**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ**

**Кафедра технологий программирования**

**Проектирование приложения на микросервисной архитектуре с использованием Microsoft Azure**

Курсовая

|  |
| --- |
| Гущенсковой Александры Александровны |
| студентки 3 курса,  специальность «прикладная информатика» |
| Научный руководитель: |
| старший преподаватель |
| Давидовская М.И. |

|  |  |
| --- | --- |
|  |  |

Минск, 2019

РЕФЕРАТ

Курсовая работа, 38 c., 18 рис., 2 таблицы.

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

**Объект исследования** — объектом исследования является среда разработки приложений Unity3D и использование ее возможностей для создания 3D-объектов в 2D-приложении. В качестве предмета исследования выбираем разработку и исследование характеристик образовательного мобильного приложения по стереометрии для ОС Android.

**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 —** а) теоретические: изучение литературы, посвященной проблеме проектирования трехмерных и двумерных изображений; б) практические: обобщение опыта работ в 2D и 3D-графики, моделирование, проектирование спецификации и разработка образовательного приложения по стереометрии для платформы Android.

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

**Область применения** — виртуальное моделирование образовательных систем с помощью средств разработки компьютерных игр.

РЭФЕРАТ

Курсавая работа, 38 с., 18 рыс., 2 табліцы

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

**Аб'ект даследавання** **—** аб’ектам даследавання з’яўляецца асяроддзе распрацоўкі прыкладанняў Unity3D і выкарыстанне яго магчымасцяў для стварэння 3D-аб'ектаў у 2D-прыкладанні. У якасці прадмета даследавання выбіраем распрацоўку і даследаванне характарыстык адукацыйнага мабільнага прыкладання па стэрэаметрыі для АС Android.

**Мэты працы** **—** разглядзець 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** **—** а) тэарэтычныя: вывучэнне літаратуры, прысвечанай праблеме праектавання трохмерных і двухмерных малюнкаў; б) практычныя: абагульненне вопыту работ у 2D і 3D-графікі, мадэляванне, праектаванне спецыфікацыі і распрацоўка адукацыйнага прыкладання па стэрэаметрыі для платформы Android.

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

**Вобласць ужывання** **—** віртуальнае мадэляванне адукацыйных сістэм з дапамогай сродкаў распрацоўкі камп'ютэрных гульняў.

ESSAY

Course work, 38 p., 18 illustrations, 2 tables.

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

**Object of research —** the object of research is the Unity3D application development environment and the use of its capabilities for creating 3D objects in a 2D application. As the subject of the study, we choose the development and study of the characteristics of the educational mobile application on stereometry for the Android OS.

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

**Methods of research** **—** a) theoretical: a study of the literature devoted to the problem of designing three-dimensional and two-dimensional images; b) practical: summarizing the experience of working in 2D and 3D graphics, modeling, designing specifications and developing educational applications for stereometry for the Android platform.

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

**Scope** **—** virtual modeling of educational systems using computer games development tools.

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 8](#_gjdgxs)

[**ГЛАВА 1 ТЕХНОЛОГИИ РАЗРАБОТКИ МОБИЛЬНЫХ ПРИЛОЖЕНИЙ 2D- И 3D- ГРАФИКИ** 10](#_30j0zll)

[1.1 2D- и 3D-графика в мобильных приложенияx 10](#_1fob9te)

[1.1.1 Пакет android.graphics 10](#_3znysh7)

[1.1.2 OpenGL ES 11](#_2et92p0)

[1.1.3 Виртуальная игровая среда Unreal Engine 13](#_tyjcwt)

[1.1.4 Виртуальная игровая среда Unity3D 13](#_3dy6vkm)

[1.2. Графические возможности Unity3D 14](#_1t3h5sf)

[1.2.1 Меши 14](#_4d34og8)

[1.2.2 Освещение пользовательского меша 15](#_2s8eyo1)

[1.2.3 Текстурирование 16](#_3rdcrjn)

[1.2.4 Mesh Renderer 17](#_26in1rg)

[1.2.5 Взаимодействие с пользователем 17](#_lnxbz9)

[**ГЛАВА 2 РАЗРАБОТКА ТРЕБОВАНИЙ И ПРОЕКТИРОВАНИЕ ПРОТОТИПА ПРИЛОЖЕНИЯ** 18](#_35nkun2)

[2.1. Обзор приложений для решения геометрических и стереометрических задач 18](#_1ksv4uv)

[2.1.1 Euclidea 18](#_44sinio)

[2.2. Задачи проекта 18](#_2jxsxqh)

[2.3. Разработка требований 19](#_z337ya)

[2.4. Проектирование спецификации приложения 19](#_3j2qqm3)

[2.2.1 Диаграмма вариантов использования 19](#_1y810tw)

[2.2.2 Объектная модель 21](#_2bn6wsx)

[2.2.3 Диаграммы деятельности 22](#_3as4poj)

[2.5. Проектирование прототипа приложения 24](#_49x2ik5)

[**ГЛАВА 3 РАЗРАБОТКА МОБИЛЬНОГО ПРИЛОЖЕНИЯ ДЛЯ ПЛАТФОРМЫ ANDROID** 27](#_3o7alnk)

[3.1. Игровые экраны 27](#_23ckvvd)

[3.1.1 Главное меню 27](#_ihv636)

[3.1.2 Экран со списком уровней 27](#_1hmsyys)

[3.1.3 Игровой режим 28](#_2grqrue)

[3.2. Построение главной фигуры уровня 29](#_3fwokq0)

[3.2.1 Prefabs 29](#_1v1yuxt)

[3.2.2 Создание возможности определения нажатия пользователем на элементы фигуры 30](#_4f1mdlm)

[3.3. Хранение данных 30](#_2u6wntf)

[3.3.1 Хранение информации об уровнях 30](#_19c6y18)

[3.3.2 Сохранение прогресса пользователя между запусками приложения 31](#_3tbugp1)

[3.4. Использование общих принципов проектирования 32](#_28h4qwu)

[3.4.1 Принципы SOLID 32](#_nmf14n)

[3.4.2 Шаблоны проектирования 33](#_37m2jsg)

[3.5. Реализация возможности создания объектов на геометрической фигуре пользователем 33](#_1mrcu09)

[3.6. Пример прохождения уровня 35](#_3l18frh)

[ЗАКЛЮЧЕНИЕ 37](#_1egqt2p)

[СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ 39](#_3ygebqi)

[ПРИЛОЖЕНИЯ 40](#_2dlolyb)

[*Приложение А* 40](#_sqyw64)

[*Приложение Б* 41](#_3cqmetx)

[*Приложение В* 42](#_1rvwp1q)

[*Приложение Г* 43](#_4bvk7pj)

[*Приложение Д* 44](#_2r0uhxc)

[*Приложение Е* 46](#_1664s55)

[*Приложение Ж* 50](#_3q5sasy)

**ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ**

|  |  |
| --- | --- |
| OpenGL ES | Open Graphics Library for Embedded Systems, |
| XML | расширяемый язык разметки (Extensible Markup Language), |
| API | программный интерфейс приложения (Application Programming Interface), |
| UML | унифицированный язык моделирования (Unified Modeling Language), |
| ООП | объектно-ориентированное программирование, |
| КОП | компонентно-ориентированное программирование, |
| LINQ | язык интегрированных запросов (Language Integrated Query), |
| DOM | объектная модель документа (Document Object Model), |
| JSON | JavaScript Object Notation. |

**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 в этой сфере.

Приложение разрабатывается для платформы Android.

Объектом исследования являются 2D- и 3D-объекты, с которыми пользователь может взаимодействовать, и их создание в приложениях для платформы Android, а также виртуальная среда Unity3D и ее возможности для создания таких объектов.

Методы исследования, которые применялись в работе, включают в себя изучение литературы, посвященной проблеме проектирования трехмерных и двумерных изображений, обобщение опыта работ в реализации двумерной и трехмерной графики, моделирование, проектирование спецификации и разработка образовательного приложения по стереометрии для платформы Android.

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 стереометрии. Разработана спецификация и требования к функционалу приложений. В частности, представлены диаграммы вариантов использования, деятельности и другое.

В главе 3 представлены результаты реализации мобильного приложения. Также описаны принципы и технологии, использованные в процессе реализации.

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

* 1. **1.1 2D- и 3D-графика в мобильных приложенияx**

Существует множество способов проектирования мобильных графических приложений 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, библиотеку OpenGL, а также возможности, которые предоставляют игровые среды и библиотеки.

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 [2]. 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 [6]:

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 — небольших программ-сценариев, позволяющих настраивать параметры обработки графики аппаратным обеспечением устройства. 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 [4]. Событие происходит только в случае, если во время касания позиция касания изменилась. Код представлен на языке Java.

* + 1. **1.1.3 Виртуальная игровая среда Unreal Engine**

Две наиболее популярных игровых среды — это Unreal Engine и Unity3D.

Unreal Engine — это игровая среда, разработанная Epic Games, впервые представленная в 1998 году. Unreal Engine использует С++. Используя данную среду, можно разрабатывать игры для iOS и Android. В среде есть мощный редактор, заключающий в себе несколько узкоспециальных редакторов. Некоторые редакторы даже могут заменить определённые программы. Unreal Engine предлагает один из наилучших механизмов рендеринга — процесса получения изображения по модели с помощью компьютерной программы на рынке.

Unreal Engine 4 имеет решение, позволяющее не писать много кода на C++. Это Blueprint — редактор визуального написания скриптов. 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 Виртуальная игровая среда Unity3D**

Unity3d — это мощный мультиплатформенный инструмент для разработки десктопных, мобильных и веб-приложений 2D- и 3D-графики, обрабатываемой в реальном времени [8].

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 дополнительных модулей для использования 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 примитивы 3D-объектов. К ним относятся куб, сфера, цилиндр, капсула. Но также существует возможность определить свои трехмерные объекты.

* + 1. **1.2.1 Меши**

Свои трехмерные объекты определяются с помощью 3D мешей. 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 [5].

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 отрисовывается, 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 [5]. Для правильного затенения при освещении каждой вершине 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 направленный свет анфас 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).

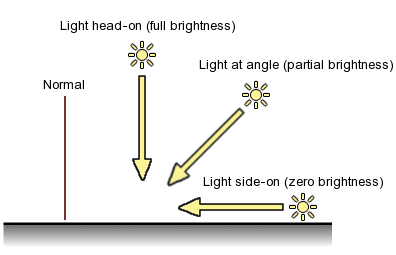


Рисунок 1.1 – Уровень яркости в зависимости от расположения света к вектору нормали

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 [7]. 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 значений вдоль ребра. Очевидный пример, 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 ресурс 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 в скрипте нужного объекта реализовать функцию OnMouseDrag(). Пример реализации в Приложение 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. Обзор приложений для решения геометрических и стереометрических задач**
     1. **2.1.1 Euclidea**

Аналогом разработанного приложения является приложение Euclidea. Это приложение является игрой, разбитой на уровни, в каждом из которых нужно решить задачу на построение на плоскости.

Приложение создано для платформы Android, имеет удобный интерфейс и ряд ключевых возможностей:

* Имеется большое количество инструментов, которые можно открыть в ходе прохождения уровней. Таким образом, увеличивается разнообразие уровней;
* Пользователю предоставлено большое количество уровней разной сложности. Уровни открываются постепенно;
* Пользователь может просмотреть справочный материал;
* Можно сбросить прогресс и пройти игру заново;
* Ведётся статистика прохождения уровней. За каждый уровень можно получить от одной до трех звёзд;
* Есть возможность авторизации.

Как видно из списка функций, данное приложение является хорошим примером обучающей игры для решения задач на построение на плоскости.

Euclidea имеет более одного миллиона пользователей, оценку 4.8 из 5 по результатам 68954 отзывов. Полученная статистика показывает, что такого рода приложения пользуются успехом и востребованы. Но аналогов для решения задач на построение в трехмерном пространстве на платформе Android нет.

* 1. **2.2. 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. **2.3. 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;
* Возможность вращения фигуры во время прохождения уровня;
* Возможность просмотреть условие уровня в любой момент времени;
* Отмена действия;
* Сброс прогресса;

Нефункциональные требования описывают характер поведения системы.

* Простота в использовании для любого пользователя;
* Интуитивно понятный интерфейс.
  1. **2.4. Проектирование спецификации приложения**

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 загружает игру и авторизуется (Рисунок 2.1 – Диаграмма вариантов использования «Новая игра»).

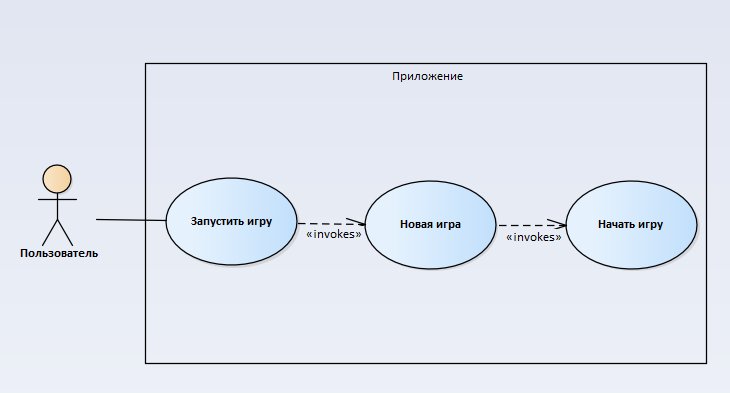


Рисунок 2.1 – Диаграмма вариантов использования «Новая игра»

1. Игрок заходит во второй раз и более (Рисунок 2.2 – Диаграмма вариантов использования «Продолжить игру»).

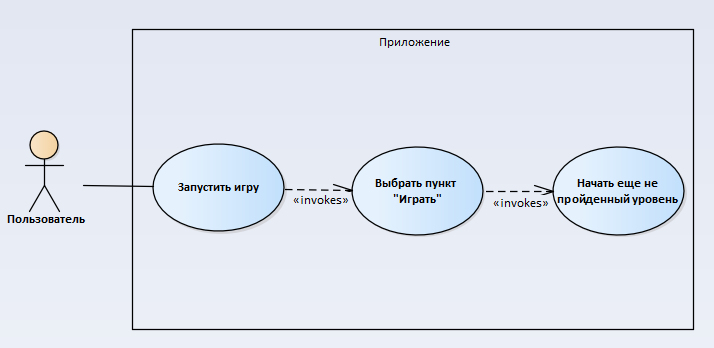


Рисунок 2.2 – Диаграмма вариантов использования «Продолжить игру»

1. Игрок переигрывает уровень N (Рисунок 2.3 – Диаграмма вариантов использования «Выбрать уровень N»).

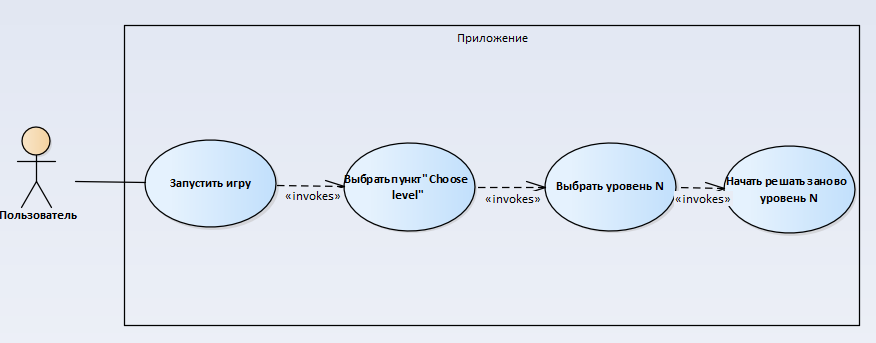


Рисунок 2.3 – Диаграмма вариантов использования «Выбрать уровень N»

1. Игрок сбрасывает свой прогресс (Рисунок 2.4 – Диаграмма вариантов использования «Сбросить прогресс»).

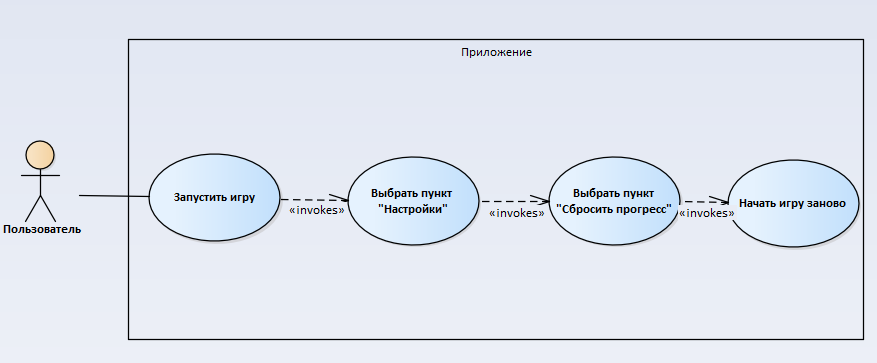


Рисунок 2.4 – Диаграмма вариантов использования «Сбросить прогресс»

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 среды, наиболее приспособлена 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 | Прозрачный многогранник, у которого выделены ребра, вершины. Вершинам присвоены буквенные обозначения. | Фигура может вращаться при определенных нажатиях на экран пользователем. У фигуры можно выделять ребра и вершины. |
| Уровень | Номер уровня, описание, название используемой фигуры, координаты точек, прямых и букенных обозначений точек, относящихся к условию, координаты решения. | Определяет данные каждого уровня. Загружаются при загрузке. |
| Кнопка рисования |  | Рисует объект согласно тому, что выбрал пользователь. |
| Игровая сессия | Выбранный уровень, шаги, выполненные пользователем с начала момента загрузки уровня, информация о расположении файлов, необходимых для загрузки уровня. | Содержит информацию, необходимую для выполнения основных функций во время игрового процесса. |

* + 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:

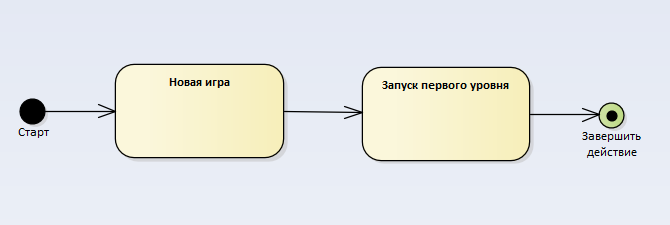


Рисунок 2.5 – Диаграмма деятельности «Начать новую игру»

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.

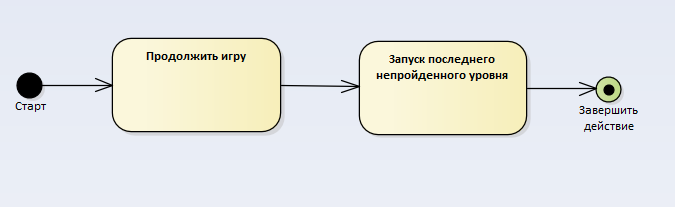


Рисунок 2.6 – Диаграмма деятельности «Продолжить игру»

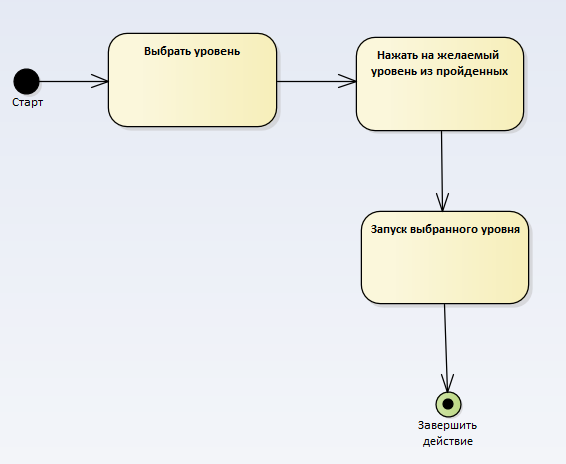


Рисунок 2.7 – Диаграмма деятельности «Выбрать уровень N»

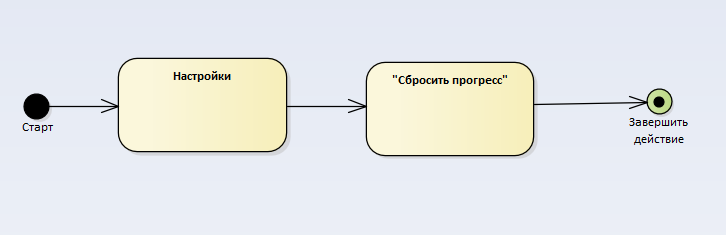


Рисунок 2.8 – Диаграмма деятельности «Сбросить прогресс»

* 1. **2.5. Проектирование прототипа приложения**

Приложение будет иметь три экрана: первый — главное меню. Второй — список уровней. 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 содержит информацию об управлении в режиме игры.

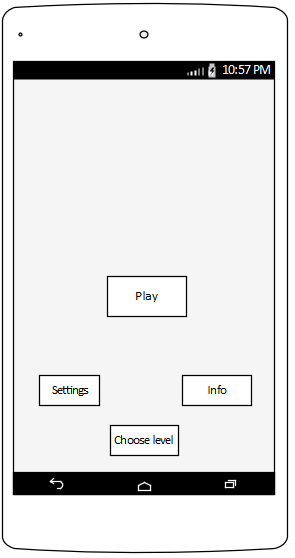


Рисунок 2.9 – Прототип главного экрана

На втором экране (Рисунок 3.2) отображается список уровней, также показано, какие уровни доступны, а какие нет.

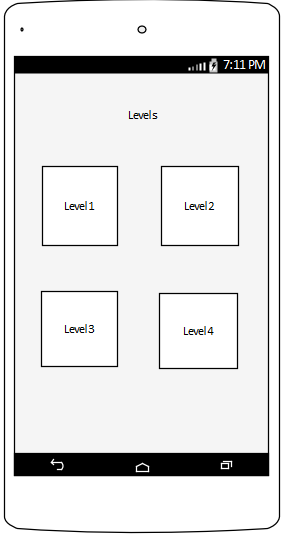


Рисунок 2.10 – Прототип экрана со списком уровней

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.

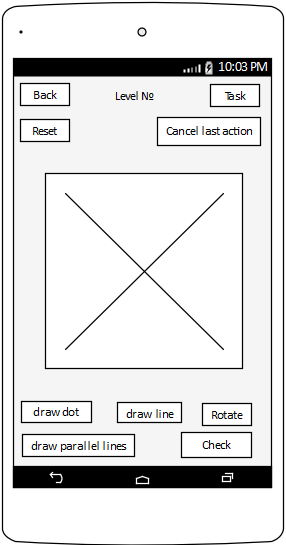


Рисунок 2.11 – Прототип режима игры

**ВЫВОДЫ**

1. Рассмотрено приложение, являющееся аналогом разработанного приложения, но ориентированном на построениях на плоскости.
2. Разработаны функциональные и нефункциональные требования к приложению.
3. Разработаны диаграммы, показывающие поведение пользователя в системе, а также объектная модель.
4. Спроектированы прототипы приложения.

# **РАЗРАБОТКА МОБИЛЬНОГО ПРИЛОЖЕНИЯ ДЛЯ ПЛАТФОРМЫ ANDROID**

* 1. **3.1. Игровые экраны**
     1. **3.1.1 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. Чтобы повторно пройти 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.

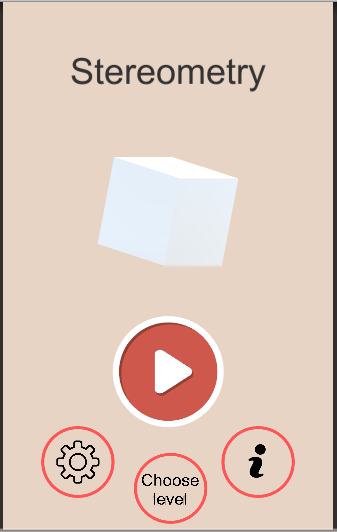


Рисунок 3.1 – Скриншот главного меню

* + 1. **3.1.2 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 открыл, следовательно, они не кликабельны. Также присутствует кнопка, с помощью которой пользователь может вернуться на главный экран.

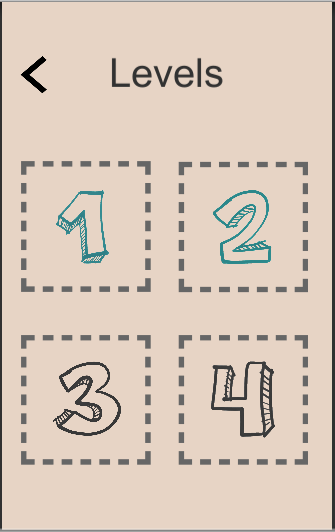


Рисунок 3.2 – Скриншот экрана со списком уровней

* + 1. **3.1.3 Игровой режим**

На этом экране происходит игровой процесс (Рисунок 3.3). У пользователя есть условие, которое показывается при открытии этого экрана, а также его можно посмотреть еще раз, нажав на верхнюю кнопку справа.

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. Проверяется это нажатием на кнопку «Check» и выделением вершин, входящих в сечение.

Построение выполняется с помощью кнопок, расположенных внизу экрана. Каждая из кнопок представляет, какой вид построения нужно выполнить (точка, прямая, параллельная прямая). Есть возможность вернуть фигуру в начальное положение с помощью кнопки «Rotate».

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

Кнопка выхода переводит пользователя на экран со списком уровней.

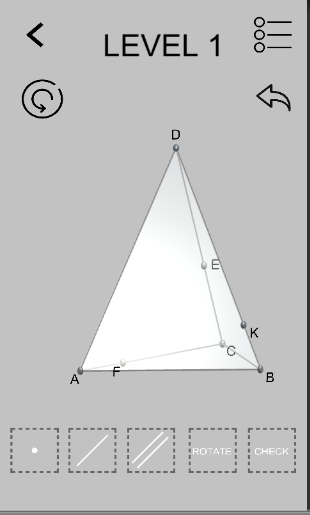


Рисунок 3.3 – Скриншот режима игры

* 1. **3.2. Построение главной фигуры уровня**

В первом уровне игры сечение выполняется на тетраэдре. Эта фигура не определена как примитив Unity, поэтому нужно было создавать свою фигуру. Как описывалось в первой главе, для создания необходимо определить свой объект класса Mesh, который записывается в поле Mesh Filter объекта и добавить создаваемому объекту Mesh Renderer, чтобы фигура отображалась (Приложение В).

Чтобы сделать тетраэдр прозрачным к нему был добавлен материал, у которого прозрачность alpha установлена в 0. А затем прорисованы ребра, являющиеся объектами у которых тип компонента Renderer’а — подтип LineRenderer, вершины — примитивы Unity сферы, а буквы — объекты с TextMesh в поле Mesh Filter. Также к фигуре добавлена возможность вращения при прикосновениях пользователя.

При запуске уровня загружаются данные уровня — точки, линии, буквенные обозначения точек, координаты ответа и его описание (Приложение Г). Также загружается главная фигура, тип которой мы получаем из данных уровня.

* + 1. **3.2.1 Prefabs**

Система префабов в Unity позволяет сохранять шаблонные объекты, в последствии префабы. Затем использовать их в приложении простой загрузкой неоходимого префаба. Эта техника позволяет избавиться от трудоемкого процесса создания одинаковых фигур. Все фигуры, в том числе и описанный выше тетраэдр, определяются как префабы и сохраняются.

Таким образом, при загрузке уровня, полученный тип указывает на необходимый префаб, который и используется на сцене.

* + 1. **3.2.2 Создание возможности определения нажатия пользователем на элементы фигуры**

Для того, чтобы пользователь мог создать некоторое построение, ему необходимо выделить объекты, которые участвуют в построении. К таким объектам относятся вершины, ребра фигуры и прямые, принадлежащие ей, точки в пространстве, которые используются в уровне.

Для этого от места нажатия перпендикулярно плоскости экрана посылается объект типа Ray, являющего своего рода лучом. Этот луч движется до тех пор, пока не попадает на объект, но для этого у объекта должен быть определён компонент — объект типа Collider. Такие компоненты позволяют определить фигуру объекта для физического взаимодействия с ним.

У вершин и точек — примитива Unity сфера — определён SphereCollider — наследник базового класса Collider. А для рёбер — объектов, имеющих в качестве компонента типа Renderer объект подтипа LineRenderer — создавались вручную CapsuleCollider с необходимыми размерами.

Рассматриваемые объекты довольно маленькие, выделить их становится затруднительно. Поэтому, увеличив размеры объектов типа Collider (в данном случае радиус), можно добиться удобства использования.

* 1. **3.3. Хранение данных** 
     1. **3.3.1 Хранение информации об уровнях**

Приложение организовано таким образом, чтобы уменьшить количество сцен, за счет создания одной сцены для игрового режима, а также вынесением данных об уровнях в отдельный файл.

Существует несколько техник для хранения данных мобильного приложения: использование XML-файла, JSON-файла или SQLite базы данных.

* SQLite — компактная встраиваемая СУБД. Хорошо использовать, если необходимо хранить большое количество данных, но нетривиальна в реализации соединения. Не создано библиотек-обёрток для упрощения работы с ней;
* JSON — текстовый формат обмена данными, основанный на JavaScript. Сериализованные объекты записываются как JavaScript объекты. На данный момент этот формат очень популярен. В Unity есть специальные встроенные классы для работы с JSON.

При разработке приложения выбор пал на XML-файлы за счет языковых возможностей C# при работе с XML-документами. C# предоставляет удобную технологию LINQ to XML. Она загружает в память компьютера документ в виде DOM. DOM — Document Object Model (объектная модель документа) — позволяет представить документ в виде дерева узлов, каждый узел которого представляет собой элемент, атрибут, текстовый или другой объект [1].

С помощью данной модели, называемой X-DOM, можно выполнять манипуляции над данными, как если бы это был массив объектов. Данная технология хорошо спроектирована и имеет высокую производительность.

В приложении нужна функция нахождения данных уровня по его номеру. С помощью LINQ to XML можно это реализовать с помощью лишь одного запроса.

После нахождения необходимого элемента в XML-документе, он преобразовывается в объект класса Level (Приложение Д).

* + 1. **3.3.2 Сохранение прогресса пользователя между запусками приложения**

В Unity определен класс PlayerPrefs. Он сохраняет и получает необходимую информацию для игры между игровыми сеансами. Этот способ хранения данных предоставляет очень быстрый доступ к самим данным.

На Android данные сохраняются на устройстве. Данные сохраняются в параметрах SharerPreferences. C # / JavaScript, Android Java и исходный 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.

В разработанном приложении PlayerPrefs используется для хранения номера уровня, на котором остановился пользователь.

* 1. **3.4. Использование общих принципов проектирования**
     1. **3.4.1 Принципы SOLID**

SOLID — набор принципов проектирования, служащие шаблоном для написания легко поддерживаемого и расширяемого программного обеспечения.

***S — Single Responsibility Principle*** (принцип единственной ответственности) говорит, что должна быть одна и только одна причина для изменения класса или класс должен иметь только одну ответственность. Принцип единственной ответственности предназначен для борьбы со сложностью. Зависимость между числом строк кода и сложностью решения является нелинейной. Добавление каждой новой функции в систему требует все больше и больше усилий. При разработке крупной системы очень важно иметь возможность сосредоточиться на главной задаче метода, класса или модуля и выбросить из рассмотрения все второстепенные детали.

***O — Open/Closed Principle*** (принцип открытости/закрытости) говорит о том, что класс должен быть открыт для расширений, но закрыт для модификаций. Имеется в виду, что расширение функциональности класса должно осуществляться посредством включения каких-либо зависимостей в класс, а не посредством изменения кода данного класса. А эти зависимости не должны нарушать логику класса.

***L — Liskov Substitution Principle*** (принцип подстановки Лисков) заключается в следующем: должна быть возможность вместо базового типа подставить любой его подтип. Этот принцип основан на принципах проектирования по контракту, а именно:

* Производные классы не должны усиливать предусловия;
* Производные классы не должны ослаблять постусловия;
* Производные классы не должны нарушать инварианты базового класса;
* Производные классы не должны генерировать исключения, не описанные базовым классом.

Этот принцип позволяет согласовать поведение наследника и базового класса.

***I — Interaface Segregation Principle*** (принцип разделения интерфейсов) говорит, что клиенты не должны вынужденно зависеть от методов, которыми не пользуются. Принцип разделения интерфейса предназначен для получения простого и слабосвязного кода. Чем больше у класса зависимостей, тем сложнее понять его роль, сложнее тестировать и использовать повторно. К тому же большое число связей увеличивает вероятность поломки класса при изменении зависимостей.

***D — Dependency Inversion Principle*** (принцип инверсии зависимостей) гласит: «Модули верхнего уровня не должны зависеть от модулей нижнего уровня. И те и другие должны зависеть от абстракций. Абстракции не должны зависеть от деталей. Детали должны зависеть от абстракций» [9].

Разработанное приложение соответствует описанным выше принципам. Все классы отвечают за одну ответственность, легко расширяемые, но закрыты для изменений, построена правильная иерархия наследования, интерфейсы независимы.

Например, за создание объектов, не привязанных к окружению, отвечает один класс, за математические расчеты другой, за отображение объекта на экране третий. Выделен интерфейс загрузки данных уровня из файла. Следовательно, можно будет легко перевести приложение на хранение данных в JSON-файле, например.

Таким образом, код приложения расширяемый и слабосвязанный.

* + 1. **3.4.2 Шаблоны проектирования**

Шаблоны проектирования — рекомендации по решению некоторых распространенных задач. Шаблоны проектирования служат для решения часто встречаемых задач. В данном приложении наиболее часто были использованы следующие шаблоны: «Стратегия» и «Адаптер».

С помощью шаблона «Стратегия» можно писать слабосвязанный код, засчет включения конкретных реализаций необходимых частей алгоритма. Он был использован, например, при передаче стратегии загрузки данных из файла путём указания в качестве параметра метода объект типа интерфейса.

Шаблон «Адаптер» позволяет объектам с несовместимыми интерфейсами работать вместе. В разработанном приложении он использовался при организации создания возможности сериализации класса.

* 1. **3.5. Реализация возможности создания объектов на геометрической фигуре пользователем**

Для того, чтобы пройти уровень игры пользователю необходимо с помощью инструментов построить сечение. Инструментами являются построение прямой, точки пересечения прямых и прямой, параллельной данной, проходящей через заданную точку (Приложение Е).

Для выполнения любого из построений пользователь нажимает кнопку с желаемым объектом построения, выделяет необходимые для этого объекты (так для построения прямой нужно выделить две точки), а потом нажимает кнопку, означающую подтверждение построения.

Для построения прямых используется параметрическое представление этой прямой, которое легко строится, зная координаты двух точек (x0, y0, z0) и (x1, y1, z1) (1):

(1)

Тогда подставляя вместо p любые рациональные числа, получим координаты точек, принадлежащих данной прямой.

По похожему принципу строится уравнение прямой, параллельной данной, проходящей через заданную точку (a, b, c). Также необходимо взять две точки, через которые проходит прямая (x0, y0, z0) и (x1, y1, z1). Тогда параметрическое уравнение искомой прямой будет выглядеть следующим образом (2):

(2)

А для построения точки пересечения двух прямых необходимо составить систему линейных уравнений (3):

(3)

Здесь (x0, y0, z0) и (x1, y1, z1) — точки, принадлежащие одной прямой, а (x2, y2, z2) и (x3, y3, z3) — другой, p1 и p2 — неизвестные.

Решив данную систему, например, методом Гаусса (Приложение Ж), находятся p1 и p2. Подставив p1 или p2 в одно из уравнений прямых, полученных формулой (1), вычисляются значения координат точки пересечения.

* 1. **3.6. Пример прохождения уровня**

При загрузке приложения появляется условие уровня. Его можно просмотреть в любой другой момент игры (Рисунок 3.4).

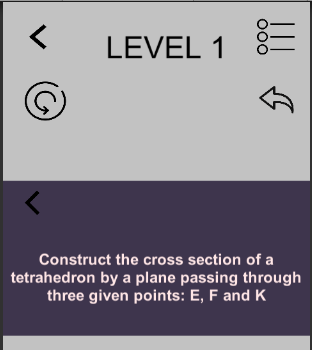


Рисунок 3.4 – Скриншот описания уровня

Начальный вид уровня был представлен на рисунке выше (Рисунок 3.3). То есть нужно построить сечение тетраэдра, проходящее через точки E, F и K. Для этого используются следующие инструменты: построение прямой и построение точки пересечения двух прямых. Итоговое построение выглядит следующим образом (Рисунок 3.5):

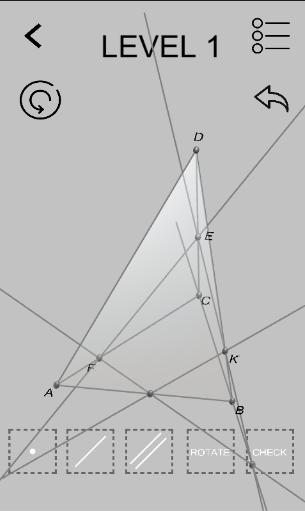


Рисунок 3.5 – Скриншот итогового построения сечения

По окончании построения сечения нужно проверить его. Делается это с помощью кнопки «Check». После того, как выяснится, что сечение построено верно, появляется окно, говорящее об успешном завершении уровня (Рисунок 3.6).

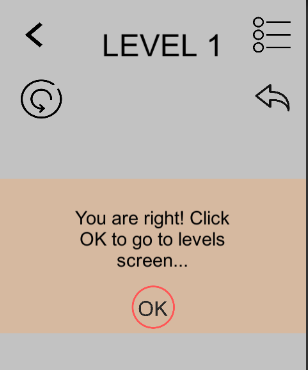


Рисунок 3.6 – Скриншот окна, говорящем об успешном завершении уровня

Далее открывается второй уровень, и так по цепочке. С каждым уровнем сложность увеличивается.

**ВЫВОДЫ**

1. Разработаны сцены приложения.
2. Реализован функционал и создан удобный и понятный интерфейс.
3. Добавлены уровни в приложение.
4. Использованы принципы проектирования SOLID и шаблоны проектирования при написании кода приложения.

**ЗАКЛЮЧЕНИЕ**

Среди образовательных технологий большую роль играют информационные технологии и программные средства. С ростом рынка мобильных телефонов активно развивается и рынок мобильных приложений, включая образовательные приложения. В настоящее время мобильные смартфоны есть практически у всех и являются удобным инструментом для доступа к информации, включая образовательную.

Раздел «Стереометрия» является одним из сложных разделов школьного курса геометрии. Решение задач по стереометрии с помощью мобильного приложения позволит повысить качество и интенсивность обучения, а также позволит продемонстрировать задачи наглядно, используя возможности среды Unity3D для создания трехмерных объектов. Приложение «Stereometry» создано для решения задач на построение сечений 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 диаграммы UML: вариантов использования и деятельности.
5. Разработана объектная модель приложения.
6. Разработаны прототипы приложения.
7. Изучены принципы проектирования SOLID.
8. Изучены шаблоны проектирования.
9. Изучены возможности языка программирования C# при работе с XML-документами.
10. Реализовано приложение.
11. Создан удобный и понятный функционал для пользователя.

Для проектирования приложения использовалась среда Enterprise Architect Version 13.

Для реализации приложения использовались среды Unity 2018.2.10f1 и Visual Studio 2019.

**СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ**

* 1. Albahari, J. C# 7.0 in a Nutshell / J. Albahari, B. Albahari. — Sebastopol, CA: O’Reilly Media, Inc., 2017. — 1071 p: im.
  2. Android Developers [Electronic resource] / Documentation for Android Operation System – Mode of access: <http://developer.android.com/index.html>. – Date of access: 15.11.2018
  3. Android для программистов: создаём приложения / П. Дейтел [и др.]; по общ. ред. П. Дейтела. – СПб.: Питер, 2014. – 560 с.: ил.
  4. Developers [Electronic resource] – Mode of access: https://stuff.mit.edu/afs/sipb/project/android/docs/index.html. – Date of access: 17.11.2018
  5. Unity Technologies [Electronic resource] / Documentation for Unity – Mode of access:<https://docs.unity3d.com/Manual/index.html>. – Date of access: 15.04.2019
  6. Введение в OpenGL [Электронный ресурс] / Лекция 19 – Режим доступа: https://www.bsu.by/Cache/Page/353453.pdf. – Дата доступа: 14.11.2018
  7. Мэннинг Д. Unity для разработчика. Мобильные мультиплатформенные игры / Д. Мэннинг, П. Батфилд-Эддисон. – СПб.: Питер, 2018. – 352 с: ил.
  8. Степчева, З. Основы геометрического моделирования в Unity3D / З. Степчева // Методические указания [Электронный ресурс]. – 2012. – Режим доступа: http://venec.ulstu.ru/lib/disk/2013/Stepcheva,Hodos.pdf. – Дата доступа: 21.11.2018
  9. Тепляков, С. Паттерны проектирования на платформе .NET / С. Тепляков. — СПб.: Питер, 2015. — 320 с: ил.

**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 UnityEditor;

using UnityEngine;

public class TetrahedronCreator : MonoBehaviour

{

public float height = 1;

public float width = 1.2f;

public float length = 1.2f;

private void Start()

{

var meshFilter = gameObject.AddComponent<MeshFilter>();

var mesh = new Mesh();

var widthOffset = width \* 0.5f;

var lengthOffset = length \* 0.5f;

var points = new Vector3[]

{

new Vector3(-widthOffset, -height / 2, -lengthOffset),

new Vector3(widthOffset, -height / 2, -lengthOffset),

new Vector3(widthOffset, -height / 2, lengthOffset),

new Vector3(0.2f, height / 2, 0.3f)

};

mesh.vertices = new Vector3[]

{

points[0], points[1], points[2],

points[0], points[1], points[3],

points[0], points[2], points[3],

points[1], points[2], points[3]

};

mesh.triangles = new int[]

{

0, 1, 2,

5, 4, 3,

6, 7, 8,

11, 10, 9

};

mesh.RecalculateNormals();

mesh.RecalculateBounds();

meshFilter.mesh = mesh;

gameObject.AddComponent<MeshRenderer>().material = Resources.Load<Material>("Materials/Transparent");

}

* 1. *Приложение Г*

using UnityEngine;

public class MainShapeBehaviour : MonoBehaviour

{

private float \_rotSpeed = 3;

private ObjectCreator \_objectCreator;

void Start ()

{

\_objectCreator = new ObjectCreator();

CreateLevel();

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 CreateLevel()

{

foreach (Line line in Game.CurrentLevelData.Lines)

{

\_objectCreator.CreateLine(line, transform, Resources.Load<Material>(Game.PathToLinesMaterial));

}

foreach (SerializableVector3 vertice in Game.CurrentLevelData.Vertices)

{

\_objectCreator.CreateDot(vertice, transform, Resources.Load<Material>(Game.PathToDotsMaterial));

}

foreach (Letter letter in Game.CurrentLevelData.Letters)

{

\_objectCreator.CreateLetter(letter, transform, Color.black);

}

}

}

* 1. *Приложение Д*

using System.Linq;

using System.Xml.Linq;

using System.Xml.Serialization;

public class XmlDataLoader : IDataLoader

{

private readonly string \_xmlText;

public XmlDataLoader(string xmlText)

{

\_xmlText = xmlText;

}

public Level LoadLevel(int number)

{

XDocument xDocument = XDocument.Parse(\_xmlText);

XElement levelNode = xDocument.Element("Levels").Elements("Level").First(e => int.Parse(e.Element("Number").Value) == number);

var serializer = new XmlSerializer(typeof(Level));

return (Level) serializer.Deserialize(levelNode.CreateReader());

}

}

using System;

using System.Collections.Generic;

[Serializable]

public class Level

{

public Level()

{

}

public int Number { get; set; }

public string Description { get; set; }

public ShapeType Type { get; set; }

public List<Line> Lines { get; set; }

public List<SerializableVector3> Vertices { get; set; }

public List<SerializableVector3> Answer { get; set; }

public List<Letter> Letters { get; set; }

}

[Serializable]

public enum ShapeType

{

Cube,

Tetrahedron

}

* 1. *Приложение Е*

using System;

using System.Collections.Generic;

using UnityEngine;

public static class LineEquationAlgorithms

{

private static readonly float \_approximation = 1e-5f;

public static List<Func<float, float>> CreateLineEquation(Vector3 firstPoint, Vector3 secondPoint)

{

return CreateLineEquationOfParallelLine(firstPoint, secondPoint, firstPoint);

}

public static List<Func<float, float>> CreateLineEquationOfParallelLine(Vector3 firstPoint, Vector3 secondPoint, Vector3 dot)

{

var equations = new List<Func<float, float>>

{

p => dot.x + (secondPoint.x - firstPoint.x) \* p,

p => dot.y + (secondPoint.y - firstPoint.y) \* p,

p => dot.z + (secondPoint.z - firstPoint.z) \* p

};

return equations;

}

public static Vector3 FindLinesIntersection(Line firstLine, Line secondLine, ILinearEquationsSolver solver)

{

float[][] equations = CreateMatrix(firstLine, secondLine);

float[] vector = CreateVector(firstLine, secondLine);

int lineToRemove = GetLineToRemove(equations, vector);

float[][] equationsForGauss = RemoveRedundantLine(equations, lineToRemove);

float[] vectorForGauss = RemoveRedundantLine(vector, lineToRemove);

solver.SolveSystemOfLinearEquastions(equationsForGauss, vectorForGauss);

var equationForFirstLine = CreateLineEquation(firstLine.StartPoint, firstLine.EndPoint);

return new Vector3(equationForFirstLine[0](vectorForGauss[0]), equationForFirstLine[1](vectorForGauss[0]), equationForFirstLine[2](vectorForGauss[0]));

}

private static float[][] CreateMatrix(Line firstLine, Line secondLine)

{

return new[]

{

new[] {firstLine.EndPoint.x - firstLine.StartPoint.x, secondLine.StartPoint.x - secondLine.EndPoint.x},

new[] {firstLine.EndPoint.y - firstLine.StartPoint.y, secondLine.StartPoint.y - secondLine.EndPoint.y},

new[] {firstLine.EndPoint.z - firstLine.StartPoint.z, secondLine.StartPoint.z - secondLine.EndPoint.z}

};

}

private static float[] CreateVector(Line first, Line second)

{

return new[]

{

second.StartPoint.x - first.StartPoint.x,

second.StartPoint.y - first.StartPoint.y,

second.StartPoint.z - first.StartPoint.z

};

}

private static float[][] RemoveRedundantLine(float[][] matrix, int redundantLine)

{

float[][] newMatrix = new float[matrix.Length - 1][];

for (int i = 0; i < redundantLine; i++)

{

newMatrix[i] = matrix[i];

}

for (int i = redundantLine + 1; i < matrix.Length; i++)

{

newMatrix[i - 1] = matrix[i];

}

return newMatrix;

}

private static float[] RemoveRedundantLine(float[] vector, int redundantLine)

{

float[] newVector = new float[vector.Length - 1];

for (int i = 0; i < redundantLine; i++)

{

newVector[i] = vector[i];

}

for (int i = redundantLine + 1; i < vector.Length; i++)

{

newVector[i - 1] = vector[i];

}

return newVector;

}

private static int GetLineToRemove(float[][] matrix, float[] vector)

{

if (matrix[0][0].ApproximatelyEquals(0, \_approximation) && matrix[0][1].ApproximatelyEquals(0, \_approximation))

{

return 0;

}

if (matrix[1][0].ApproximatelyEquals(0, \_approximation) && matrix[1][1].ApproximatelyEquals(0, \_approximation))

{

return 1;

}

if (matrix[2][0].ApproximatelyEquals(0, \_approximation) && matrix[2][1].ApproximatelyEquals(0, \_approximation))

{

return 2;

}

if (ProportionalLines(matrix, vector, 0, 1))

{

return 1;

}

return 2;

}

private static bool ProportionalLines(float[][] matrix, float[] vector, int i, int j)

{

float firstQuotient = matrix[i][0] / matrix[j][0];

float secondQuotient = matrix[i][1] / matrix[j][1];

float thirdQuotient = vector[i] / vector[j];

if (firstQuotient.ApproximatelyEquals(secondQuotient, \_approximation) &&

firstQuotient.ApproximatelyEquals(thirdQuotient, \_approximation))

{

return true;

}

if (float.IsNaN(firstQuotient) && secondQuotient.ApproximatelyEquals(thirdQuotient, \_approximation))

{

return true;

}

if (float.IsNaN(secondQuotient) && firstQuotient.ApproximatelyEquals(thirdQuotient, \_approximation))

{

return true;

}

if (float.IsNaN(thirdQuotient) && firstQuotient.ApproximatelyEquals(secondQuotient, \_approximation))

{

return true;

}

return false;

}

}

* 1. *Приложение Ж*

public class LinearEquationsSolverWithGaussMethod : ILinearEquationsSolver

{

public void SolveSystemOfLinearEquastions (float[][] matrix, float[] vector)

{

SwapLinesIfFirstElementIsZero(matrix, vector);

DirectStep(matrix, vector);

ReverseStep(matrix, vector);

DivideElements(matrix, vector);

}

private void DirectStep(float[][] matrix, float[] vector)

{

int dimension = vector.Length;

for (int k = 0; k < dimension - 1; k++)

{

for (int i = k + 1; i < dimension; i++)

{

float multiplier = matrix[i][k] / matrix[k][k];

for (int j = k; j < dimension; j++)

{

matrix[i][j] -= matrix[k][j] \* multiplier;

}

vector[i] -= vector[k] \* multiplier;

}

}

}

private void ReverseStep(float[][] matrix, float[] vector)

{

int dimension = vector.Length;

for (int k = dimension - 1; k >= 0; k--)

{

for (int i = k - 1; i >= 0; i--)

{

float multiplier = matrix[i][k] / matrix[k][k];

for (int j = dimension - 1; j >= k; j--)

{

matrix[i][j] -= matrix[k][j] \* multiplier;

}

vector[i] -= vector[k] \* multiplier;

}

}

}

private void DivideElements(float[][] matrix, float[] vector)

{

for (int i = 0; i < vector.Length; i++)

{

vector[i] /= matrix[i][i];

}

}

private void SwapLinesIfFirstElementIsZero(float[][] matrix, float[] vector)

{

float approximation = 1e-10f;

if (matrix[0][0].ApproximatelyEquals(0, approximation))

{

Swap(ref matrix[0], ref matrix[1]);

Swap(ref vector[0], ref vector[1]);

}

}

private void Swap(ref float[] a, ref float[] b)

{

float[] tmp = a;

a = b;

b = tmp;

}

private void Swap(ref float a, ref float b)

{

float tmp = a;

a = b;

b = tmp;

}

}