МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

ЧУО «Минский КОЛЛЕДЖ предпринимательства»

**Приложение на андроид «Color name»**

Пояснительная записка к курсовому проекту по учебной дисциплине

«Конструирование программ и языки программирования»

КПП31015.104.081ПЗ

**Автор проекта**

Учащегося 3 курса группы П-31 /А. В. Масловский /

**Руководитель**

Преподаватель /А. В. Чеботарёв/

Минск, 2016

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

ЧУО «МИНСКИЙ КОЛЛЕДЖ ПРЕДПРИНИМАТЕЛЬСТВА»

Специальность

«Программное обеспечение информационных технологий» 2-40 01 01

УТВЕРЖДАЮ

Председатель цикловой комиссии

\_\_\_\_\_\_\_\_\_\_/ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/

**ЗАДАНИЕ**

**на курсовой проект (работу)**

Учащемуся Лабусу Вячеславу Анатольевичу

курса 3

группы П-31

по предмету «Конструирование программ и языки программирования»

Тема курсового проекта (работы) Игра «Pusher»

Исходные данные масса объекта, направление броска.

**Состав проекта (работы)**

**Пояснительная записка**

Содержание разделов Срок выполнения

Введение, Постановка задачи\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_23.03.2016\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Вычислительная система 08.04.2016\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Проектирование\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_15.04.2016\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Описание программы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_29.04.2016\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Отладка и испытание программы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_07.05.2016\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Описание применения программы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_19.05.2016\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Заключение\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_27.05.2016\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Графическая часть проекта**

Схема работы программы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_03.06.2016\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата выдачи «16» марта 2016 г.

Срок завершения «10» июня 2016 г.

**Преподаватель-руководитель курсового проекта** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Подпись учащегося**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Содержание**  [Введение 5](#_Toc453323295)  [1 Постановка задачи 7](#_Toc453323296)  [1.1 Формулировка задания 7](#_Toc453323297)  [1.2 Обоснование необходимости автоматизации.](#_Toc453323298) 7  [1.3 Определение данных и их представление.](#_Toc453323299) 8  [2 Вычислительная система](#_Toc453323300) 9  [2.1 Обоснование выбора языка программирования](#_Toc453323301) 9  [2.2 Обоснование выбора среды разработки](#_Toc453323302) 10  [2.3 Требования к конфигурации программного и аппаратного обеспечения](#_Toc453323303) 11  [3 Проектирование](#_Toc453323304) 12  [3.1 Проектирование интерфейса](#_Toc453323305) 12  [3.2 Описание алгоритмов](#_Toc453323306) 12  [3.3 Описание справочной системы](#_Toc453323308) 13  [4 Описание Программы](#_Toc453323309) 14  [4.1 Логическая структура](#_Toc453323310) 14  [4.2 Физическая структура](#_Toc453323311) 14  [4.3 Описание интерфейса](#_Toc453323312) 15  [5 Отладка и испытание программы 20](#_Toc453323313)  [5.1 Тестовые примеры 20](#_Toc453323314)  [5.2 Анализ полученных результатов 21](#_Toc453323315) | | | | | | | | | | |
|  |  |  |  |  | *КПП31015.104.081ПЗ* | | | | | |
|  |  |  |  |  |
| *Изм* | *Лист* |  | *Подпись* | *Дата* |
| *Разраб.* | | *Масловский* |  |  | *ПРИЛОЖЕНИЕ НА АНДРОИД*  *«COLOR NAME»* | *Лит.* | | | *Лист* | *Листов* |
| *Пров.* | | *Чеботарёв* |  |  |  | *у* |  | 3 | 43 |
| *Реценз.* | |  |  |  | *ЧУО МКП* | | | | |
| *Н.Контр.* | |  |  |  |
| *Утв.* | |  |  |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| [6 Описание применения программы](#_Toc453323316) 25  [6.1 Назначение программы](#_Toc453323317) 25  [6.2 Условия использования](#_Toc453323318) 25  [Заключение 26](#_Toc453323319)  [Список использованных источников 27](#_Toc453323320)  [Приложение 1 28](#_Toc453323321) | | | | | | |
|  |  |  |  |  | *КПП31015.104.081ПЗ* | *Лист* |
|  |  |  |  |  | 4 |
| *Изм* | *Лист* | *№ докум.* | *Подпись* | *Дата* |

# Введение

В данном курсовом проекте необходимо разработать игру «Pusher» реализованный в виде 3D симулятора.

Симуляторы – программные и аппаратные средства, создающие впечатление действительности, отображая часть реальных явлений и свойств в виртуальной среде. Чаще всего сейчас слово «симулятор» используется применительно к компьютерным программам (обычно играм).

В настоящее время существует много различных аналогов, например: модель поведения тел брошенных под углом к горизонту и Throw and leave. Данный курсовой проект отличается от других аналогов дизайном, возможностью передвигаться по игровому миру и переводом в 3D.

Основной целью выполнения курсового проекта является расширение, углубление знаний в области алгоритмизации и программирования на языке C# и в среде разработки Visual Studio 2013, расширение знаний о XNA и BEPUphysics, расширение знаний о 3D моделировании и формирование навыков научно-исследовательской деятельности.

Задачи выполнения курсового проекта:

* систематизация научных знаний в области алгоритмизации и программирования на языке C#;
* изучение технологии XNA и BEPUphysics;
* углубление уровня и расширение объема профессионально значимых знаний, умений и навыков работы в среде быстрой разработки Visual Studio;
* формирование умений и навыков самостоятельной организации научно-исследовательской работы;
* овладение современными методами поиска, обработки и использования информации.

Данная пояснительная записка содержит следующие разделы:

1. Раздел «Постановка задачи» содержит формулировку задачи с указанием всех функций проекта, определение круга задач, которые должны быть автоматизированы в данном курсовом проекте, описание входных и выходных данных проекта.
2. Раздел «Вычислительная система» содержит обоснование выбора языка программирования и среды разработки проекта, описание операционной системы, в которой была реализована программа. Требования конфигурации программного и аппаратного обеспечения.
3. Раздел «Проектирование» содержит описание внешнего пользовательского интерфейса, структуру меню, алгоритма, функционально независимых подзадач и путей их решения.
4. В разделе «Описание программы» содержится описание всех модулей, функций, входящих в проект, их назначение и взаимодействие, особенности интерфейса программы.
5. В разделе «Отладка и испытание программы» описывается проверка работы программы на различных тестах и анализ полученных результатов.
6. Раздел «Описание и применение программы» содержит сведения о назначении программного средства, области применения, требования к необходимым программным и техническим ресурсам.
7. В «Заключении» описывается краткая формулировка проблемы и пути ее решения, использованные методы и средства, возможность дальнейшей модификации проекта.

# 1 Постановка задачи

## Формулировка задания

Разработать игру «Pusher». Реализовать механику подбора предметов и возможностью кинуть их.

При старте игры игрок должен появится в игровом мире. Перед ним должны находиться пять разных объектов. У каждого вида объектов должна быть своя масса, которая при броске должна быстрее замедлять объект и притягивать его к земле. При подходе к объекту должен появиться индикатор, который сигнализирует о том, что можно этот объект подобрать. При нажатии на правую кнопку мыши объект должен быть подобран и находиться в руках у игрока. Этот объект должен постоянно следовать за игроком так, как это было бы в реальном мире. При повторном нажатие на правую кнопку мыши объект должен будет упасть на землю так, как будто его положил человек. Если пользователь нажмет на левую кнопку мыши, то объект должен будет отброшен так, как будто его человек выбросит в ту сторону, в которую он смотрит. Так же разработать возможность ходить по игровому миру, прыгать, приседать и ложиться на землю.

## 1.2 Обоснование необходимости автоматизации

Компьютерная игра – компьютерная программа, служащая для организации игрового процесса (геймплея), связи с партнёрами по игре, или сама выступающая в качестве партнёра.

Компьютерные игры могут быть классифицированы по нескольким признакам:

1. Жанр. Игра может принадлежать как к одному, так и к нескольким жанрам, а в уникальных случаях – открывать новый или быть вне всяких жанров;
2. Количество игроков и способ их взаимодействия. Игра может быть однопользовательской – рассчитанной на игру одного человека, или многопользовательской – рассчитанной на одновременную игру нескольких человек; а также вестись на одном компьютере, через интернет, электронную почту, или массово;
3. Визуальное представление. Игра может как использовать графические средства оформления, так и напротив, быть текстовой. Игра также может быть двухмерной или трехмерной. Есть и звуковые игры – в них вместо визуального представления используются звуки.
4. Платформа. Игра может принадлежать как к одной платформе, так и быть мультиплатформенной.

Если сравнивать данный курсовой проект с приложением «модель поведения тел брошенных под углом к горизонту» то можно заметить, что он отличается преобразованием проекта из 2D в 3D, а также возможностью передвигаться по игровому миру. Недостатком же данного проекта является в том, что нельзя изменять массу объектов, а так же силу притяжения.

## 1.3 Определение данных и их представление

Форматы входных и выходных данных являются частью этапа проектирования. Входные форматы должны быть разработаны с учетом максимального удобства для пользователя и минимальной возможности ошибок. Порядок переменных и форматы данных, привычные для пользователя, помогут избежать ошибок и облегчат использование программ. В данном проекте входными данными являются: курсор мыши, настройки поиска, данные авторизации.

Выходные спецификации могут сильно различаться. Иногда даются четкие инструкции и выходные данные подгоняются под определенный стандарт. Однако часто вообще отсутствуют какие-либо указания. Выходные данные подчас представляют собой: физика полета объекта с учетом его массы, силы и направления броска.

# 2 Вычислительная система

## 2.1 Обоснование выбора языка программирования

В качестве языка программирования, выбранного для разрабатываемого проекта выбран язык программирования C#. Преимуществами данного языка являются:

1. C# создавался параллельно с каркасом Framework .Net и в полной мере учитывает все его возможности как FCL, так и CLR.
2. С# является полностью объектно-ориентированным языком, где даже типы, встроенные в язык, представлены классами.
3. C# является мощным объектным языком с возможностями наследования и универсализации.
4. C# является наследником языков C/C++, сохраняя лучшие черты этих популярных языков программирования: общий с этими языками синтаксис, знакомые операторы языка облегчают переход программистов от С++ к C#.
5. Сохранив основные черты своего великого родителя, язык стал проще и надежнее. Простота и надежность, главным образом, связаны с тем, что на C# хотя и допускаются, но не поощряются такие опасные свойства С++ как указатели, адресация, разыменование, адресная арифметика.
6. Благодаря каркасу Framework .Net, ставшему надстройкой над операционной системой, программисты C# получают те же преимущества работы с виртуальной машиной, что и программисты Java. Эффективность кода даже повышается, поскольку исполнительная среда CLR представляет собой компилятор промежуточного языка, в то время как виртуальная Java-машина является интерпретатором байт-кода.
7. Мощная библиотека каркаса поддерживает удобство построения различных типов приложений на C#, позволяя легко строить Web-службы, другие виды компонентов, достаточно просто сохранять и получать информацию из базы данных и других хранилищ данных.

Язык C# поддерживает даже указатели и понятие «небезопасного» кода для тех случаев, когда прямой доступ к памяти имеет крайне важное значение [1, с. 4-8].

BEPUphysics – библиотека трехмерной физики от BEPU. Она быстрая и имеет множество возможностей вроде связей, сцен, статичных и быстроменяющихся сеток, непрерывной регистрации столкновений, настраиваемых правил столкновений, транспортных средств, простой многопоточности и так далее.

XNA Framework основывается на нативной реализации .NET Compact Framework 2.0 для разработки игр для Xbox 360 и .NET Framework 2.0 на Windows. Он включает обширный набор библиотек классов, специфичных для разработки игр, поддерживающий максимальное повторное использование кода на всех целевых платформах. Фреймворк выполняется на модификации Common Language Runtime, оптимизированной для игр, чтобы предоставить управляемую среду выполнения. Среда времени выполнения доступна для Windows XP, Windows Vista, Windows 7 и Xbox 360. Так как игры XNA пишутся для среды времени выполнения, они могут быть запущены на любой платформе, поддерживающей XNA Framework с минимальными изменениями или вообще без таковых. Игры, которые запускаются на фреймворке, технически могут быть написаны на любом .NET-совместимом языке, но официально поддерживается только язык программирования C# и среды быстрой разработки XNA Game Studio Express и все версии Visual Studio 2005.

## 2.2 Обоснование выбора среды разработки

Проект был реализован в среде быстрой разработки Visual Studio 2015.

Visual Studio – линейка продуктов компании Microsoft, включающих интегрированную среду разработки программного обеспечения и ряд других инструментальных средств. Данные продукты позволяют разрабатывать как консольные приложения, так и приложения с графическим интерфейсом, в том числе с поддержкой технологии Windows Forms, а также веб-сайты, веб-приложения, веб-службы как в родном, так и в управляемом кодах для всех платформ, поддерживаемых Windows, Windows Mobile, Windows CE, .NET Framework, Xbox, Windows Phone .NET Compact Framework и Silverlight.

Visual Studio включает в себя редактор исходного кода с поддержкой технологии IntelliSense и возможностью простейшего рефакторинга кода. Встроенный отладчик может работать как отладчик уровня исходного кода, так и отладчик машинного уровня.

Преимущества Visual Studio 2015:

* использование вычислительных мощностей локального компьютера и облака;
* простая реализация общих задач и индивидуальный подход;
* быстрое создание высококачественного кода;
* функция поддержки нескольких мониторов;
* возможность реализации идей и решений для широкого спектра платформ, включая Windows, Windows Server, веб-среду, облачную среду, Office и SharePoint.

## 2.3 Требования к конфигурации программного и аппаратного обеспечения

Данный проект разработан для ОС Windows 7. Достоинства:

1. Данную ОС устанавливают на большей массе современных ноутбуков, нетбуков и стационарных компьютеров, поэтому намного проще выполнить [восстановление windows-системы после вирусов](http://pcnotes.ru/soft/kak-vosstanovit-rabotu-noutbuka-v-otsutstvie-zashhity-ot-virusov/).
2. Высокая стабильность работы.
3. Интерфейс является крайне удобным. Интерфейс пользователя «семерки» в сравнении с Windows XP и Vista отличается яркостью красок и стилями оформления.
4. Windows 7 имеет высочайший уровень устойчивости к ошибочным и неосторожным действиям пользователей.
5. Перечень настроек Windows 7 весьма богат – начиная от графики и заканчивая правами доступа и настройкой безопасности.
6. Данная ОС совместима с современными распространенными программами.

Минимальные системные требования:

* ОС: Windows 7/8/10;
* процессор: Intel Core i3-3110M 2.40GHz;
* оперативная память: 2 ГБ (рекомендуется 4 ГБ).

# 3 Проектирование

## 3.1 Проектирование интерфейса

Разработать игру «Pusher». Реализовать механику подбора предметов и возможностью кинуть их.

При старте игры игрок должен появится в игровом мире. Перед ним должны находиться пять разных объектов. У каждого вида объектов должна быть своя масса, которая при броске должна быстрее замедлять объект и притягивать его к земле. При подходе к объекту должен появиться индикатор, который сигнализирует о том, что можно этот объект подобрать. При нажатии на правую кнопку мыши объект должен быть подобран и находиться в руках у игрока. Этот объект должен постоянно следовать за игроком так, как это было бы в реальном мире. При повторном нажатие на правую кнопку мыши объект должен будет упасть на землю так, как будто его положил человек. Если пользователь нажмет на левую кнопку мыши, то объект должен будет отброшен так, как будто его человек выбросит в ту сторону, в которую он смотрит. Так же разработать возможность ходить по игровому миру, прыгать, приседать и ложиться на землю.

## 3.2 Описание алгоритмов

Основным алгоритмом программы является «создания коллизий». При создание объекта необходимо создать низко-полигональную модель, по которой будут рассчитываться столкновения. Для этого текущую модель необходимо добавить в метод GetVerticesAndIndicesFromModel, который реализован в классе ModelDataExtractor, который возвращает вершины и индексы, необходимые для создания низко-полигональной модели, а после чего необходимо создать модель с помощью класса Entity. После того, как модель будет создана, необходимо добавить ее в общее пространство, реализованное через класс Space, в котором происходят обновления для расчета столкновений.

## 3.3 Описание справочной системы

Справочная система предназначена для получения пользователем максимально точной (релевантной) информации по интересующей его/её (и ограниченной базой статей) теме. Обычно выбор статьи происходит по иерархии разделов справки. Справочные системы часто комбинируются с поисковыми, где выборка релевантных статей определяется по заданным ключевым словам или (при полнотекстовом поиске) частью предложения.

Одним из примеров справочной системы является справка программного продукта: для платформы MS Windows это HTMLHelp, для Unix-подобных операционных системах это man.

Другой тип справочных – это вопросно-ответные системы, дающие один краткий ответ на поставленный вопрос. Примером может служить электронный телефонный справочник.

Также существуют телефонные справочно-сервисные службы, сотрудники которых дают абонентам ответы на различные вопросы и оказывают различные услуги.

# 4 Описание Программы

## 4.1 Логическая структура

Логическая структура проекта «Pusher» представлена классами:

1. Game1 представлен методами:
   * Initialize – инициализирует класс;
   * LoadContent – загрузка объектов в игру;
   * Update – обновление всех компонентов;
   * Draw – рисование всех компонентов.
2. Camera представлен методами:
   * ViewDirection – Направление камеры;
   * MoveForward – движение вперед;
   * Pitch – расчет наклона;
   * Yaw – расчет смещения.
3. Sky представлен методом Draw, который отвечает за рисования неба.
4. ModelBase представлен методами:
   * UpdateWorld – обновление мировой матрицы;
   * CreateMorphableEntity – Создание физической модели объекта.
5. ModelSimple представлен методами:
   * Update – обновление модели;
   * Draw – рисование модели.
6. SimpliesUpdatesAndDrawings представлен методом Add, добавляет ссылку объекта к общему обновлению и рисованию.
7. CameraControlScheme представлена методом Update, который осуществляет обновление класса Camera.
8. CharacterControllerInput представлен методами:
   * Update – обновление персонажа;
   * Activate – активация персонажа;
   * Deactivate – деактивация персонажа.
9. GrabEntity представлен методами:
   * Update – обновление захвата модели;
   * Draw – рисование индикатора захвата.
10. ModelDataExtractor представлен методами:
    * GetVerticesAndIndicesFromModel – получение вершин и индексов модели;
    * AddMesh – получение модели из вершин и индексов.
11. MotorizedGrabSpring представлен методами:
    * Setup – захват объекта;
    * Release – отпускание объекта;
    * Update – обновление захваченного объекта.

## 4.2 Физическая структура

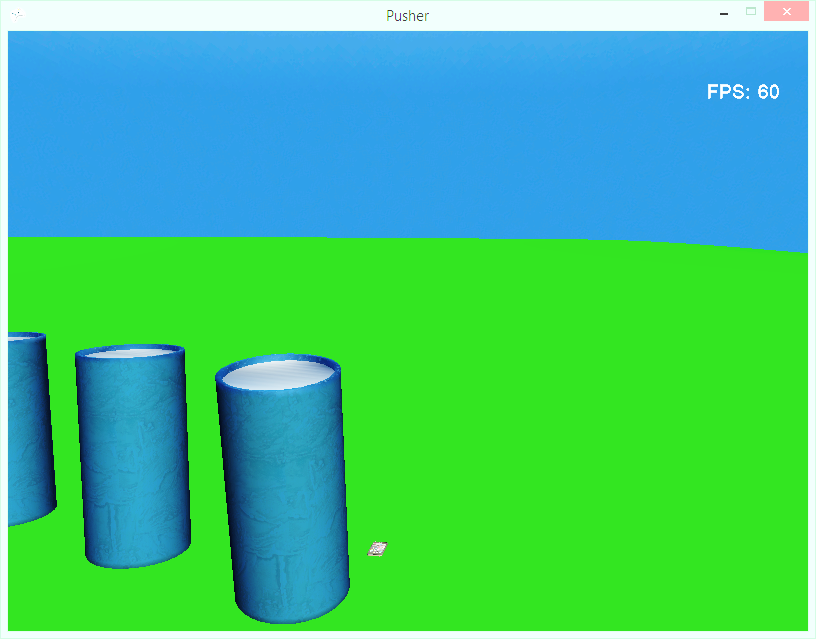
Физическая структура подразумевает описание рабочих файлов программы, к которым относятся в основном файлы с определением функций. Обычно такими файлами являются Unit и динамические библиотеки, подключаемые к программе для обеспечения работы программы.

Физическая структура курсового проекта представлена следующими модулями:

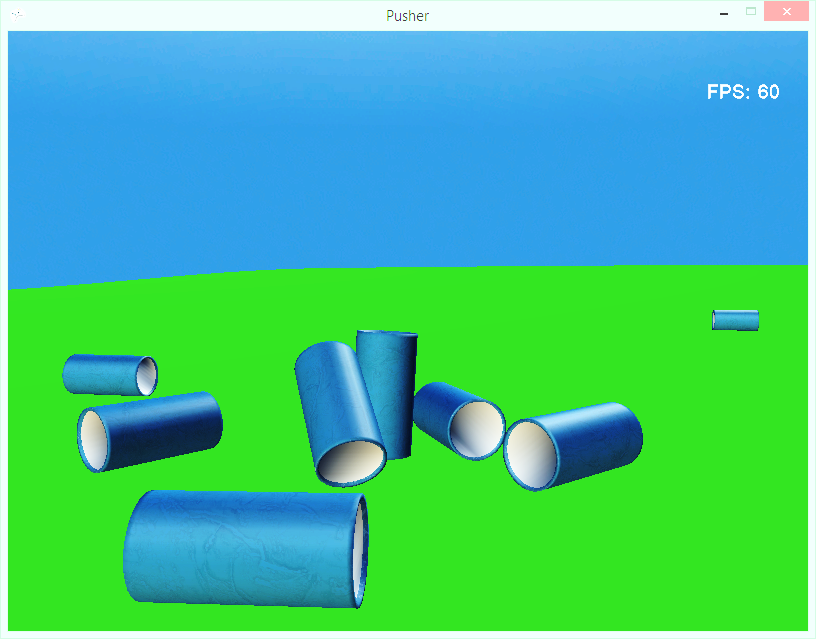
1. BEPUphysics – в этом модуле происходят все работы с физическими представлениями моделей. Используются классы Entity, MorableEntity, CharacterController, Space, Terrain, CollisiomRule.
2. BEPUutilites – в этом модуле происходят все работы для корректировки физический моделей. Исользуются классы Vector3, Matrix, Matrix3x3, MathHelper, Quaternion, Ray, RayHit.
3. BEPUphysicsDrawer – в этом модуле происходят все работы для отображения физических моделей. Используются классы ModelDrawer и TextDrawer.
4. ConversionHelper – в этом модуле происходят все работы для пересчета из XNA единиц в BEPU единицы. Используется класс MathConverter.

## 4.3 Описание интерфейса

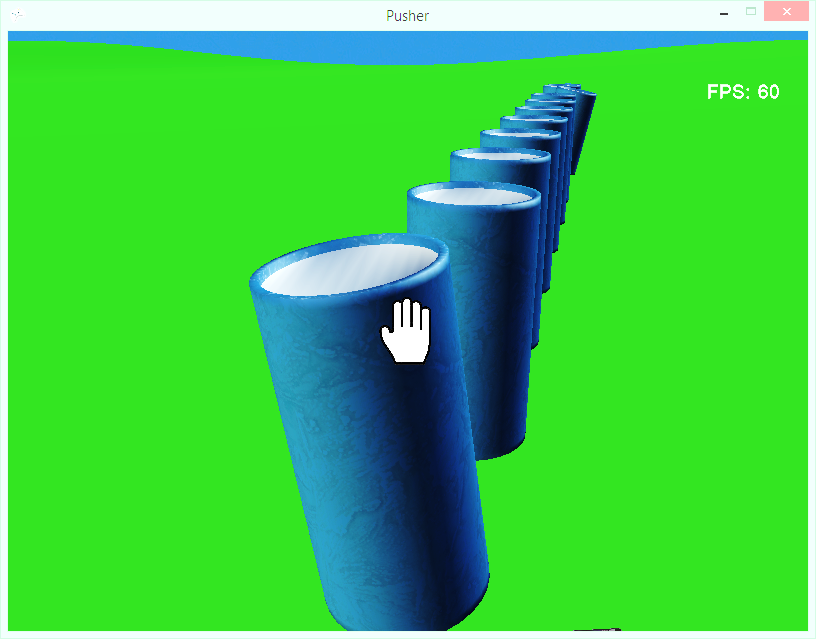
При запуске приложения игрок попадает на игровое поле, изображённая на рисунке 1.

Рисунок 1 – Запуск игры

При толкание моделей они будут падать как на рисунке 2.

Рисунок 2 – Толкание объектов

Если пользователь подойдет к какому-то объекту, который можно подобрать, то перед ним покажется индикатор, изображённый на рисунке 3.

Рисунок 3 – Индикатор подпора объект

Если при появление индикатора нажать на правую кнопку мыши, то объект подберется, как на рисунке 4.

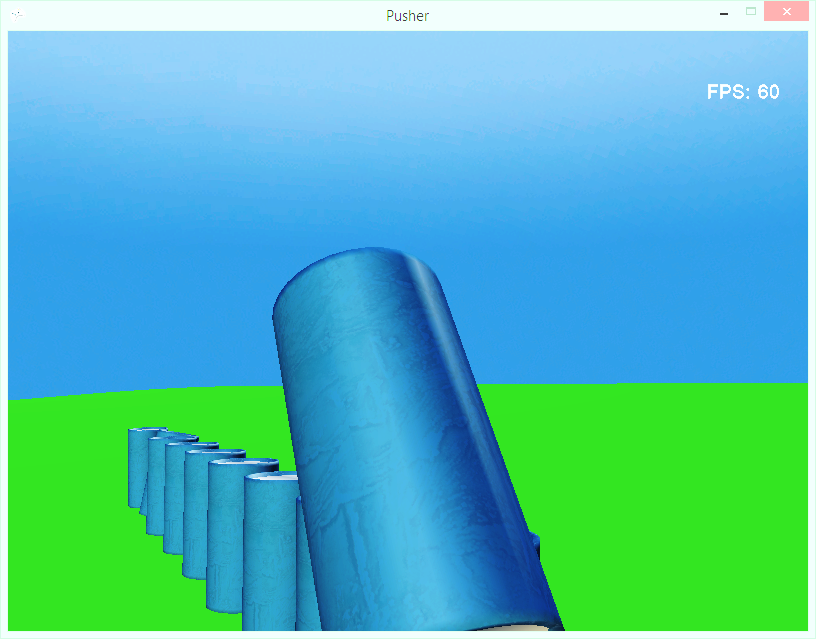


Рисунок 4 – Объект подобран

Если объект подобран, то его можно бросить, нажав на левую кнопку мыши, как на рисунке 5.

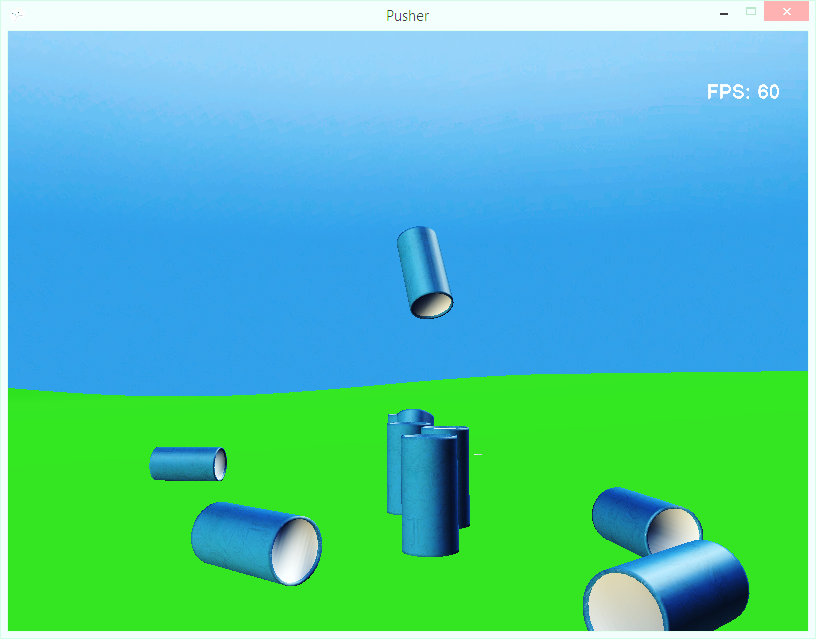


Рисунок 5 – Бросание модели

При нажатие на пробел персонаж подпрыгнет, как на рисунке 6.

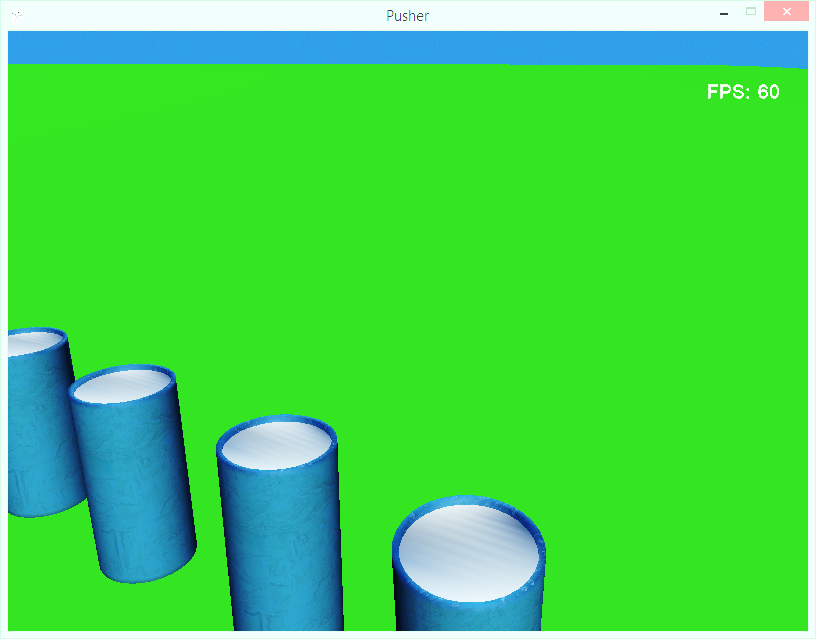


Рисунок 6 – Прыжок

# 5 Отладка и испытание программы

## 5.1 Тестовые примеры

Тестирование программного обеспечения – процесс исследования, испытания [программного продукта](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5), имеющий две различные цели:

* продемонстрировать разработчикам и заказчикам, что программа соответствует требованиям;
* выявить ситуации, в которых поведение программы является неправильным, нежелательным или не соответствующим спецификации.

Существующие на сегодняметоды тестирования программного обеспечения не позволяют однозначно и полностью выявить все дефекты и установить корректность функционирования анализируемой программы, поэтому все существующие методы тестирования действуют в рамках формального процесса проверки исследуемого или разрабатываемого программного обеспечения. Такой процесс формальной проверки может доказать, что дефекты отсутствуют с точки зрения используемого метода. Существует множество подходов к решению задачи тестирования и верификации программного обеспечения, но эффективное тестирование сложных программных продуктов – это процесс в высшей степени творческий, не сводящийся к следованию строгим и чётким процедурам или созданию таковых.

Для данного проекты будут проведены тесты:

1. Тест 1 – Подбор объектов.
2. Тест 2 – Бросание объектов.
3. Тест 3 – Опускание объектов на другие объекты.
4. Тест 4 – Бросание объекта в другой объект.
5. Тест 5 – Подбор объекта до появления индикатора.
6. Тест 6 – Попытка уйти за границы карты.

## 5.2 анализ полученных результатов

Тест 1 – Попробуем подойти к объекту и подобрать его, как на рисунке 7.

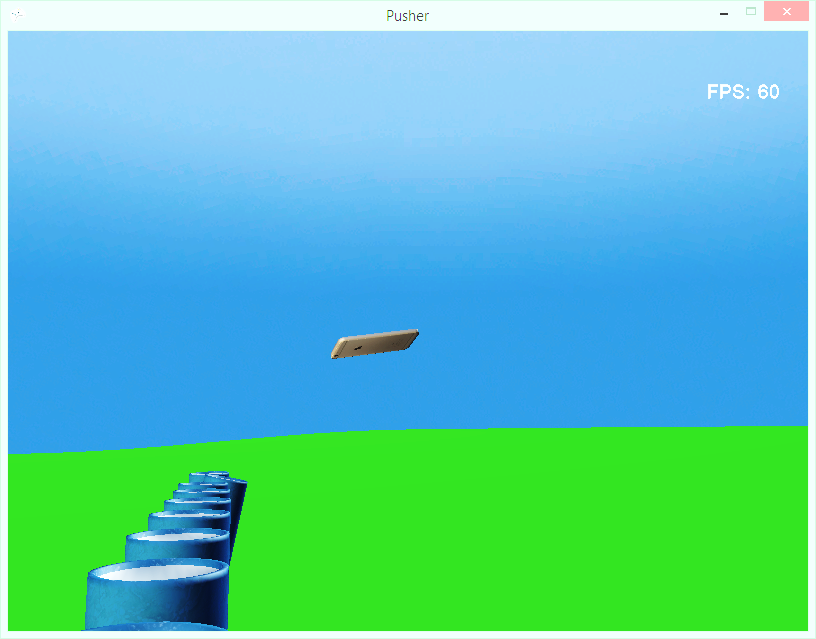


Рисунок 7 – Объект подобрался

Тест 2 – Попробуем подойти к объекту, подобрать его и брось, изображён на рисунке 8.

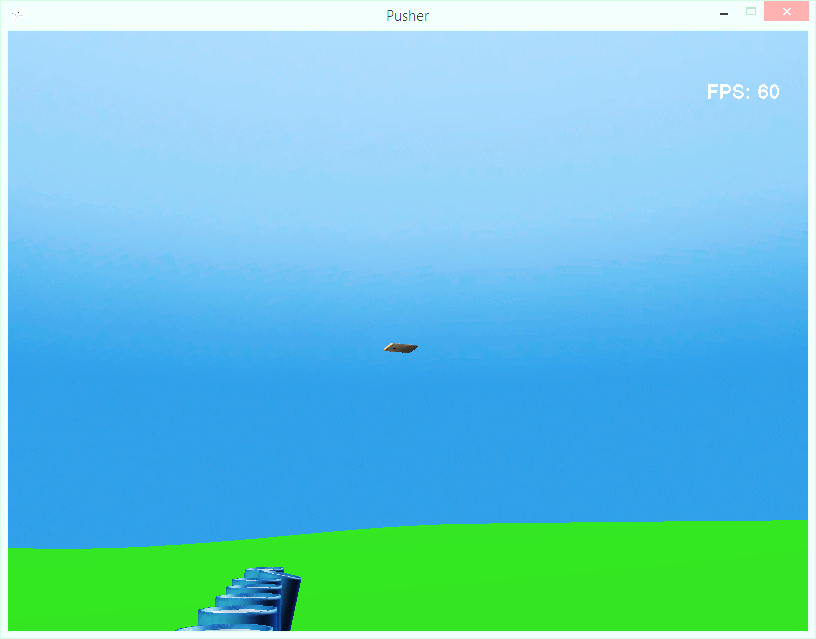


Рисунок 8 – Объект полетел

Тест 3 – Попробуем подойти к объекту, подобрать его и положить объект на другой объект, изображён на рисунке 9.



Рисунок 9 – Объект лежит на другом объекте

Тест 4 – Попробуем подойти к объекту, подобрать его и кинуть в другой объект, изображён на рисунке 10.

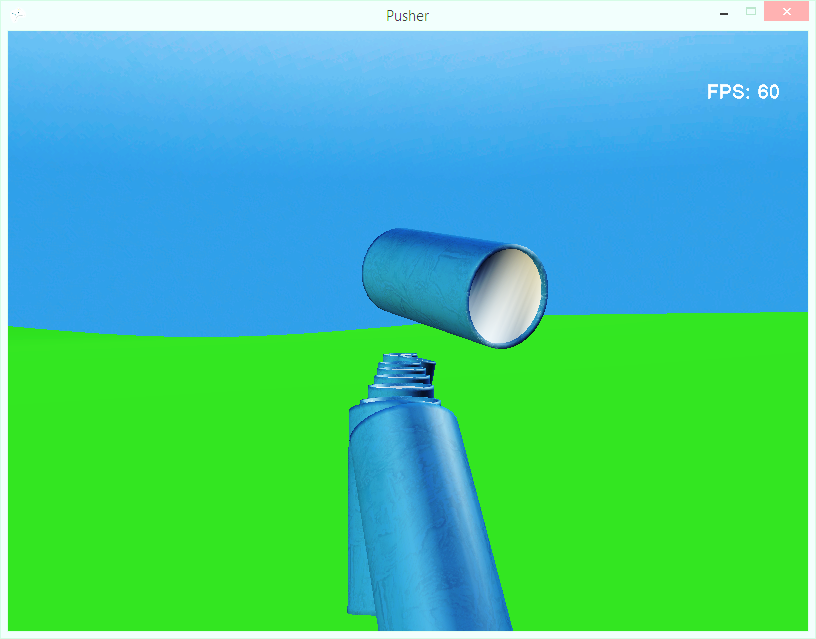


Рисунок 10 – Объект сбивает другие объекты при броске

Тест 5 – Попробуем подобрать объект до появления индикатора, изображён на рисунке 11.

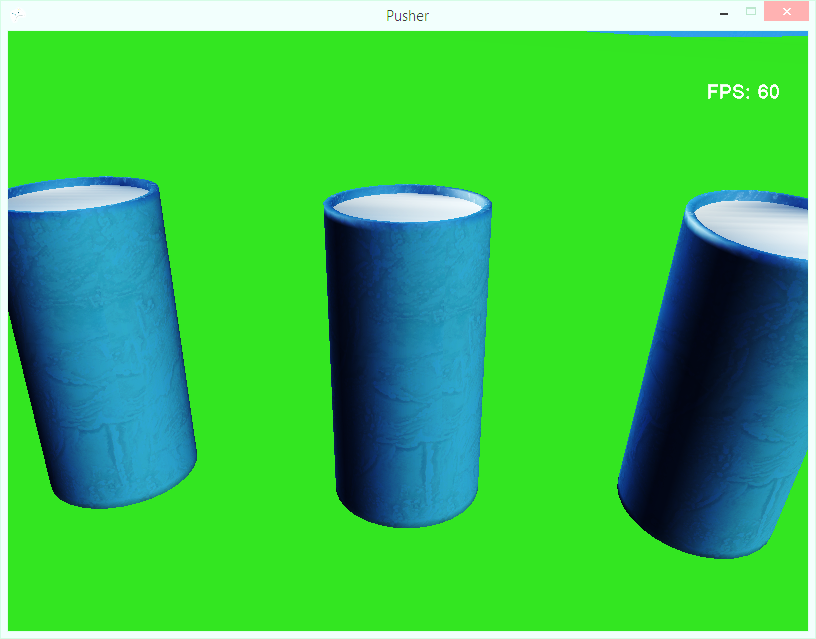


Рисунок 11 – Нельзя подобрать объект до появления индикатора

Тест 6 – Попробуем идти в одну сторону, пока не дойдем до границы карты, изображён на рисунке 12.

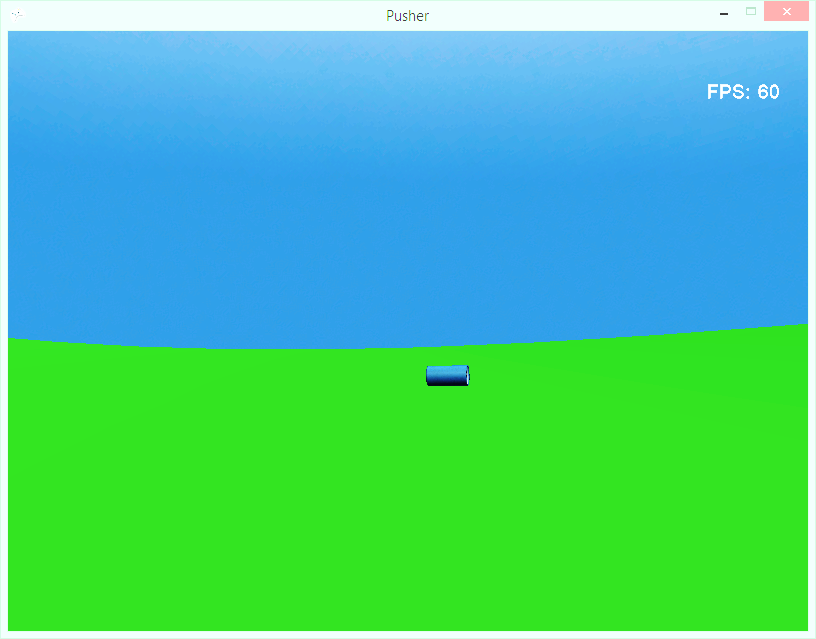


Рисунок 12 – До границы карты дойти не возможно

На всех тестах игра ведёт себя корректно:

1. Все объекты подбираются корректно.
2. Все объекты бросаются корректно.
3. Все объекты могут находится на других объектах.
4. При бросании объекта он будет сбивать все другие объекты, находящиеся на его пути до того, как он не остановиться.
5. Невозможно подобрать объект до появления индикатора.
6. Не возможно уйти за границы карты.

# 6 Описание применения программы

## 6.1 Назначение программы

Данное программное средство является игрой симулятором, которое предназначено для ознакомления с физикой, почти как в реальной жизни. Во время игры пользователь может увидеть, как взаимодействуют объекты друг с другом, и что будет, если бросить один объект в другой. Используя данное программное средство, пользователь имеет возможность перемещаться по игровому миру. Сделано это было для лучшего погружения в игру и для более лучшего анализа. Так же пользователь может подбирать предметы, что бы затем кинуть их в любую желаемую сторону, или просто положить его на землю или другой объект.

## 6.2 Условия использования

Для возможности пользования игрой у пользователя должен быть установлен на компьютере .Net Framework, а также XNA Framework.

Для того, что бы поиграть в игру пользователь должен:

* 1. Если игра не установлена – установить ее, запустив SetupPusher.exe.
  2. Если не установлен .Net Framework – установить его.
  3. Если не установлен XNA Framework – установить его.
  4. Запустить игру нажав на Pusher.exe.

# Заключение

В данном курсовом проекте была разработана игра «Pusher».

В проекте была реализована реалистичная физика поведения объектов с возможностью взаимодействия с ними, а так же перемещения по игровому миру.

Программа была разработана в среде разработки Visual Studio 2015, с использованием технологии XNA, а так же с открытым исходным кодом BEPUphysics для создания реалистичной физики поведения объектов.

Имеются возможности для дальнейшей модификации проекта, такие как улучшения интерфейса, добавления расширенных настроек, а так же добавления сюжета.

# Список использованных источников

1. Андрей Гарнаев Самоучитель Visual Studio .NET 2003. БХВ-Петербург.
2. Брайн Джонсон, Основы Microsoft Visual Studio.NET 2003. Русская Редакция,2003. – 402 с.
3. Карли Уотсон Visual С# 2008. Базовый курс,2008.
4. Культин, Н. Б. Основы программирования в Microsoft Visual C# 2010. – СПб.: БХВ-Петербург, 2011. – 368 с.
5. Либерти Джесс Освой C# самостоятельно за 21 день 2000. – 654с.
6. Медведев, В.И. Особенности объектно-ориентированного программирования на C# и Java. – 444с.
7. Николай Секунов Самоучитель C#. БХВ-Петербург,2001.
8. Послед, Б.С. Borland C# Builder 6. Разработка приложений с базами данных ООО «ДиаСофтЮП», 2003. – 320с.
9. Т.А. Павловская C#.Программирование на языке высокого уровня Питер,2009.
10. Microsoft Добро пожаловать в Windows 8.1 Корпорация Майкрософт (Microsoft Corp.), 2012.

# Приложение 1

**Листинг программы**

Class Game1

using System;

using Microsoft.Xna.Framework;

using Microsoft.Xna.Framework.Graphics;

using Microsoft.Xna.Framework.Input;

using BEPUphysics;

using BEPUphysicsDrawer.Models;

using Vector3 = BEPUutilities.Vector3;

using Matrix = BEPUutilities.Matrix;

using Vector3Xna = Microsoft.Xna.Framework.Vector3;

using MatrixXna = Microsoft.Xna.Framework.Matrix;

using BEPUphysicsDrawer.Font;

using BEPUphysicsDrawer.Lines;

using ConversionHelper;

using BEPUphysics.BroadPhaseEntries;

namespace Game

{

/// <summary>

/// This is the main type for your game

/// </summary>

public class Game1 : Microsoft.Xna.Framework.Game

{

#region Fields

private SpriteBatch spriteBatch;

private SpriteFont \_dataFont;

//FPS calculation variables

public bool displayUI = true;

private double FPSlastTime;

private double FPStotalSinceLast;

private double FPStoDisplay;

private int FPStotalFramesSinceLast;

private bool \_displayEntities = false;

//Models

private ModelSimple[] barrels;

private ModelSimple iphoneSix;

private Terrain Terrain;

private Sky Sky;

#endregion

#region Properties

public int ScreenBottom { get; private set; }

public int ScreenRight { get; private set; }

public MouseState MouseState { get; set; }

public MouseState PreviousMouseState { get; set; }

public KeyboardState KeyboardState { get; set; }

public KeyboardState PreviousKeyboardState { get; set; }

public GraphicsDeviceManager Graphics { get; set; }

public Space Space { get; set; }

public ModelDrawer ModelDrawer { get; set; }

public Camera Camera { get; private set; }

public CharacterControllerInput Character { get; private set; }

public GrabEntity GrabEntity { get; private set; }

public SpriteBatch UIDrawer { get; private set; }

public TextDrawer DataTextDrawer { get; private set; }

#endregion

public Game1()

{

Graphics = new GraphicsDeviceManager(this);

Content.RootDirectory = "Content";

//Graphics.PreferredBackBufferHeight = 600;

//Graphics.PreferredBackBufferWidth = 800;

Window.Title = "Pusher";

Graphics.PreferredBackBufferWidth = GraphicsAdapter.DefaultAdapter.CurrentDisplayMode.Width;

Graphics.PreferredBackBufferHeight = GraphicsAdapter.DefaultAdapter.CurrentDisplayMode.Height;

Graphics.IsFullScreen = true;

IsMouseVisible = false;

}

/// <summary>

/// Allows the game to perform any initialization it needs to before starting to run.

/// This is where it can query for any required services and load any non-graphic

/// related content. Calling base.Initialize will enumerate through any components

/// and initialize them as well.

/// </summary>

protected override void Initialize()

{

ScreenBottom = GraphicsDevice.Viewport.Bounds.Height;

ScreenRight = GraphicsDevice.Viewport.Bounds.Width;

Camera = new Camera(new Vector3(0, 1.5f, 0),

0,

0,

Matrix.CreatePerspectiveFieldOfViewRH(MathHelper.PiOver4,

ScreenRight / (float)ScreenBottom,

0.1f,

10000));

Space = new Space();

Space.ForceUpdater.Gravity = new Vector3(0, -9.81f, 0);

ModelDrawer = new InstancedModelDrawer(this);

Character = new CharacterControllerInput(Space, Camera, this);

Character.Activate();

int xLength = 50;

int zLength = 50;

float xSpacing = 8f;

float zSpacing = 8f;

var heights = new float[xLength, zLength];

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

{

for (int j = 0; j < zLength; j++)

{

float x = i - xLength / 2;

float z = j - zLength / 2;

heights[i, j] = 3 \* (float)Math.Sin(x \* z / 100f);

}

}

Terrain = new Terrain(heights, new BEPUutilities.AffineTransform(

new Vector3(xSpacing, 1, zSpacing),

BEPUutilities.Quaternion.Identity,

new Vector3(-xLength \* xSpacing / 2, -10, -zLength \* zSpacing / 2)), this);

Space.Add(Terrain.terrain);

base.Initialize();

}

/// <summary>

/// LoadContent will be called once per game and is the place to load

/// all of your content.

/// </summary>

protected override void LoadContent()

{

spriteBatch = new SpriteBatch(GraphicsDevice);

GrabEntity = new GrabEntity(Content.Load<Texture2D>("GrabHand"), this, Space);

\_dataFont = Content.Load<SpriteFont>("DataFont");

UIDrawer = new SpriteBatch(GraphicsDevice);

DataTextDrawer = new TextDrawer(UIDrawer, \_dataFont, Color.White);

Vector3 Angle = new Vector3(-90, 0, 0);

Sky = new Sky(Content.Load<Model>("sky"), this, Angle);

barrels = new ModelSimple[10];

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

{

barrels[i] = new ModelSimple(Content.Load<Model>("Barrel"),

Content.Load<Texture2D>("BarrelT"),

0.5f,

this,

Angle);

barrels[i].ModelLowPoly.Position = new Vector3(i, 1, 1);

}

iphoneSix = new ModelSimple(Content.Load<Model>("iphone6"),

Content.Load<Model>("iphone6LP"),

Content.Load<Texture2D>("iphone6T"),

0.3f,

this,

Angle);

iphoneSix.ModelLowPoly.Position = new Vector3(0, 6, 1);

}

/// <summary>

/// UnloadContent will be called once per game and is the place to unload

/// all content.

/// </summary>

protected override void UnloadContent()

{

// TODO: Unload any non ContentManager content here

}

#region Handle Input

/// <summary>

/// Return was key pressed.

/// </summary>

public bool WasKeyPressed(Keys key)

{

return KeyboardState.IsKeyDown(key) && PreviousKeyboardState.IsKeyUp(key);

}

/// <summary>

/// Return was mouse right button click.

/// </summary>

public bool WasMouseRightButtonClick()

{

return MouseState.RightButton == ButtonState.Pressed && PreviousMouseState.RightButton == ButtonState.Released;

}

/// <summary>

/// Return was mouse left button click.

/// </summary>

public bool WasMouseLeftButtonClick()

{

return MouseState.LeftButton == ButtonState.Pressed && PreviousMouseState.LeftButton == ButtonState.Released;

}

/// <summary>

/// Handles input for control the game.

/// </summary>

private void HandleInput(GameTime gameTime)

{

if (WasKeyPressed(Keys.Escape))

Exit();

if (WasKeyPressed(Keys.Y))

\_displayEntities = !\_displayEntities;

if (WasKeyPressed(Keys.Q))

ModelDrawer.IsWireframe = !ModelDrawer.IsWireframe;

}

#endregion

/// <summary>

/// Allows the game to run logic such as updating the world,

/// checking for collisions, gathering input, and playing audio.

/// </summary>

/// <param name="gameTime">Provides a snapshot of timing values.</param>

protected override void Update(GameTime gameTime)

{

ScreenBottom = GraphicsDevice.Viewport.Bounds.Height;

ScreenRight = GraphicsDevice.Viewport.Bounds.Width;

PreviousKeyboardState = KeyboardState;

KeyboardState = Keyboard.GetState();

PreviousMouseState = MouseState;

MouseState = Mouse.GetState();

Character.Update((float)gameTime.ElapsedGameTime.TotalSeconds,

PreviousKeyboardState,

KeyboardState);

if (!IsMouseVisible)

Mouse.SetPosition(ScreenRight / 2, ScreenBottom / 2);

HandleInput(gameTime);

SimpliesUpdatesAndDrawings.Update(gameTime);

Space.Update();

GrabEntity.Update(gameTime);

if (\_displayEntities)

ModelDrawer.Update();

Vector3 NewPosition = Character.CharacterController.Body.Position;

if (NewPosition.X >= 150 || NewPosition.X <= -150)

{

NewPosition.X = Math.Sign(NewPosition.Y) \* 150;

Character.CharacterController.Body.Position = NewPosition;

}

if (NewPosition.Z >= 150 || NewPosition.Z <= -150)

{

NewPosition.Z = Math.Sign(NewPosition.Z) \* 150;

Character.CharacterController.Body.Position = NewPosition;

}

base.Update(gameTime);

}

/// <summary>

/// This is called when the game should draw itself.

/// </summary>

/// <param name="gameTime">Provides a snapshot of timing values.</param>

protected override void Draw(GameTime gameTime)

{

GraphicsDevice.Clear(new Color(.41f, .41f, .45f, 1));

GraphicsDevice.BlendState = BlendState.Opaque;

GraphicsDevice.RasterizerState = RasterizerState.CullNone;

GraphicsDevice.DepthStencilState = DepthStencilState.Default;

Terrain.Draw(Camera.ViewMatrix, Camera.ProjectionMatrix);

if (\_displayEntities)

ModelDrawer.Draw(Camera.ViewMatrix, Camera.ProjectionMatrix);

SimpliesUpdatesAndDrawings.Draw(Graphics.GraphicsDevice);

Sky.Draw(Graphics.GraphicsDevice);

base.Draw(gameTime);

#region UI Drawing

UIDrawer.Begin();

if (displayUI)

{

FPStotalSinceLast += gameTime.ElapsedGameTime.TotalSeconds;

FPStotalFramesSinceLast++;

if (gameTime.TotalGameTime.TotalSeconds - FPSlastTime > 0.25 && gameTime.ElapsedGameTime.TotalSeconds > 0)

{

double avg = FPStotalSinceLast / FPStotalFramesSinceLast;

FPSlastTime = gameTime.TotalGameTime.TotalSeconds;

FPStoDisplay = Math.Round(1 / avg, 1);

FPStotalSinceLast = 0;

FPStotalFramesSinceLast = 0;

}

DataTextDrawer.Draw("FPS: ", FPStoDisplay, new Vector2(ScreenRight - 100, 50));

GrabEntity.Draw(Graphics.GraphicsDevice);

}

UIDrawer.End();

#endregion

}

}

}

GrabObject

using System;

using BEPUphysics;

using BEPUphysics.BroadPhaseEntries.MobileCollidables;

using Microsoft.Xna.Framework;

using Microsoft.Xna.Framework.Graphics;

using BEPUphysics.CollisionRuleManagement;

using BEPUphysics.BroadPhaseEntries;

using Vector3 = BEPUutilities.Vector3;

using Matrix = BEPUutilities.Matrix;

using Vector3Xna = Microsoft.Xna.Framework.Vector3;

using MatrixXna = Microsoft.Xna.Framework.Matrix;

namespace Game

{

public class GrabEntity

{

private Space Space;

private Game1 Game;

private MotorizedGrabSpring grabber;

private float grabDistance;

private Texture2D GrabHand;

public GrabEntity(Texture2D grabHand, Game1 game, Space space)

{

GrabHand = grabHand;

Space = space;

Game = game;

grabber = new MotorizedGrabSpring();

space.Add(grabber);

rayCastFilter = RayCastFilter;

}

Func<BroadPhaseEntry, bool> rayCastFilter;

bool RayCastFilter(BroadPhaseEntry entry)

{

return entry != Game.Character.CharacterController.Body.CollisionInformation && entry.CollisionRules.Personal <= CollisionRule.Normal;

}

public virtual void Draw(GraphicsDevice device)

{

RayCastResult raycastResult;

if (Space.RayCast(new BEPUutilities.Ray(Game.Camera.Position, Game.Camera.WorldMatrix.Forward), 2, rayCastFilter, out raycastResult))

{

var entityCollision = raycastResult.HitObject as EntityCollidable;

//If there's a valid ray hit, then grab the connected object!

if (!grabber.IsGrabbing && entityCollision != null && entityCollision.Entity.IsDynamic)

{

//Vector3 local = raycastResult.HitData.Location;

Vector2 position = new Vector2();

position.X = Game.ScreenRight / 2 - GrabHand.Width / 2;

position.Y = Game.ScreenBottom / 2 - GrabHand.Height / 2;

//position.X = (float)Math.Sqrt(local.X \* local.X + local.Z \* local.Z);

//position.Y = local.Y;

Game.UIDrawer.Draw(GrabHand, position, Color.White);

}

}

}

//The raycast filter limits the results retrieved from the Space.RayCast while grabbing.

public virtual void Update(GameTime gameTime)

{

if (grabber.IsGrabbing)

grabber.GoalPosition = Game.Camera.Position + Game.Camera.WorldMatrix.Forward \* grabDistance;

//Throw the grabbing entity

//if (Game.MouseState.LeftButton == Microsoft.Xna.Framework.Input.ButtonState.Pressed && grabber.IsGrabbing)

if (Game.WasMouseLeftButtonClick() && grabber.IsGrabbing)

{

CollisionRules.RemoveRule(grabber.Entity, Game.Character.CharacterController.Body);

grabber.Entity.ApplyImpulse(grabber.Entity.MotionState.Position,

Game.Camera.WorldMatrix.Forward \* (1 - grabber.Entity.Mass) \* 10);

grabber.Release();

return;

}

//Put the grabbing entity

//if (Game.MouseState.RightButton == Microsoft.Xna.Framework.Input.ButtonState.Released && grabber.IsGrabbing)

if (Game.WasMouseRightButtonClick() && grabber.IsGrabbing)

{

CollisionRules.RemoveRule(grabber.Entity, Game.Character.CharacterController.Body);

grabber.Release();

return;

}

//Grabbing the entity

//if (Game.MouseState.RightButton == Microsoft.Xna.Framework.Input.ButtonState.Pressed && !grabber.IsGrabbing && Game.MouseState.LeftButton == Microsoft.Xna.Framework.Input.ButtonState.Released)

if (Game.WasMouseRightButtonClick() && !grabber.IsGrabbing)

{

RayCastResult raycastResult;

if (Space.RayCast(new BEPUutilities.Ray(Game.Camera.Position, Game.Camera.WorldMatrix.Forward), 2, rayCastFilter, out raycastResult))

{

var entityCollision = raycastResult.HitObject as EntityCollidable;

//If there's a valid ray hit, then grab the connected object!

if (entityCollision != null && entityCollision.Entity.IsDynamic)

{

grabber.Setup(entityCollision.Entity, raycastResult.HitData.Location);

grabDistance = raycastResult.HitData.T;

CollisionRules.AddRule(grabber.Entity, Game.Character.CharacterController.Body, CollisionRule.NoBroadPhase);

}

}

return;

}

}

}

}

Class Camera

using System;

using BEPUutilities;

namespace Game

{

/// <summary>

/// Simple camera class for moving around the demos area.

/// </summary>

public class Camera

{

/// <summary>

/// Gets or sets the position of the camera.

/// </summary>

public Vector3 Position { get; set; }

/// <summary>

/// Gets or sets the projection matrix of the camera.

/// </summary>

public Matrix ProjectionMatrix { get; set; }

/// <summary>

/// Gets the view matrix of the camera.

/// </summary>

public Matrix ViewMatrix

{

get { return Matrix.CreateViewRH(Position, viewDirection, lockedUp); }

}

/// <summary>

/// Gets the world transformation of the camera.

/// </summary>

public Matrix WorldMatrix

{

get { return Matrix.CreateWorldRH(Position, viewDirection, lockedUp); }

}

private Vector3 viewDirection = Vector3.Forward;

/// <summary>

/// Gets or sets the view direction of the camera.

/// </summary>

public Vector3 ViewDirection

{

get { return viewDirection; }

set

{

float lengthSquared = value.LengthSquared();

if (lengthSquared > Toolbox.Epsilon)

{

Vector3.Divide(ref value, (float)Math.Sqrt(lengthSquared), out value);

//Validate the input. A temporary violation of the maximum pitch is permitted as it will be fixed as the user looks around.

//However, we cannot allow a view direction parallel to the locked up direction.

float dot;

Vector3.Dot(ref value, ref lockedUp, out dot);

if (Math.Abs(dot) > 1 - Toolbox.BigEpsilon)

{

//The view direction must not be aligned with the locked up direction.

//Silently fail without changing the view direction.

return;

}

viewDirection = value;

}

}

}

private float maximumPitch = MathHelper.PiOver2 \* 0.99f;

/// <summary>

/// Gets or sets how far the camera can look up or down in radians.

/// </summary>

public float MaximumPitch

{

get { return maximumPitch; }

set

{

if (value < 0)

throw new ArgumentException("Maximum pitch corresponds to pitch magnitude; must be positive.");

if (value >= MathHelper.PiOver2)

throw new ArgumentException("Maximum pitch must be less than Pi/2.");

maximumPitch = value;

}

}

private Vector3 lockedUp = Vector3.Up;

/// <summary>

/// Gets or sets the current locked up vector of the camera.

/// </summary>

public Vector3 LockedUp

{

get { return lockedUp; }

set

{

var oldUp = lockedUp;

float lengthSquared = value.LengthSquared();

if (lengthSquared > Toolbox.Epsilon)

{

Vector3.Divide(ref value, (float)Math.Sqrt(lengthSquared), out lockedUp);

//Move the view direction with the transform. This helps guarantee that the view direction won't end up aligned with the up vector.

Quaternion rotation;

Quaternion.GetQuaternionBetweenNormalizedVectors(ref oldUp, ref lockedUp, out rotation);

Quaternion.Transform(ref viewDirection, ref rotation, out viewDirection);

}

//If the new up vector was a near-zero vector, silently fail without changing the up vector.

}

}

/// <summary>

/// Creates a camera.

/// </summary>

/// <param name="position">Initial position of the camera.</param>

/// <param name="pitch">Initial pitch angle of the camera.</param>

/// <param name="yaw">Initial yaw value of the camera.</param>

/// <param name="projectionMatrix">Projection matrix used.</param>

public Camera(Vector3 position, float pitch, float yaw, Matrix projectionMatrix)

{

Position = position;

Yaw(yaw);

Pitch(pitch);

ProjectionMatrix = projectionMatrix;

}

/// <summary>

/// Moves the camera forward.

/// </summary>

/// <param name="distance">Distance to move.</param>

public void MoveForward(float distance)

{

Position += WorldMatrix.Forward \* distance;

}

/// <summary>

/// Moves the camera to the right.

/// </summary>

/// <param name="distance">Distance to move.</param>

public void MoveRight(float distance)

{

Position += WorldMatrix.Right \* distance;

}

/// <summary>

/// Moves the camera up.

/// </summary>

/// <param name="distance">Distance to move.</param>

public void MoveUp(float distance)

{

Position += new Vector3(0, distance, 0);

}

/// <summary>

/// Rotates the camera around its locked up vector.

/// </summary>

/// <param name="radians">Amount to rotate.</param>

public void Yaw(float radians)

{

//Rotate around the up vector.

Matrix3x3 rotation;

Matrix3x3.CreateFromAxisAngle(ref lockedUp, radians, out rotation);

Matrix3x3.Transform(ref viewDirection, ref rotation, out viewDirection);

//Avoid drift by renormalizing.

viewDirection.Normalize();

}

/// <summary>

/// Rotates the view direction up or down relative to the locked up vector.

/// </summary>

/// <param name="radians">Amount to rotate.</param>

public void Pitch(float radians)

{

//Do not allow the new view direction to violate the maximum pitch.

float dot;

Vector3.Dot(ref viewDirection, ref lockedUp, out dot);

//While this could be rephrased in terms of dot products alone, converting to actual angles can be more intuitive.

//Consider +Pi/2 to be up, and -Pi/2 to be down.

float currentPitch = (float)Math.Acos(MathHelper.Clamp(-dot, -1, 1)) - MathHelper.PiOver2;

//Compute our new pitch by clamping the current + change.

float newPitch = MathHelper.Clamp(currentPitch + radians, -maximumPitch, maximumPitch);

float allowedChange = newPitch - currentPitch;

//Compute and apply the rotation.

Vector3 pitchAxis;

Vector3.Cross(ref viewDirection, ref lockedUp, out pitchAxis);

//This is guaranteed safe by all interaction points stopping viewDirection from being aligned with lockedUp.

pitchAxis.Normalize();

Matrix3x3 rotation;

Matrix3x3.CreateFromAxisAngle(ref pitchAxis, allowedChange, out rotation);

Matrix3x3.Transform(ref viewDirection, ref rotation, out viewDirection);

//Avoid drift by renormalizing.

viewDirection.Normalize();

}

}

}

Class CameraControlScheme

namespace Game

{

/// <summary>

/// Superclass of implementations which control the behavior of a camera.

/// </summary>

public abstract class CameraControlScheme

{

/// <summary>

/// Gets the game associated with the camera.

/// </summary>

public Game1 Game { get; private set; }

/// <summary>

/// Gets the camera controlled by this control scheme.

/// </summary>

public Camera Camera { get; private set; }

protected CameraControlScheme(Camera camera, Game1 game)

{

Camera = camera;

Game = game;

}

/// <summary>

/// Updates the camera state according to the control scheme.

/// </summary>

/// <param name="dt">Time elapsed since previous frame.</param>

public virtual void Update(float dt)

{

//Only turn if the mouse is controlled by the game.

if (!Game.IsMouseVisible)

{

Camera.Yaw((Game.ScreenRight / 2 - Game.MouseState.X) \* dt \* 0.05f);

Camera.Pitch((Game.ScreenBottom / 2 - Game.MouseState.Y) \* dt \* 0.05f);

}

}

}

}

CharacterControllerInput

using BEPUphysics;

using BEPUphysics.Character;

using Microsoft.Xna.Framework.Input;

using BEPUutilities;

namespace Game

{

/// <summary>

/// Handles input and movement of a Character in the game.

/// Acts as a simple 'front end' for the bookkeeping and math of the Character controller.

/// </summary>

public class CharacterControllerInput

{

/// <summary>

/// Gets the camera to use for input.

/// </summary>

public Camera Camera { get; private set; }

/// <summary>

/// Physics representation of the Character.

/// </summary>

public CharacterController CharacterController;

/// <summary>

/// Gets the camera control scheme used by the Character input.

/// </summary>

public CharacterCameraControlScheme CameraControlScheme { get; private set; }

/// <summary>

/// Gets whether the Character controller's input management is being used.

/// </summary>

public bool IsActive { get; private set; }

/// <summary>

/// Owning space of the Character.

/// </summary>

public Space Space { get; private set; }

/// <summary>

/// Constructs the Character and internal physics Character controller.

/// </summary>

/// <param name="owningSpace">Space to add the Character to.</param>

/// <param name="camera">Camera to attach to the Character.</param>

/// <param name="game">The running game.</param>

public CharacterControllerInput(Space owningSpace, Camera camera, Game1 game)

{

CharacterController = new CharacterController();

Camera = camera;

CameraControlScheme = new CharacterCameraControlScheme(CharacterController, camera, game);

Space = owningSpace;

}

/// <summary>

/// Gives the Character control over the Camera and movement input.

/// </summary>

public void Activate()

{

if (!IsActive)

{

IsActive = true;

Space.Add(CharacterController);

//Offset the Character start position from the camera to make sure the camera doesn't shift upward discontinuously.

CharacterController.Body.Position = Camera.Position - new Vector3(0, CameraControlScheme.StandingCameraOffset, 0);

}

}

/// <summary>

/// Returns input control to the Camera.

/// </summary>

public void Deactivate()

{

if (IsActive)

{

IsActive = false;

Space.Remove(CharacterController);

}

}

/// <summary>

/// Handles the input and movement of the Character.

/// </summary>

/// <param name="dt">Time since last frame in simulation seconds.</param>

/// <param name="previousKeyboardInput">The last frame's keyboard state.</param>

/// <param name="keyboardInput">The current frame's keyboard state.</param>

/// <param name="previousGamePadInput">The last frame's gamepad state.</param>

/// <param name="gamePadInput">The current frame's keyboard state.</param>

public void Update(float dt, KeyboardState previousKeyboardInput, KeyboardState keyboardInput) //, GamePadState previousGamePadInput, GamePadState gamePadInput

{

if (IsActive)

{

//Note that the Character controller's update method is not called here; this is because it is handled within its owning space.

//This method's job is simply to tell the Character to move around.

CameraControlScheme.Update(dt);

Vector2 totalMovement = Vector2.Zero;

//Collect the movement impulses.

if (keyboardInput.IsKeyDown(Keys.W))

{

totalMovement += new Vector2(0, 1);

}

if (keyboardInput.IsKeyDown(Keys.S))

{

totalMovement += new Vector2(0, -1);

}

if (keyboardInput.IsKeyDown(Keys.A))

{

totalMovement += new Vector2(-1, 0);

}

if (keyboardInput.IsKeyDown(Keys.D))

{

totalMovement += new Vector2(1, 0);

}

if (totalMovement == Vector2.Zero)

CharacterController.HorizontalMotionConstraint.MovementDirection = Vector2.Zero;

else

CharacterController.HorizontalMotionConstraint.MovementDirection = Vector2.Normalize(totalMovement);

if (keyboardInput.IsKeyDown(Keys.X))

CharacterController.StanceManager.DesiredStance = Stance.Prone;

else if (keyboardInput.IsKeyDown(Keys.LeftControl))

CharacterController.StanceManager.DesiredStance = Stance.Crouching;

else

CharacterController.StanceManager.DesiredStance = Stance.Standing;

//Jumping

if (previousKeyboardInput.IsKeyUp(Keys.Space) && keyboardInput.IsKeyDown(Keys.Space))

{

CharacterController.Jump();

}

CharacterController.ViewDirection = Camera.WorldMatrix.Forward;

}

}

}

}

Class

using BEPUphysics.CollisionShapes;

using BEPUphysics.Entities;

using BEPUphysics.PositionUpdating;

using Microsoft.Xna.Framework;

using Microsoft.Xna.Framework.Graphics;

using ConversionHelper;

using Vector3 = BEPUutilities.Vector3;

using Matrix = BEPUutilities.Matrix;

using Vector3Xna = Microsoft.Xna.Framework.Vector3;

using MatrixXna = Microsoft.Xna.Framework.Matrix;

namespace Game

{

public class ModelBase : DrawableGameComponent

{

#region Fields

private Vector3 \_centerLowPoly;

private Model \_modelMain;

private MorphableEntity \_modelLowPoly;

private Texture2D \_textureModel;

private Vector3 \_angle;

private Vector3 \_transpose;

private Matrix \_world;

#endregion

#region Properties

public Model ModelMain

{

get { return \_modelMain; }

set { \_modelMain = value; }

}

public MorphableEntity ModelLowPoly

{

get { return \_modelLowPoly; }

set { \_modelLowPoly = value; }

}

public Texture2D TextureModel

{

get { return \_textureModel; }

set { \_textureModel = value; }

}

public Vector3 Angle

{

get { return \_angle; }

set { \_angle = value; }

}

public Vector3 Transpose

{

get { return \_transpose; }

set { \_transpose = value; }

}

public Matrix World

{

get { return \_world; }

set { \_world = value; }

}

public Matrix Transform { get; set; }

#endregion

public ModelBase(Game1 Game) : base(Game)

{

\_angle = Vector3.Zero;

\_transpose = Vector3.Zero;

}

public ModelBase(Game1 Game, Vector3 angle) : base(Game)

{

\_angle = angle;

\_transpose = Vector3.Zero;

}

public virtual void UpdateWorld()

{

\_world = \_modelLowPoly.WorldTransform;

}

public virtual void Update(GameTime gameTime)

{

}

public virtual void Draw(GraphicsDevice device)

{

}

protected void CreateMorphableEntity(Model modelLowPoly, float mass)

{

int[] indices;

Vector3[] vertices;

ModelDataExtractor.GetVerticesAndIndicesFromModel(modelLowPoly, out vertices, out indices);

MatrixXna matrixRotation = MatrixXna.CreateFromYawPitchRoll(MathHelper.ToRadians(\_angle.Y),

MathHelper.ToRadians(\_angle.X),

MathHelper.ToRadians(\_angle.Z));

BEPUutilities.AffineTransform localTransform = new BEPUutilities.AffineTransform(Vector3.Zero);

MobileMeshShape mms = new MobileMeshShape(vertices,

indices,

localTransform,

MobileMeshSolidity.Solid,

out \_centerLowPoly);

\_modelLowPoly = mass == 0 ? new MorphableEntity(mms) : new MorphableEntity(mms, mass);

if (mass == 0)

{

\_modelLowPoly.Material.KineticFriction = 20f;

\_modelLowPoly.Material.StaticFriction = 20f;

}

\_modelLowPoly.PositionUpdateMode = PositionUpdateMode.Continuous;

\_modelLowPoly.CollisionInformation.LocalPosition = \_centerLowPoly;

\_modelLowPoly.Orientation = BEPUutilities.Quaternion.CreateFromRotationMatrix(MathConverter.Convert(matrixRotation));

\_modelLowPoly.Position = Vector3.Zero;

UpdateWorld();

}

}

}

Class ModelSimple

using Microsoft.Xna.Framework;

using Microsoft.Xna.Framework.Graphics;

using ConversionHelper;

using Vector3 = BEPUutilities.Vector3;

using Matrix = BEPUutilities.Matrix;

using Vector3Xna = Microsoft.Xna.Framework.Vector3;

using MatrixXna = Microsoft.Xna.Framework.Matrix;

namespace Game

{

class ModelSimple : ModelBase

{

public ModelSimple(Model currentModel, Texture2D textureModel, float mass, Game1 Game) : base(Game)

{

ModelMain = currentModel;

TextureModel = textureModel;

CreateMorphableEntity(currentModel, mass);

Game.Space.Add(ModelLowPoly);

Game.ModelDrawer.Add(ModelLowPoly);

SimpliesUpdatesAndDrawings.Add(this);

}

public ModelSimple(Model currentModel, Model modelLowPoly, Texture2D textureModel, float mass, Game1 Game) : base(Game)

{

ModelMain = currentModel;

TextureModel = textureModel;

CreateMorphableEntity(modelLowPoly, mass);

Game.Space.Add(ModelLowPoly);

Game.ModelDrawer.Add(ModelLowPoly);

SimpliesUpdatesAndDrawings.Add(this);

}

public ModelSimple(Model currentModel, Texture2D textureModel, float mass, Game1 Game, Vector3 angle) : base(Game, angle)

{

ModelMain = currentModel;

TextureModel = textureModel;

CreateMorphableEntity(currentModel, mass);

Game.Space.Add(ModelLowPoly);

Game.ModelDrawer.Add(ModelLowPoly);

SimpliesUpdatesAndDrawings.Add(this);

}

public ModelSimple(Model currentModel, Model modelLowPoly, Texture2D textureModel, float mass, Game1 Game, Vector3 angle) : base(Game, angle)

{

ModelMain = currentModel;

TextureModel = textureModel;

CreateMorphableEntity(modelLowPoly, mass);

Game.Space.Add(ModelLowPoly);

Game.ModelDrawer.Add(ModelLowPoly);

SimpliesUpdatesAndDrawings.Add(this);

}

public override void Update(GameTime gameTime)

{

Transpose = ModelLowPoly.Position;

}

public override void Draw(GraphicsDevice device)

{

MatrixXna worldMatrix = MathConverter.Convert(ModelLowPoly.WorldTransform);

foreach (ModelMesh mesh in ModelMain.Meshes)

{

foreach (BasicEffect effect in mesh.Effects)

{

effect.View = MathConverter.Convert((Game as Game1).Camera.ViewMatrix);

effect.Projection = MathConverter.Convert((Game as Game1).Camera.ProjectionMatrix);

effect.World = worldMatrix;

effect.Texture = TextureModel;

effect.EnableDefaultLighting();

effect.SpecularColor = new Vector3Xna(0.25f);

effect.SpecularPower = 16;

}

mesh.Draw();

}

}

}

}

Class ModelDataExtractor

using System.Collections.Generic;

using ConversionHelper;

using Microsoft.Xna.Framework.Graphics;

using Microsoft.Xna.Framework;

namespace Game

{

/// <summary>

/// Contains helper methods for extracting vertices and indices from XNA models.

/// </summary>

public static class ModelDataExtractor

{

/// <summary>

/// Gets an array of vertices and indices from the provided model.

/// </summary>

/// <param name="collisionModel">Model to use for the collision shape.</param>

/// <param name="vertices">Compiled set of vertices from the model.</param>

/// <param name="indices">Compiled set of indices from the model.</param>

public static void GetVerticesAndIndicesFromModel(Model collisionModel, out BEPUutilities.Vector3[] vertices, out int[] indices)

{

Vector3[] tempVertices;

GetVerticesAndIndicesFromModel(collisionModel, out tempVertices, out indices);

vertices = MathConverter.Convert(tempVertices);

}

/// <summary>

/// Gets an array of vertices and indices from the provided model.

/// </summary>

/// <param name="collisionModel">Model to use for the collision shape.</param>

/// <param name="vertices">Compiled set of vertices from the model.</param>

/// <param name="indices">Compiled set of indices from the model.</param>

public static void GetVerticesAndIndicesFromModel(Model collisionModel, out Vector3[] vertices, out int[] indices)

{

var verticesList = new List<Vector3>();

var indicesList = new List<int>();

var transforms = new Matrix[collisionModel.Bones.Count];

collisionModel.CopyAbsoluteBoneTransformsTo(transforms);

Matrix transform;

foreach (ModelMesh mesh in collisionModel.Meshes)

{

if (mesh.ParentBone != null)

transform = transforms[mesh.ParentBone.Index];

else

transform = Matrix.Identity;

AddMesh(mesh, transform, verticesList, indicesList);

}

vertices = verticesList.ToArray();

indices = indicesList.ToArray();

}

/// <summary>

/// Adds a mesh's vertices and indices to the given lists.

/// </summary>

/// <param name="collisionModelMesh">Model to use for the collision shape.</param>

/// <param name="transform">Transform to apply to the mesh.</param>

/// <param name="vertices">List to receive vertices from the mesh.</param>

/// <param name="indices">List to receive indices from the mesh.</param>

public static void AddMesh(ModelMesh collisionModelMesh, Matrix transform, List<Vector3> vertices, IList<int> indices)

{

foreach (ModelMeshPart meshPart in collisionModelMesh.MeshParts)

{

int startIndex = vertices.Count;

var meshPartVertices = new Vector3[meshPart.NumVertices];

//Grab position data from the mesh part.

int stride = meshPart.VertexBuffer.VertexDeclaration.VertexStride;

meshPart.VertexBuffer.GetData(

meshPart.VertexOffset \* stride,

meshPartVertices,

0,

meshPart.NumVertices,

stride);

//Transform it so its vertices are located in the model's Space as opposed to mesh part Space.

Vector3.Transform(meshPartVertices, ref transform, meshPartVertices);

vertices.AddRange(meshPartVertices);

if (meshPart.IndexBuffer.IndexElementSize == IndexElementSize.ThirtyTwoBits)

{

var meshIndices = new int[meshPart.PrimitiveCount \* 3];

meshPart.IndexBuffer.GetData(meshPart.StartIndex \* 4, meshIndices, 0, meshPart.PrimitiveCount \* 3);

for (int k = 0; k < meshIndices.Length; k++)

{

indices.Add(startIndex + meshIndices[k]);

}

}

else

{

var meshIndices = new ushort[meshPart.PrimitiveCount \* 3];

meshPart.IndexBuffer.GetData(meshPart.StartIndex \* 2, meshIndices, 0, meshPart.PrimitiveCount \* 3);

for (int k = 0; k < meshIndices.Length; k++)

{

indices.Add(startIndex + meshIndices[k]);

}

}

}

}

}

}

Class MotorizedGrabSpring

using BEPUphysics;

using BEPUphysics.Constraints.SingleEntity;

using BEPUphysics.Constraints.TwoEntity.Motors;

using BEPUphysics.Entities;

using BEPUphysics.UpdateableSystems;

using BEPUutilities;

namespace Game

{

/// <summary>

/// Grabs an entity at a specified location and applies corrective impulses to keep the grabbed location near the goal location.

/// </summary>

public class MotorizedGrabSpring : Updateable, IEndOfFrameUpdateable

{

SingleEntityLinearMotor linearMotor;

SingleEntityAngularMotor angularMotor;

/// <summary>

/// Constructs a grab constraint.

/// </summary>

public MotorizedGrabSpring()

{

//Note that when the motor is created using the empty constructor,

//it starts deactivated. This prevents explosions from attempting

//to update it without being configured.

linearMotor = new SingleEntityLinearMotor();

angularMotor = new SingleEntityAngularMotor();

linearMotor.Settings.Mode = MotorMode.Servomechanism;

//The stiffness, damping, and maximum force could be assigned during setup if the motor

//needs to behave similarly for entities of varying masses. When using a fixed configuration,

//the grabspring will behave weakly when trying to move extremely heavy objects, while staying

//very responsive for sufficiently light objects.

IsUpdating = false;

}

/// <summary>

/// Gets the grabbed entity.

/// </summary>

public Entity Entity

{

get

{

return linearMotor.Entity;

}

private set

{

if (linearMotor.Entity != value) //Don't bother changing the entity if it is the same.

{

linearMotor.Entity = value;

angularMotor.Entity = value;

//The motors can only be on while the entity isn't null.

if (value != null)

{

linearMotor.IsActive = true;

angularMotor.IsActive = true;

IsUpdating = true;

}

else

{

linearMotor.IsActive = false;

angularMotor.IsActive = false;

IsUpdating = false;

}

}

}

}

/// <summary>

/// Gets the location that the entity will be pulled towards.

/// </summary>

public Vector3 GoalPosition

{

get

{

return linearMotor.Settings.Servo.Goal;

}

set

{

linearMotor.Settings.Servo.Goal = value;

}

}

/// <summary>

/// Gets the offset from the entity to the grabbed location in its local space.

/// </summary>

public Vector3 LocalOffset

{

get

{

return linearMotor.LocalPoint;

}

private set

{

linearMotor.LocalPoint = value;

}

}

/// <summary>

/// Gets the last updated position of the grab location on the surface of the entity.

/// </summary>

public Vector3 GrabbedPosition { get; private set; }

/// <summary>

/// Gets whether or not the grabber is currently grabbing something.

/// </summary>

public bool IsGrabbing

{

get

{

return Entity != null;

}

}

/// <summary>

/// Reinitializes the grabbing constraint with new information.

/// </summary>

/// <param name="e">Entity to grab.</param>

/// <param name="grabLocation">Location on the entity being grabbed in world space.</param>

public void Setup(Entity e, Vector3 grabLocation)

{

//You can configure the stiffness and damping of the corrective springs like so.

//For this example, the motors will be just be the nearly rigid default.

linearMotor.Settings.Servo.SpringSettings.Stiffness = 60000 \* e.Mass;

linearMotor.Settings.Servo.SpringSettings.Damping = 9000 \* e.Mass;

angularMotor.Settings.VelocityMotor.Softness = .1f / e.Mass;

//An unlimited motor will gladly push the entity through other objects.

//Putting a limit on the strength of the motor will prevent it from doing so.

linearMotor.Settings.MaximumForce = 1000 \* e.Mass;

Entity = e;

LocalOffset = Quaternion.Transform(grabLocation - e.Position, Quaternion.Conjugate(e.Orientation));

angularMotor.Settings.Servo.Goal = e.Orientation;

GoalPosition = grabLocation;

}

/// <summary>

/// Releases the entity being held by the grab spring.

/// </summary>

public void Release()

{

Entity = null;

}

/// <summary>

/// Updates the grab constraint's grab position after the end of a frame.

/// </summary>

/// <param name="dt">Time since last frame in simulation seconds.</param>

void IEndOfFrameUpdateable.Update(float dt)

{

//Since the grabbed position is usually examined graphically,

//it's good to use the interpolated positions in case the

//engine is using internal time stepping and interpolation.

GrabbedPosition = Matrix3x3.Transform(LocalOffset, Entity.BufferedStates.InterpolatedStates.OrientationMatrix) + Entity.BufferedStates.InterpolatedStates.Position;

}

public override void OnAdditionToSpace(Space newSpace)

{

newSpace.Add(linearMotor);

newSpace.Add(angularMotor);

}

public override void OnRemovalFromSpace(Space oldSpace)

{

oldSpace.Remove(linearMotor);

oldSpace.Remove(angularMotor);

}

}

}

IAuthProvider

namespace DrugShop.Infrastructure

{

public interface IAuthProvider

{

bool Authenticate(string username, string password);

}

}

Class Sky

using ConversionHelper;

using Microsoft.Xna.Framework;

using Microsoft.Xna.Framework.Graphics;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Game

{

public class Sky

{

private Model SkySphere;

private Game1 Game;

private Vector3 Correction;

private Vector3 Angle;

public Sky(Model SkySphere, Game1 Game, BEPUutilities.Vector3 Angle)

{

this.SkySphere = SkySphere;

this.Game = Game;

this.Angle = MathConverter.Convert(Angle);

Correction = new Vector3(0, -1.5f, 0);

}

public void Draw(GraphicsDevice device)

{

foreach (ModelMesh mesh in SkySphere.Meshes)

{

foreach (BasicEffect effect in mesh.Effects)

{

effect.World = Matrix.CreateFromYawPitchRoll(MathHelper.ToRadians(Angle.Y),

MathHelper.ToRadians(Angle.X),

MathHelper.ToRadians(Angle.Z)) \*

Matrix.CreateTranslation(MathConverter.Convert(Game.Character.CharacterController.Body.Position +

MathConverter.Convert(Correction)));

effect.View = MathConverter.Convert(Game.Camera.ViewMatrix);

effect.Projection = MathConverter.Convert(Game.Camera.ProjectionMatrix);

}

mesh.Draw();

}

}

}

}