Рекомендуемая литература

1. Goldstone, W. Unity Game Development Essentials. – Packt Publishing, 2009. –316 с.
2. Creighton, R.-H. Unity 3D Game Development by Example Beginner's Guide. – Packt Publishing, 2010. – 384 с.
3. Герасимов В. Unity 3.x Scripting. – Packt Publishing, 2011.
4. Sue Blackman. Beginning 3D Game Development with Unity: All-in-one, Multi-platform Game Development. – Apress, 2011. – 992 с.

Учебное электронное издание

ОСНОВЫ ГЕОМЕТРИЧЕСКОГО МОДЕЛИРОВАНИЯ В UNITY3D

Методические указания к выполнению лабораторных работ

Составители: СТЕПЧЕВА Зоя Валерьевна
ХОДОС Олег Сергеевич

Редактор Н. А. Евдокимова

Формат А4. Усл. печ. л. 1,86.
Объем данных 1,04 Мб. ЭИ № 77.

Ульяновский государственный технический университет
432027, г. Ульяновск, ул. Сев. Венец, 32.
Тел.: (8422) 778-113.
E-mail: venec@ulstu.ru
http://www.venec.ulstu.ru