**Министерство образования Республики Беларусь**

**БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**Факультет прикладной математики и информатики**

Кафедра технологий программирования

Гущенскова Александра Александровна

**ПРОЕКТИРОВАНИЕ МОБИЛЬНОГО ПРИЛОЖЕНИЯ ДЛЯ РЕШЕНИЯ ЗАДАЧ СТЕРЕОМЕТРИИ**

Курсовой проект  
студентки 3 курса 12 группы

| «Допустить к защите»  **Руководитель работы**  *Давидовская Мария Ивановна* Старший преподавательКафедра технологий программирования  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  «\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2018 г |  |
| --- | --- |

Минск 2018

РЕФЕРАТ

Курсовой проект, 32 c., 15 рис., 2 таблицы.

**Ключевые слова:** UNITY, ANDROID, 2D- И 3D- ГРАФИКА, МОБИЛЬНОЕ ПРИЛОЖЕНИЕ, ИГРА, СТЕРЕОМЕТРИЯ.

**Объект исследования –** игровой движок Unity3D и использование его возможностей для создания 3D-объектов в 2D-приложении.

**Цели работы –** рассмотреть методы для создания объектов 2D- и 3D-графики в приложениях для платформы Android, а также спроектировать приложение для решения задач стереометрии на Unity3D.

**Методы исследования –** анализ, моделирование.

**Результатами являются –** игра для решения задач стереометрии для платформы Android.

**Область применения –** для любого смартфона с операционной системой Android.

РЭФЕРАТ

Курсавы праект, 32 с., 15 рыс., 2 таблiцы

**Ключавыя словы**: UNITY, ANDROID, 2D- І 3D- ГРАФІКА, МАБIЛЬНАЕ ПРЫКЛАДАННЕ, ГУЛЬНЯ, СТЭРЭАМЕТРЫЯ.

**Аб'ект даследавання** **–** гульнявы рухавік Unity3D і выкарыстанне яго магчымасцяў для стварэння 3D-аб'ектаў у 2D-прыкладаннi.

**Мэты працы** **–** разгледзець метады для стварэння аб'ектаў 2D- і 3D-графікі ў прыкладаннях для платформы Android, а таксама спраектаваць прыкладанне для вырашэння задач стэрэаметрыі на Unity3D.

**Метады даследавання** **–** аналіз, мадэляванне.

**Вынікамі з'яўляюцца** **–** гульня для вырашэння задач стэрэаметрыі для платформы Android.

**Вобласць ужывання** **–** для любога смартфона з аперацыйнай сістэмай Android.

ESSAY

Course project, 32 p., 15 illustrations, 2 tables.

**Keywords**: UNITY, ANDROID, 2D AND 3D GRAPHICS, MOBILE APPLICATION, GAME, STEREOMETRY.

**Object of research –** the Unity3D game engine and the use of its capabilities to create 3D objects in a 2D application.

**Purpose –** to consider methods for creating 2D and 3D graphics objects in applications for the Android platform, and also design an application for solving stereometry problems on Unity3D.

**Methods of research** **–** analysis, modeling.

**The results are** **–** a game for solving problems of stereometry for the Android platform.

**Scope** **–** for any smartphone with the Android operating system.

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 7](#_gjdgxs)

[**Глава 1 ТЕХНОЛОГИИ РАЗРАБОТКИ МОБИЛЬНЫХ ПРИЛОЖЕНИЙ 2D- И 3D- ГРАФИКИ** 9](#_1fob9te)

[1.1 Проектирование приложений 2D- и 3D- графики для платформы Android 9](#_3znysh7)

[1.1.1 Пакет android.graphics 9](#_2et92p0)

[1.1.2 OpenGL ES 11](#_tyjcwt)

[1.1.3 Игровые движки. Unreal Engine 13](#_3dy6vkm)

[1.1.4 Игровые движки. Unity3D 13](#_1t3h5sf)

[1.2. Графические возможности Unity3D 14](#_4d34og8)

[1.2.1 Меши 14](#_2s8eyo1)

[1.2.2 Освещение пользовательского меша 15](#_17dp8vu)

[1.2.3 Текстурирование 17](#_3rdcrjn)

[1.2.4 Mesh Renderer 18](#_26in1rg)

[1.2.5 Взаимодействие с пользователем 18](#_lnxbz9)

[**Глава 2 РАЗРАБОТКА ТРЕБОВАНИЙ И ПРОЕКТИРОВАНИЕ ПРОТОТИПА ПРИЛОЖЕНИЯ** 19](#_35nkun2)

[2.1. Задачи проекта 19](#_1ksv4uv)

[2.2. Разработка требований 19](#_44sinio)

[2.2. Проектирование UML диаграмм 20](#_2jxsxqh)

[2.2.1 Диаграмма вариантов использования 20](#_z337ya)

[2.2.2 Объектная модель 22](#_1ci93xb)

[2.2.3 Диаграммы деятельности 23](#_2bn6wsx)

[2.3. Проектирование прототипа приложения 25](#_3as4poj)

[**Глава 3 РЕАЛИЗАЦИЯ МОБИЛЬНОГО ПРИЛОЖЕНИЯ ДЛЯ ПЛАТФОРМЫ ANDROID** 28](#_2p2csry)

[3.1 Главное меню 28](#_147n2zr)

[3.2 Экран со списком уровней 29](#_23ckvvd)

[3.2.1 Сохранение прогресса пользователя между запусками приложения 29](#_32hioqz)

[3.3 Игровой режим 30](#_1hmsyys)

[ЗАКЛЮЧЕНИЕ 32](#_2grqrue)

[СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ 33](#_vx1227)

[ПРИЛОЖЕНИЯ 34](#_3fwokq0)

[*Приложение А* 34](#_1v1yuxt)

[*Приложение Б* 35](#_4f1mdlm)

[*Приложение В* 36](#_2u6wntf)

**ВВЕДЕНИЕ**

В современном мире крайне важно иметь хорошее образование, чтобы добиться успеха. В процессе обучения люди ищут ресурсы, приложения, которые помогли бы им разобраться лучше и быстрее в какой бы то ни было теме. Также большой помощью является графическое представление тем, задач.

Образовательные приложения и ресурсы особенно популярны среди школьников и студентов. Зачастую у них возникают проблемы с естественными науками, в особенности с геометрией. Должно быть хорошее пространственное воображение, чтобы решать задачи без труда. Поэтому было бы неплохо предложить им такое приложение, которое бы помогало при решении задач в интерактивном плане.

Разрабатываемое приложение предлагает ряд типовых задач на построение сечений в многогранниках. Обычно эту тему проходят в старших классах. В приложении можно будет взаимодействовать с фигурой любым образом, что, в сравнении с решением задачи на листочке, намного облегчит процесс решения и сделает его более интересным.

Помимо этого, не стоит забывать про то, что у приложения могут быть пользователи не только школьники и студенты, но и взрослые люди. Многие любят математику и любят решать логические задачи. А в настоящее время сложно найти хорошие предложения.

Приложение разрабатывается для платформы Android, чтобы оно было под рукой в любое время.

Объектом исследования являются 2D- и 3D-объекты, с которыми пользователь может взаимодействовать, и их создание в приложениях для платформы Android, а также игровой движок Unity3D и его возможности в создании этих объектов.

Методы исследования, которые применялись в работе, включают в себя анализ и моделирование.

В главе 1 приведены технологии создания графических объектов в приложениях для платформы Android, а также подробно рассмотрено создание 3D-объектов в Unity3D.

Во главе 2 спроектировано игровое приложение для решения задач в стереометрии. Разработана спецификация и требования к функционалу приложений. В частности, представлены диаграммы вариантов использования, деятельности и другое.

В главе 3 показана реализация мобильного приложения.

# **ТЕХНОЛОГИИ РАЗРАБОТКИ МОБИЛЬНЫХ ПРИЛОЖЕНИЙ 2D- И 3D- ГРАФИКИ**

* 1. **1.1 Проектирование приложений 2D- и 3D- графики для платформы Android**

Существует множество способов проектирования мобильных графических приложения для платформы Android. Выбор конкретной технологии зависит от решаемой задачи, выделенного времени и бюджета.

Для анализа взяты самые популярные подходы к рисованию графических объектов в приложениях для платформы Android. Отнесем к ним пакет android.graphics, фреймворк OpenGL, а также возможности, которые предоставляют игровые движки.

Таблица 1.1 Сравнение технологий для создания графических объектов

|  | **Android.graphics** | **OpenGL ES** | **Unity3D** |
| --- | --- | --- | --- |
| **2D** | + | + | + |
| **3D** | - | + | + |
| **Взаимодействие с пользователем** | Только через View, на котором он расположен | Присутствует, но не тривиально | + |
| **Кроссплатформенность** | - | + | + |

* + 1. **1.1.1 Пакет android.graphics**

Пакет android.graphics предлагает графические инструменты низкого уровня, такие как точки, прямоугольники, холсты, цветовые фильтры. Они позволяют обрабатывать рисунок непосредственно на экране. В пакете определены возможности для рисования только 2D-графики. Библиотека android.graphics.Drawable как раз и предоставляет эти возможности.

Класс Drawable является базовым классом для того, что может быть нарисовано. Различные подклассы помогают с конкретными сценариями. Имеется возможность их расширения, чтобы определить свои собственные объекты с возможностью рисования, которые будут вести себя уникальным образом. Объекты Drawable не имеют возможности получать события или иным образом взаимодействовать с пользователем. Эту функциональность выполняют объекты View, которые и содержат кастомный Drawable-объект.

Объекты Drawable могут принимать различные формы:

1. Растровое изображение: простейший объект Drawable формата PNG или JPEG.
2. Вектор: определяется в XML-файле, как набор точек, линий и кривых вместе с соответствующей информацией о цвете. Этот тип объекта можно масштабировать без потери качества отображения.
3. Фигура: содержит простые команды рисования вместо необработанного растрового изображения, позволяя в некоторых случаях изменять размер.
4. Слои: составной объект, который рисует несколько drawable-объектов слоями и другие.
5. Состояния: составной объект, который выбирает один из объектов из множества drawable-объектов согласно состоянию.
6. Уровни: составной объект, который выбирает один из объектов из множества drawable-объектов согласно уровню.
7. Масштаб: составной drawable с одним дочерним drawable, общий размер которого изменяется на основе текущего уровня.

Существует три способа определения и создания экземпляра Drawable: использование изображения из ресурсов проекта; используя XML-файл, который определяет свойства Drawable; или используя конструкторы абстрактного класса.

* + 1. **1.1.2 OpenGL ES**

Android включает поддержку Open Graphics Library(OpenGL) для 2D- и 3D-графики, в частности, API OpenGL ES. OpenGL является одним из самых популярных прикладных программных интерфейсов для разработки приложений двумерной и трехмерной графики. Графическая система OpenGL поддерживается большинством производителей аппаратных и программных платформ. Библиотеку можно использовать в Windows, MacOS. Она позволяет стандартным образом работать с графикой через аппаратные 3D ускорители. Независимо от характеристик устройства и размера экрана, используя OpenGL API, получаются одинаковые результаты. Характерными особенностями OpenGL, являются:

1. Стабильность. Дополнения и изменения в стандарте реализуются таким образом, чтобы сохранить совместимость с разработанным ранее программным обеспечением.
2. Легкость применения. OpenGL имеет интуитивно понятный интерфейс и продуманную структуру. Это позволяет с меньшими затратами создавать приложения, которые содержат меньше строк кода, чем с использованием других графических библиотек.
3. Надежность и переносимость. Приложения, которые используют OpenGL, будут гарантировать одинаковый визуальный результат вне зависимости от организации отображения информации и типа используемой операционной системы. Кроме того, эти приложения могут выполняться на персональных компьютерах, рабочих станциях и суперкомпьютерах.

OpenGL ES - это образец спецификации OpenGL, предназначенной для встроенных устройств. OpenGL ES - низкоуровневый API. Другими словами, он не предлагает никаких методов, которые позволяют быстро создавать или манипулировать 3D-объектами. Вместо этого во время работы с ним вы должны вручную управлять задачами, такими как создание отдельных вершин и граней трехмерных объектов, вычисление различных 3D-преобразований и создание различных типов шейдеров. Android поддерживает OpenGL как через API инфраструктуры, так и через Native Development Kit (NDK).

В платформе Android есть два фундаментальных класса, которые позволяют создавать и управлять графикой с помощью API OpenGL ES: GLSurfaceView и GLSurfaceView.Renderer.

GLSurfaceView - представление, в котором рисуют и манипулируют объектами, используя вызовы API OpenGL, и аналогично функции SurfaceView. Можно использовать этот класс, создав экземпляр GLSurfaceView и добавив к нему пользовательский Renderer – интерпретатор каркаса объекта. Однако, если нужно захватывать события сенсорного экрана, нужно расширить класс GLSurfaceView для внедрения подписчиков событий.

GLSurfaceView.Renderer – интерфейс, который определяет методы, необходимые для рисования графики в GLSurfaceView. Реализация этого интерфейса предоставляется как отдельный класс и прикрепляется к экземпляру GLSurfaceView с помощью GLSurfaceView.setRenderer().

Так как в работе рассматривается создание 3D-объектов и взаимодействие пользователя с ним, то рассмотрим, как можно задать вращение объекта при прикосновении.

Чтобы приложение OpenGL ES реагировало на события касания, вы должны реализовать метод onTouchEvent() в своем классе GLSurfaceView. Пример реализации (Приложение А) показывает, как прослушивать события MotionEvent.ACTION\_MOVE и переводить их на угол поворота для фигуры. Событие происходит только в случае, если во время касания позиция касания изменилась. Код представлен на языке Java.

* + 1. **1.1.3 Игровые движки. Unreal Engine**

Два наиболее популярных игровых движка – это Unreal Engine и Unity3D.

Unreal Engine - это игровой движок, разработанный Epic Games, впервые представленный в 1998 году. Unreal Engine использует С++, на нём можно разрабатывать игры для iOS и Android. В движке есть мощный редактор, заключающий в себе несколько узкоспециальных редакторов. Некоторые редакторы даже могут заменить определённые программы. Unreal Engine предлагает один из наилучших механизмов рендеринга на рынке.

Unreal Engine 4 имеет решение, позволяющее не писать много кода на C++. Это Blueprint — редактор визуального скриптинга. Технически не нужно писать ни одной строки кода. Это очень удобно для создания быстрых прототипов.

Текущий релиз – Unreal Engine 4 – позволяет разрабатывать для Windows, MacOS, Linux, HTML5, iOS, Android, PlayStation 4, Xbox One и виртуальной реальности.

* + 1. **1.1.4 Игровые движки. Unity3D**

Unity3d ˗ это мощный мультиплатформенный инструмент для разработки десктопных, мобильных и веб приложений 2D- и 3D-графики, 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 C# или JavaScript.

Unity3d позволяет разрабатывать для платформ: Windows, MacOs, Linux, iOS, Android, PlayStation 4, Xbox One. Есть возможность создания VR и AR приложений.

Для создания мобильных игр Unity является идеальным решением. Это подтверждается доминированием Unity среди разработчиков мобильных игр, а также большим количеством плагинов для использования нативных возможностей мобильных платформ. Среди них реклама, аналитика, внутренние покупки, игровые центры и т. д. — все это интегрируется в игру за считанные минуты. Unity также является прекрасным выбором для разработки 2D игр, поскольку обладает большим количеством возможностей создания данного вида игр.

Так как проектируемое приложение является 2D игрой, содержащей элементы 3D графики, то удачным выбором будет именно разработка на Unity3D.

* 1. **1.2. Графические возможности Unity3D**

В Unity3D определены 3D-объекты примитивы. К ним относятся куб, сфера, цилиндр, капсула. Но также существует возможность определить свои трехмерные объекты. Это осуществляется с помощью написания меша.

* + 1. **1.2.1 Меши**

3D меши – главный графический примитив в Unity. Они используются для отображения визуальной геометрии в играх. Меш состоит из треугольников, расположенных в 3D-пространстве так, чтобы создать впечатление замкнутого объекта. Три угловые точки и вершины определяют треугольник.

В Unity меш имеет:

* vertices — массив координат вершин меша;
* normals — массив данных содержащих информацию о том, куда каждая вершина смотрит (каждая нормаль перпендикулярна плоскости из вершин);
* triangles — массив данных о том, как вершины соединены между собой. Берутся группами по три от начала этого массива таким образом, чтобы элементы 0, 1 и 2 определили первый треугольник, 3, 4 и 5 определили второй, и так далее;
* colors — массив цветов вершин;
* uv — текстурные координаты;
* tangents — касательные к каждой вершине.

То есть меш — это класс с набором атрибутов, с помощью которых он рендерится, отражает свет, отображает текстуру и т.д.

* + 1. **1.2.2 Освещение пользовательского меша**

Треугольников достаточно, чтобы определить основную форму объекта, но в большинстве случаев необходима дополнительная информация для отображения сетки. Для правильного затенения при освещении, для каждой вершины указывается вектор нормали. Он направлен наружу перпендикулярно поверхности сетки в положении вершины с которой он связан.

Когда рассчитывают затенения, каждая нормаль к вершине сравнивается с направлением падающего света. Он также является вектором. Если направления этих векторов параллельны, то поверхность получает свет в лоб в этой точке и для затенения будет использоваться полная яркость света. Если свет приходит точно перпендикулярно вектору нормали, то он не даст освещения поверхности в этой точке. Как правило, свет падает под углом к нормали и поэтому в зависимости от угла затенение будет где-то между полной яркостью и полной темнотой.

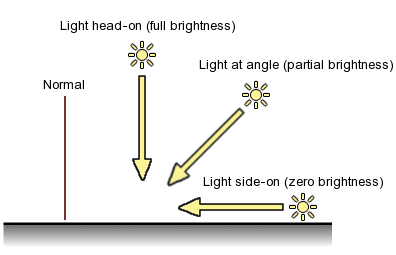


Рисунок 1.1 – Уровень яркости в зависимости от расположения света к вектору нормали

За счет того, что сетка состоит из треугольников, может показаться, что нормали на его углах будут точно перпендикулярны плоскости их треугольника. Однако на самом деле нормали интерполируются между треугольниками с получением среднего значения между соседними углами. Если все три нормали треугольника указывают в одном направлении, то треугольник будет равномерно освещен на всем протяжении. Показателем того, что разные треугольники равномерно затенены будет то, что края будут очень четкими и отчетливыми. Это именно то, что требуется для модели куба или других объектов с острым краем, но интерполяция нормалей может быть использована для создания плавного затенения, чтобы представить изогнутую поверхность.

Чтобы получить резкие ребра, необходимо удвоить вершины в ребре, т.к. оба прилегающих треугольника должны иметь отдельные нормали. Для изогнутых поверхностей вершины смежных треугольников обычно общие, однако нужна некоторая интуиция для определения лучшего направления общей нормали. Нормаль может быть просто средней от нормалей плоскостей окружающих треугольников. Однако для такого объекта как сфера нормаль просто направлена наружу от центра сферы.

Вызывая Mesh.RecalculateNormals, можно поручить Unity рассчитать нормали, сделав некоторые предположения о “смысле” геометрии сетки; она предполагает, что вершины общие для нескольких треугольников обозначают гладкую поверхность в то время как удвоенные вершины указывают на четкие края. В большинстве случаев это не плохое приближение, однако RecalculateNormals будет спотыкаться в некоторых ситуациях текстурирования, когда вершины должны быть удвоены, хотя поверхность гладкая.

* + 1. **1.2.3 Текстурирование**

Помимо освещения модель также обычно может использовать текстурирование для создания тонких деталей поверхности. Текстура подобна изображению, напечатанному на растягиваемой пленке или резине. Для каждого треугольника сетки, определяется треугольная площадь изображения текстуры и этот треугольник текстуры растягивается и сжимается, чтобы соответствовать треугольнику сетки. Чтобы сделать эту работу, в каждой вершине нужно сохранять координаты точки текстуры, которая ей соответствует. Эти координаты двумерны и масштабированы в диапазон от 0 до 1 (0 означает нижний/левый угол изображения, а 1 верхний/правый). Чтобы избежать путаницы этих координат с декартовыми координатами в 3D-мире, они называются U и V.

Подобно нормалям, текстурные координаты уникальны для каждой вершины и, таким образом, существуют ситуации, когда приходится дублировать вершины для получения различных UV значение вдоль ребра. Очевидный пример, это когда два соседних треугольника используют разделенные части текстуры (например, глаза на текстуре лица). Также, большинство полностью замкнутых объемов потребуют “шов”, где область текстуры заворачивается, и соединяется. Значения UV на одной стороне шва будут отличаться от тех, что на другой стороне.

* + 1. **1.2.4 Mesh Renderer**

Чтобы увидеть в своей сцене меш, необходимо добавить к игровому объекту компонент Mesh Renderer. По умолчанию он должен назначаться автоматически. При отсутствии компонента Mesh Renderer, меш по-прежнему будет виден в сцене (и считаться загруженным в память компьютера) просто он не будет прорисовываться.

Компонент Mesh Filter берёт указанный ассет и передаёт его компоненту Mesh Renderer для последующего вывода этого меша на экран монитора.

Mesh Renderer в свою очередь располагает меш на позициях, определенных в компоненте Transform.

* + 1. **1.2.5 Взаимодействие с пользователем**

Чтобы приложение реагировало на касания, которые относятся к объектам, нужно просто реализовать функцию, относящуюся к типу касания. Так, чтобы вращать объект при касании пользователя необходимо просто в скрипте нужного объекта реализовать функцию OnMouseDrag(). Пример реализации в Приложении Б. Код приведен на языке C#.

ВЫВОДЫ

1. Проведена сравнительная характеристика основных технологий для создания приложений 2D- и 3D-графики на платформе Android. Среди технологий для создания 3D-объектов была выбрана разработка на игровом движке Unity3D, за счет простоты использования, легкого внедрения 3D-объектов в 2D-приложение и своей специализации на 2D- и 3D-графике.
2. Выявлено, что в Unity уже предопределены некоторые трехмерные объекты. Приведено описание процесса создания своих трехмерных объектов, а также взаимодействие пользователя с этими объектами.

# **РАЗРАБОТКА ТРЕБОВАНИЙ И ПРОЕКТИРОВАНИЕ ПРОТОТИПА ПРИЛОЖЕНИЯ**

* 1. **2.1. Задачи проекта**

Задача проекта состоит в разработке мобильного приложения для решения типичных задач на построение в стереометрии. С одной стороны, оно может служить как обучающее приложение для старшеклассников. С другой стороны, оно может быть интересно и взрослым людям, поскольку является логической игрой и глубоких знаний теорем для решения задач не требуется.

Приложение построено на уровнях. Каждый уровень – новая задача. Цель – построить сечение в некотором многограннике по условию.

* 1. **2.2. Разработка требований**

Требования делятся на функциональные и нефункциональные.

Функциональные требования определяют поведение системы.

* Оперативное начало игры;
* Сохранение пройденных уровней;
* Повторное прохождение уровня;
* Просмотр справочного материала по теме уровня;
* Просмотр обучающего материала;
* Возможность вращения фигуры во время прохождения уровня;
* Отмена действия;
* Сброс прогресса;

Нефункциональные требования описывают характер поведения системы.

* Простота в использовании для любого пользователя;
* Интуитивно понятный интерфейс.
  1. **2.2. Проектирование UML диаграмм**

UML (Unified Modeling Language) – это система обозначений, которую применяют для объектно-ориентированного анализа и проектирования.

* + 1. **2.2.1 Диаграмма вариантов использования**

Диаграммы вариантов использования обычно называются диаграммами поведения или прецедентов, которые описывают набор действий (вариантов использования), которые системы должны или могут выполнять в сотрудничестве с одним или несколькими внешними пользователями системы (участниками). Каждый вариант использования должен обеспечивать некоторый конечный результат для участников или других заинтересованных сторон системы. Описывается поведение системы с точки зрения пользователя. Соответственно, действующим лицом в диаграмме является пользователь.

В ходе подготовки спецификации, используемой для разработки приложения, составлено несколько основных диаграмм вариантов использования:

1. Игрок в первый раз заходит в игру (Рисунок 2.1)

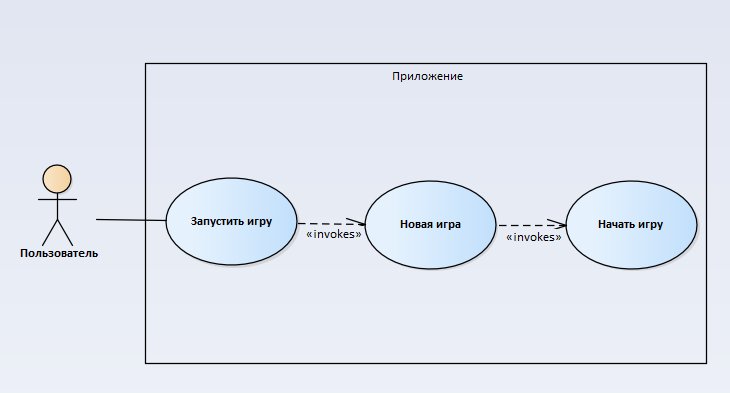


Рисунок 2.1 – Диаграмма вариантов использования «Новая игра»

1. Игрок заходит во второй раз и более (Рисунок 2.2)

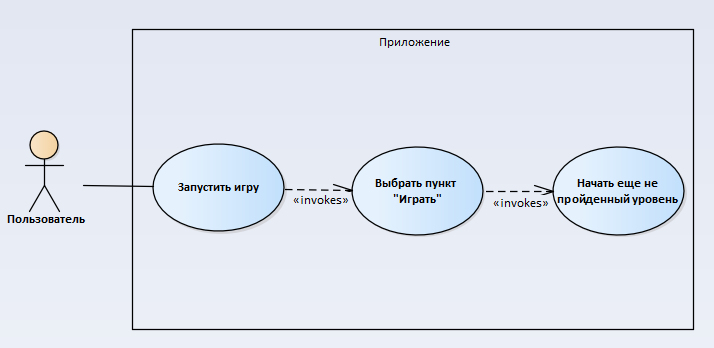


Рисунок 2.2 – Диаграмма вариантов использования «Продолжить игру»

1. Игрок переигрывает уровень N (Рисунок 2.3)

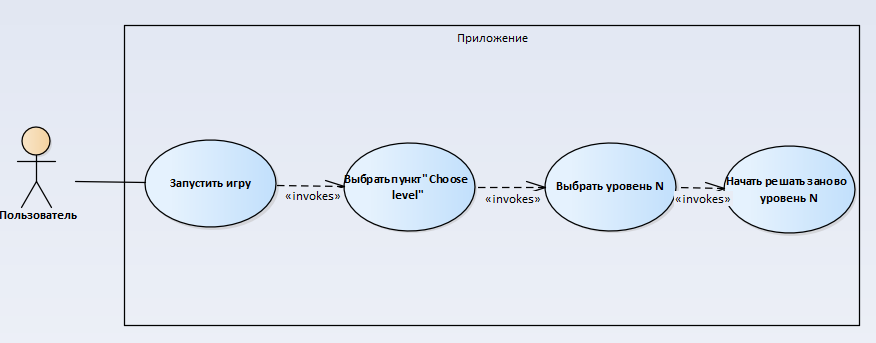


Рисунок 2.3 – Диаграмма вариантов использования «Выбрать уровень N»

1. Игрок сбрасывает свой прогресс (Рисунок 2.4)

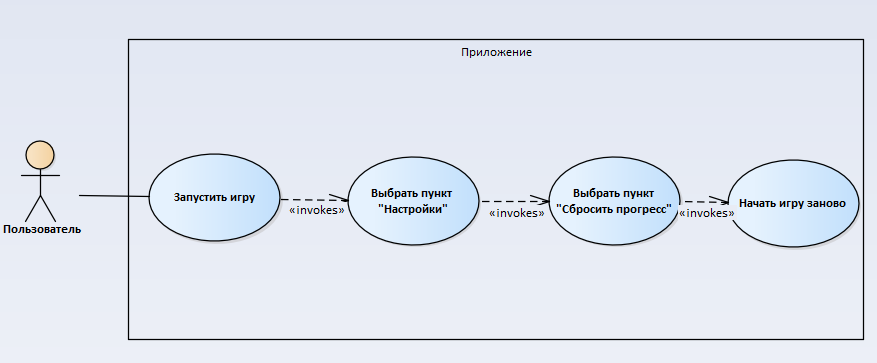


Рисунок 2.4 – Диаграмма вариантов использования «Сбросить прогресс»

Выше представлены все основные варианты использования приложения.

* + 1. **2.2.2 Объектная модель**

Объектная модель представляет объекты, их свойства и поведение.

Так как Unity, как и многие другие игровые движки, наиболее приспособлен к КОП (компонентно-ориентированное программирование), то проектирование будет построено на определении объектов-контейнеров с набором компонентов. Именно компонент является базовой единицей для реализации бизнес-логики в Unity. Каждый компонент решают одну определенную задачу. В целом, КОП можно рассматривать как развитие принципов ООП с устранением проблемного места, известного как хрупкий базовый класс. В Unity компонентами являются классы, унаследованные от MonoBehaviour.

В приложении несколько объектов с разными свойствами и поведением:

Таблица 2.1 – Описание всех игровых объектов

| Объект | Свойства | Поведение |
| --- | --- | --- |
| Фигура | Массив объектов координат ребер, массив буквенных обозначений вершин и массив граней. | Фигура может вращаться при определенных нажатиях на экран пользователем. |
| Уровень | Номер уровня, свойства, определяющие фигуру, координаты решения, описание. | Определяет данные каждого уровня. |
| Кнопка рисования |  | Рисует объект согласно тому, что нажал пользователь. |

* + 1. **2.2.3 Диаграммы деятельности**

Диаграмма деятельности является поведенческой диаграммой, описывающей динамические аспекты системы. Она показывает некоторые последовательности (алгоритмы) действий, которые могут происходить в системе. Диаграмма позволит определить, как поведет себя пользователь, поставив перед собой некоторую цель.

Ниже приведены диаграммы деятельности для некоторых ситуаций:

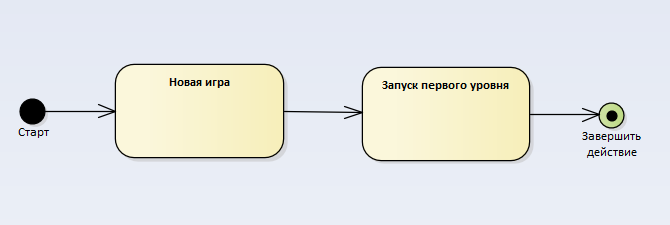


Рисунок 2.5 – Диаграмма деятельности «Начать новую игру»

Предложено минимальное количество действий для того, чтобы пользователь смог начать новую игру (Рисунок 2.5). Это сделано из следующих соображений: чем быстрее пользователь начнет играть, тем быстрее он сможет влиться в процесс. Следующие диаграммы строились по этому же принципу.

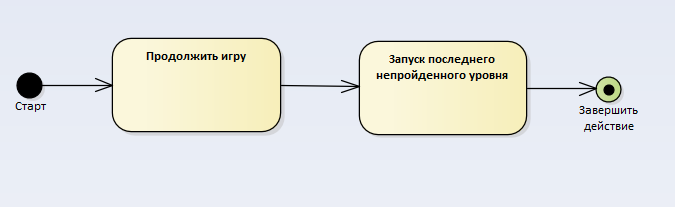


Рисунок 2.6 – Диаграмма деятельности «Продолжить игру»

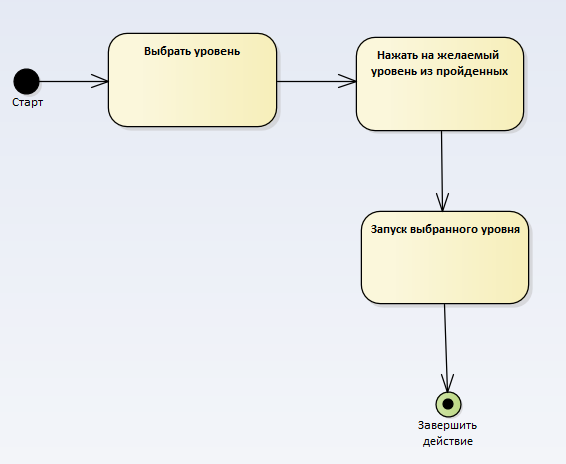


Рисунок 2.7 – Диаграмма деятельности «Выбрать уровень N»

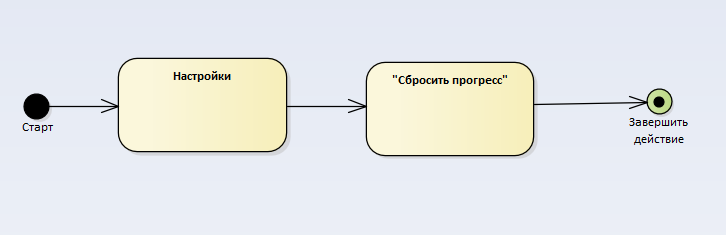


Рисунок 2.8 – Диаграмма деятельности «Сбросить прогресс»

* 1. **2.3. Проектирование прототипа приложения**

Приложение будет иметь три экрана: первый – главное меню. Второй – уровни. Третий экран – непосредственно сам уровень.

На первом экране (Рисунок 2.9) будут элементы для перехода сразу в режим игры и на экран уровней, для перехода в настройки и справку. Настройки будут предоставлять пользователю возможность сбросить прогресс. Справка будет предоставлять информацию об управлении в режиме игры.

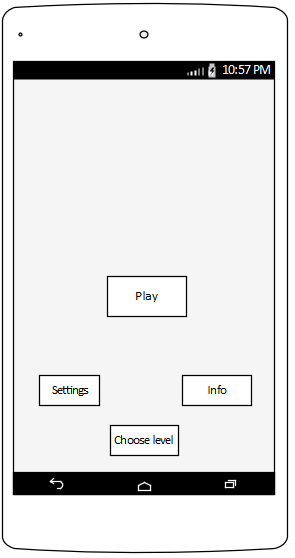


Рисунок 2.9 – Прототип главного экрана

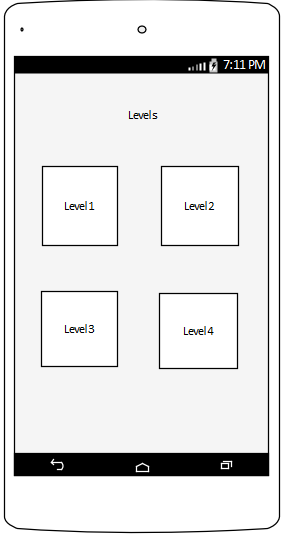


Рисунок 2.10 – Прототип экрана со списком уровней

На экране режима игры (Рисунок 2.11) есть следующие элементы: кнопка возврата к списку уровней, кнопка, которая открывает условие задачи и краткую теоретическую справку по уровню, кнопки для отмены действия и всех действий, фигура, на которой выполняется сечение и непосредственно кнопки возможных действий по построению.

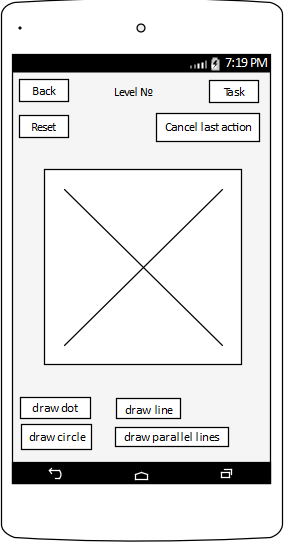


Рисунок 2.11 – Прототип режима игры

ВЫВОДЫ

1. Разработаны функциональные и нефункциональные требования к приложению.
2. Разработаны диаграммы, показывающие поведение пользователя в системе, а также объектная модель.
3. Спроектированы прототипы приложения.

# **РЕАЛИЗАЦИЯ МОБИЛЬНОГО ПРИЛОЖЕНИЯ ДЛЯ ПЛАТФОРМЫ ANDROID**

* 1. **3.1 Главное меню**

Первое, что увидит игрок, запустив приложение, – это главное меню (Рисунок 3.1). В нижней части экрана располагаются кнопки с интуитивно понятным интерфейсом. Нажав на кнопку со значком «Играть», пользователь сможет моментально приступить к игре, будь то заход в приложение в первый раз или нет. Чтобы перепройти один из пройденных уже уровней нужно нажать на кнопку «Choose level».

Кнопка слева – «Настройки». Она переносит пользователя на экран, на котором будет несколько пунктов, самым важным из которых является «Сбросить игру».

Кнопка справа – «Справка». Нажав на нее, пользователь получит информацию по управлению в режиме игры.

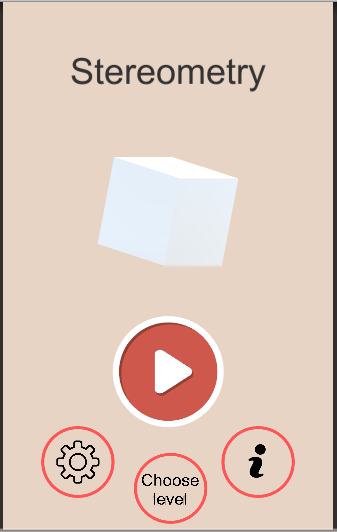


Рисунок 3.1 – Скриншот главного меню

* 1. **3.2 Экран со списком уровней**

На этом экране располагается список уровней (Рисунок 3.2). Цветная цифра у тех уровней, которые уже доступны игроку. А черная у тех, которые пользователь еще не прошел, следовательно, они не кликабельны. Также присутсвует кнопка, с помощью которой пользователь может вернуться на главный экран.

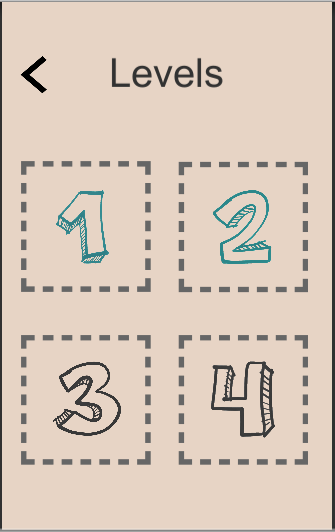


Рисунок 3.2 – Скриншот экрана со списком уровней

* + 1. **3.2.1 Сохранение прогресса пользователя между запусками приложения**

В Unity определен класс PlayerPrefs. Он сохраняет и получает необходимую информацию для игры между игровыми сеансами.

На Android данные сохраняются на устройстве. Данные сохраняются в параметрах SharerPreferences. C # / JavaScript, Android Java и нативный код могут получить доступ к данным PlayerPrefs. Данные PlayerPrefs физически хранятся в файле /data/data/pkg-name/shared\_prefs/pkg-name.xml.

Данные сохраняются по ключу, который указывается разработчиком. Можно сохранять данные с типом int, float и string. Для разрабатываемого приложения хватит и этих типов.

* 1. **3.3 Игровой режим**

На этом экране происходит игра (Рисунок 3.3). У пользователя есть условие, которое показывается при открытии этого экрана, а также его можно посмотреть еще раз, нажав на вернхнюю кнопку справа. При нажатии на эту же кнопку можно получить краткую информацию по теме уровня: определения и важные теоремы.

По центру экрана располагается многогранник, на котором выполняются построения. При нажатии на экран пользователь может его вращать. Считается, что уровень пройден, если на многограннике построено правильное сечение, которое требуется в условии.

Построение выполняется с помощью кнопок, расположенных внизу экрана. Каждая из кнопок представляет, какой вид построения нужно выполнить (точка, окружность, прямая, параллельная прямая, перпендикулярная прямая).

Отменить действие или начать уровень заново можно также с помощью кнопок с соответствующими значками.

Кнопка выхода переводит пользователя на экран со списком уровней.

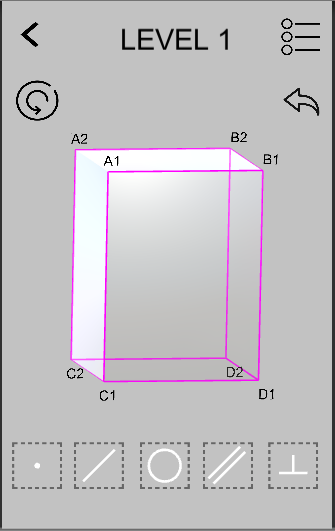


Рисунок 3.3 – Скриншот режима игры

Реализован для примера первый уровень, в котором происходит работа с предопределенным объектом Unity3D – кубом (Приложение В). Чтобы сделать куб прозрачным был добавлен к нему материал, у которого прозрачность alpha установлена в 0. А затем прорисованы ребра, являющиеся объектами LineRenderer и буквы – TextMesh. Также к кубу добавлена возможность вращения при прикосновениях пользователя.

ВЫВОДЫ

1. Разработаны сцены приложения.
2. Определено, как сохранять данные между игровыми сессиями.
3. Рассмотрена работа с предопределенным трехмерным объектом – кубом, с LineRenderer и TextMesh.

**ЗАКЛЮЧЕНИЕ**

Приложение “Stereometry” создано для решения задач на построение в стереометрии на телефонах и планшетах с операционной системой Android.

В ходе выполнения работы были рассмотрены и решены следующие задачи:

1. Изучены технологии для создания 2D- и 3D-объектов в приложениях для платформы Android.
2. Детально изучены возможности Unity3D в создании и работе с 3D-объектами.
3. Разработаны требования для приложения решения задач на построение в стереометрии
4. Разработаны следующие диаграммы UML: вариантов использования и деятельности
5. Разработана объектная модель приложения
6. Разработаны прототипы приложения
7. Реализовано приложение. При этом создан удобный и понятный функционал для пользователя.

Для проектирования приложения использовалась среда Enterprise Architect Version 13.

Для реализации приложения использовалась среда Unity 2018.2.10f1.

**СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ**

* 1. П. Дейтел, Х. Дейтел, Э. Дейтел, М. Моргано. Android для программистов: создаём приложения. — СПб.: Питер, 2.14. — 560 с.: ил.
  2. Голощапов А.Л. Google Android. Программирование для мобильных устройств (2011)
  3. Android Developers [Electronic resource] / Documentation for Android Operation System – Mode of access: [*http://developer.android.com/index.html*](http://developer.android.com/index.html)
  4. Developers [Electronic resource] – Mode of access: *https://stuff.mit.edu/afs/sipb/project/android/docs/index.html*
  5. Хокинг Д. Unity в действии. Мультиплатформенная разработка на C#. — СПб.: Питер, 2016. — 336 с: ил.
  6. Мэннинг Д., Батфилд-Эддисон П. Unity для разработчика. Мобильные мультиплатформенные игры. — СПб.: Питер, 2018. — 352 с: ил.
  7. Unity Technologies [Electronic resource] / Documentation for Unity – Mode of access:[*https://docs.unity3d.com/Manual/index.html*](https://docs.unity3d.com/Manual/index.html)

**ПРИЛОЖЕНИЯ**

* 1. *Приложение А*

private final float TOUCH\_SCALE\_FACTOR = 180.0f / 320;  
private float mPreviousX;  
private float mPreviousY;  
  
@Override  
public boolean onTouchEvent(MotionEvent e) {  
    float x = e.getX();  
    float y = e.getY();  
  
    switch (e.getAction()) {  
        case MotionEvent.ACTION\_MOVE:  
  
            float dx = x - mPreviousX;  
            float dy = y - mPreviousY;  
  
            if (y > getHeight() / 2) {  
              dx = dx \* -1 ;  
            }  
  
            if (x < getWidth() / 2) {  
              dy = dy \* -1 ;  
            }

            mRenderer.setAngle(  
                    mRenderer.getAngle() +  
                    ((dx + dy) \* TOUCH\_SCALE\_FACTOR));  
            requestRender();  
    }  
  
    mPreviousX = x;  
    mPreviousY = y;  
    return true;  
}

* 1. *Приложение Б*

using UnityEngine;

public class RotatedObjectBehaviour : MonoBehaviour

{

private float rotSpeed = 10;

private void OnMouseDrag()

{

float rotX = Input.GetAxis("Mouse X") \* rotSpeed \* Mathf.Deg2Rad;

float rotY = Input.GetAxis("Mouse Y") \* rotSpeed \* Mathf.Deg2Rad;

transform.RotateAround(Vector3.up, -rotX);

transform.RotateAround(Vector3.right, rotY);

}

}

* 1. *Приложение В*

using UnityEngine;

using System.Collections.Generic;

[RequireComponent(typeof(LineRenderer))]

public class TransparentBehaviour : MonoBehaviour {

private float rotSpeed = 10;

private Vector3[][] edges;

void Start ()

{

InitEdges();

CreateShape();

CreateLetters();

transform.Rotate(-8, -12, 0.5f);

}

private void OnMouseDrag()

{

float rotX = Input.GetAxis("Mouse X") \* rotSpeed \* Mathf.Deg2Rad;

float rotY = Input.GetAxis("Mouse Y") \* rotSpeed \* Mathf.Deg2Rad;

transform.RotateAround(Vector3.up, -rotX);

transform.RotateAround(Vector3.right, rotY);

}

private void CreateShape()

{

foreach (var list in edges)

{

CreateLine(list[0], list[1]);

}

}

private void CreateLetters()

{

CreateText(-0.53f, 0.59f, -0.5f, "A1");

CreateText(0.5f, 0.59f, -0.5f, "B1");

CreateText(-0.53f, -0.53f, -0.5f, "C1");

CreateText(0.5f, -0.53f, -0.5f, "D1");

CreateText(-0.53f, 0.59f, 0.5f, "A2");

CreateText(0.5f, 0.59f, 0.5f, "B2");

CreateText(-0.53f, -0.53f, 0.5f, "C2");

CreateText(0.5f, -0.53f, 0.5f, "D2");

}

private void CreateLine(Vector3 first, Vector3 second)

{

GameObject line = new GameObject();

line.transform.SetParent(transform);

line.AddComponent<LineRenderer>();

LineRenderer lineRenderer = line.GetComponent<LineRenderer>();

lineRenderer.positionCount = 2;

lineRenderer.SetWidth(0.03f, 0.03f);

lineRenderer.SetPosition(0, first);

lineRenderer.SetPosition(1, second);

lineRenderer.useWorldSpace = false;

}

private void CreateText(float x1, float y1, float z1, string letter)

{

GameObject text = new GameObject();

text.transform.SetParent(transform);

TextMesh t = text.AddComponent<TextMesh>();

t.color = Color.black;

t.text = letter;

t.fontSize = 300;

t.characterSize = 0.009f;

t.transform.localPosition += new Vector3(x1, y1, z1);

}

private void InitEdges()

{

edges = new Vector3[][]

{

new []

{

new Vector3(-1.5f, 2f, 1.5f), new Vector3(-1.5f, 2f, -1.5f)

},

new []

{

new Vector3(-1.5f, -2f, 1.5f), new Vector3(-1.5f, -2f, -1.5f)

},

new []

{

new Vector3(1.5f, 2f, 1.5f), new Vector3(1.5f, 2f, -1.5f)

},

new []

{

new Vector3(1.5f, -2f, 1.5f), new Vector3(1.5f, -2f, -1.5f)

},

new []

{

new Vector3(-1.5f, 2f, 1.5f), new Vector3(1.5f, 2f, 1.5f)

},

new []

{

new Vector3(-1.5f, -2f, 1.5f), new Vector3(1.5f, -2f, 1.5f)

},

new []

{

new Vector3(1.5f, 2f, 1.5f), new Vector3(-1.5f, 2f, 1.5f)

},

new []

{

new Vector3(1.5f, -2f, 1.5f), new Vector3(-1.5f, -2f, 1.5f)

},

new []

{

new Vector3(-1.5f, 2f, 1.5f), new Vector3(-1.5f, -2f, 1.5f)

},

new []

{

new Vector3(-1.5f, -2f, 1.5f), new Vector3(-1.5f, 2f, 1.5f)

},

new []

{

new Vector3(1.5f, 2f, 1.5f), new Vector3(1.5f, -2f, 1.5f)

},

new []

{

new Vector3(1.5f, -2f, 1.5f), new Vector3(1.5f, 2f, 1.5f)

},

new []

{

new Vector3(-1.5f, 2f, -1.5f), new Vector3(1.5f, 2f, -1.5f)

},

new []

{

new Vector3(-1.5f, -2f, -1.5f), new Vector3(1.5f, -2f, -1.5f)

},

new []

{

new Vector3(1.5f, 2f, -1.5f), new Vector3(1.5f, -2f, -1.5f)

},

new []

{

new Vector3(-1.5f, -2f, -1.5f), new Vector3(-1.5f, 2f, -1.5f)

}

};

}

}