МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ ДНР

ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Кафедра ПИ им. Л.П. Фельдмана

Лабораторная работа №4

Тема: Создание самодокументирующегося кода

Курс: Профессиональная практика программной инженерии

Выполнил

Сырых А.С.

Проверил

Ищенко А. П.

Донецк 2022

Задание к лабораторной работе

1. Во всех файлах с исходным кодом добавить докблоки ко всем классам, методам, свойствам и функциям.
2. Закоммитить изменения в репозиторий.
3. Установить один из XML-редакторов (по желанию).
4. Оформить краткое описание разработанного продукта (не менее пяти абзацев).
5. Рассмотреть с примерами кода не менее трех сценариев использования вашего приложения.
6. Проверить корректность созданного DocBook файла с помощью средств редактора или одного из онлайн валидаторов, например, mashuosoft.com/docbook/validator

Ход работы

Т.к. данный проект разрабатывался на языке C#, то для XML-документации у него предусмотрен свой инструмент и теги форматирования.

В дополнение к комментариям в стиле C, в C# имеется очень искусное средство – способность генерировать документацию в формате [XML](https://professorweb.ru/my/glossariy/glossy/files/103.php#x1) на основе специальных комментариев. Это однострочные комментарии, начинающиеся с трех слешей (///) вместо двух. В таких комментариях можно размещать XML-дескрипторы, содержащие документацию по типам и членам типов, используемым в коде.

XML-дескрипторы, распознаваемые компилятором, перечислены в следующей таблице:

|  |  |
| --- | --- |
| XML-дескрипторы для комментариев | |
| Дескриптор | Описание |
| <example> | Помечает пример кода |
| <exception> | Документирует класс исключения (синтаксис проверяется компилятором) |
| <include> | Включает комментарии из другого файла документации (синтаксис проверяется компилятором) |
| <list> | Вставляет список в документацию |
| <param> | Помечает параметр метода (синтаксис проверяется компилятором) |
| <paramref> | Указывает, что слово является параметром метода (синтаксис проверяется компилятором) |
| <permission> | Документирует доступ к члену (синтаксис проверяется компилятором) |
| <remarks> | Добавляет описание члена |
| <returns> | Документирует возвращаемое методом значение |
| <see> | Представляет перекрестную ссылку на другой параметр (синтаксис проверяется компилятором) |
| <seealso> | Представляет раздел "see also" ("смотреть также") в описании (синтаксис проверяется компилятором) |
| <summary> | Представляет краткий итог о типе или члене |
| <value> | Описывает свойство |

Ниже представлены примеры XML-комментариев в разработанных классах на примере Code\_editor и Сamera:

namespace VR-Engine

{

///<summary>

/// Класс Code\_editor

/// интерфейс для редактирования кода

/// родительский класс Form

///</summary>

public partial class Code\_editor : Form

{

///<summary>

/// Конструктор класса Code\_editor

/// инициализация графических компонентов

///</summary>

public Code\_editor()

{

InitializeComponent();

}

///<summary>

/// свойство code

/// объект класса Pattern\_code

/// быстрое применение шаблонов

///</summary>

Pattern\_code code = new Pattern\_code();

///<summary>

/// свойство interpreter

/// объект класса Interpreter

/// интерпретатор написанного кода

///</summary>

Interpreter = new Interpreter();

///<summary>

/// свойство expert\_System

/// объект класса Expert\_system

/// экспертная система по выявлению потребностей

///</summary>

Expert\_system = new Expert\_system();

}

}

namespace VR-Engine

{

///<summary>

/// Класс Camera

/// реализация камеры

/// основной объект в сцене

///</summary>

public class Camera : INotifyPropertyChanged

{

///<summary>

/// свойство \_position

/// значение типа класса Vector3

/// расположения камеры

///</summary>

private Vector3 \_position;

///<summary>

/// свойство \_observerPoint

/// объект класса Vector3

/// вектор расположения точки наблюдения

///</summary>

private Vector3 \_observerPoint;

///<summary>

/// свойство \_focus

/// значение типа double

/// фокус камеры

///</summary>

private double \_focus;

///<summary>

/// Метод Focus типа double

/// получение и отправка текущего

/// значения фокуса камеры

///</summary>

public double Focus

{

get

{

return \_focus;

}

set

{

\_focus = value;

OnCoordinateChanged(nameof(Focus));

}

}

///<summary>

/// Метод Position типа Vector3

/// получение и отправка текущего

/// значения позиции камеры

///</summary>

public Vector3 Position

{

get {

return \_position;

}

set

{

\_position = value;

OnCoordinateChanged(nameof(Position));

}

}

///<summary>

/// Метод ObserverPoint типа Vector3

/// получение и отправка текущего

/// значения точки наблюдения

///</summary>

public Vector3 ObserverPoint

{

get

{

return \_observerPoint;

}

set

{

\_observerPoint = value;

OnCoordinateChanged(nameof(ObserverPoint));

}

}

public event PropertyChangedEventHandler PropertyChanged;

protected void OnCoordinateChanged(string propertyName)

{

OnCoordinateChanged(new PropertyChangedEventArgs(propertyName));

}

protected void OnCoordinateChanged(PropertyChangedEventArgs e)

{

PropertyChanged?.Invoke(this, e);

}

public event PositionChanged OnPositionChangedEvent;

public event ObserverPointChanged OnObserverPointChangedEvent;

public delegate void PositionChanged();

public delegate void ObserverPointChanged();

///<summary>

/// Конструктор класса Camera

/// Создание объекта класса Camera

/// с другим местоположением

///</summary>

///<param name="x">Координата x</param>

///<param name="y">Координата y</param>

///<param name="z">Координата z</param>

public Camera(double x, double y, double z)

{

Position = new Vector3();

ObserverPoint = new Vector3();

Position.X = x;

Position.Y = y;

Position.Z = z;

ObserverPoint.X = 0;

ObserverPoint.Y = 0;

ObserverPoint.Z = 0;

Position.PropertyChanged += CameraPositionChanged;

Position.PropertyChanged += CameraObserverPointChanged;

}

///<summary>

/// Конструктор класса Camera

/// Создание объекта класса Camera

/// с другой позицией камеры и точкой наблюдения

///</summary>

///<param name="position">Новая позиция камеры</param>

///<param name="observerPoint">Новая точка наблюдения</param>

public Camera(Vector3 position, Vector3 observerPoint)

{

Position = new Vector3(position);

ObserverPoint = new Vector3(observerPoint);

Position.PropertyChanged += CameraPositionChanged;

Position.PropertyChanged += CameraObserverPointChanged;

}

private void CameraPositionChanged(object sender, PropertyChangedEventArgs e)

{

OnPositionChangedEvent();

}

private void CameraObserverPointChanged(object sender, PropertyChangedEventArgs e)

{

OnObserverPointChangedEvent();

}

///<summary>

/// Метод Rotate

/// Поворот камеры на определенный угол

///</summary>

///<param name="doubleX">Угловой коэффициент по оси Х</param>

///<param name="doubleY">Угловой коэффициент по оси Y</param>

public void Rotate(double angleX, double angleY)

{

if (ObserverPoint.X == 0 && ObserverPoint.Y == 0 && ObserverPoint.Z == 0)

{

Vector3 CameraVector = Position - ObserverPoint;

double r = CameraVector.Length();

Position = new Vector3(r \* Math.Sin(angleY) \* Math.Sin(angleX),

r \* Math.Cos(angleY), -r \* Math.Sin(angleY) \* Math.Cos(angleX));

}

else

{

Vector3 CameraVector = Position - ObserverPoint;

double r = 8;

Position = new Vector3(r \* Math.Sin(angleY) \* Math.Sin(angleX),

r \* Math.Cos(angleY), -r \* Math.Sin(angleY) \* Math.Cos(angleX));

}

}

///<summary>

/// Метод GoCloser

/// Приближение камеры

///</summary>

///<param name="value">Значение, на которое приближается камера</param>

public void GoCloser(double value)

{

Vector3 NewView = ViewVector() \* (1 - value);

Position = new Vector3(NewView.X + ObserverPoint.X, NewView.Y + ObserverPoint.Y, NewView.Z + ObserverPoint.Z);

}

///<summary>

/// Метод GoFather

/// Отдаление камеры

///</summary>

///<param name="value">Значение, на которое отдаляется камера</param>

public void GoFarther(double value)

{

Vector3 NewView = ViewVector() \* (1 + value);

Position = new Vector3(NewView.X + ObserverPoint.X, NewView.Y + ObserverPoint.Y, NewView.Z + ObserverPoint.Z);

}

///<summary>

/// Метод ViewVector типа Vector3

/// Получение наблюдаемого вектора

///</summary>

public Vector3 ViewVector()

{

return Position - ObserverPoint;

}

}

}

Остальные основные классы описаны в аналогичной манере.

Компилятор C# может извлекать XML-элементы из специальных комментариев и использовать их для генерации файлов XML. Чтобы заставить компилятор сгенерировать XML-документацию для сборки, необходимо в свойствах проекта поставить галочку на выходные данные XML-файл документации и выбрать его месторасположение.

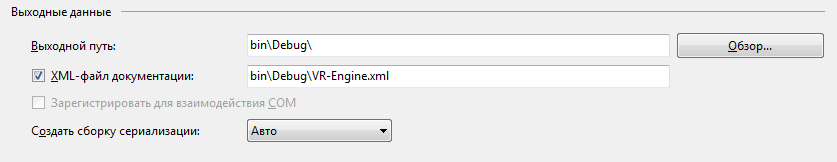


Рисунок 1 – Добавление XML-документации при сборке

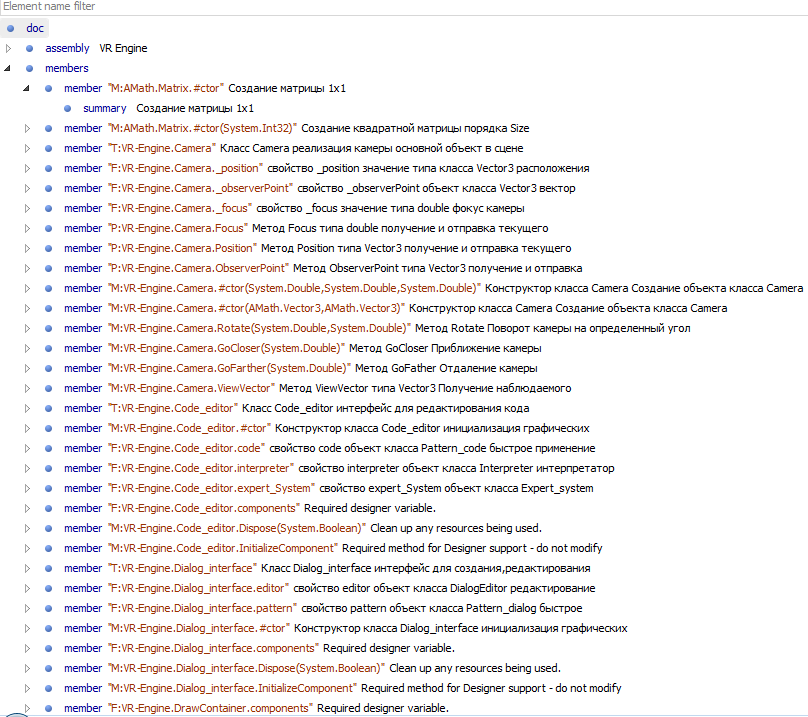


Рисунок 2 – Дерево XML-документации



Рисунок 3 – XML-документация в Oxygen Editor

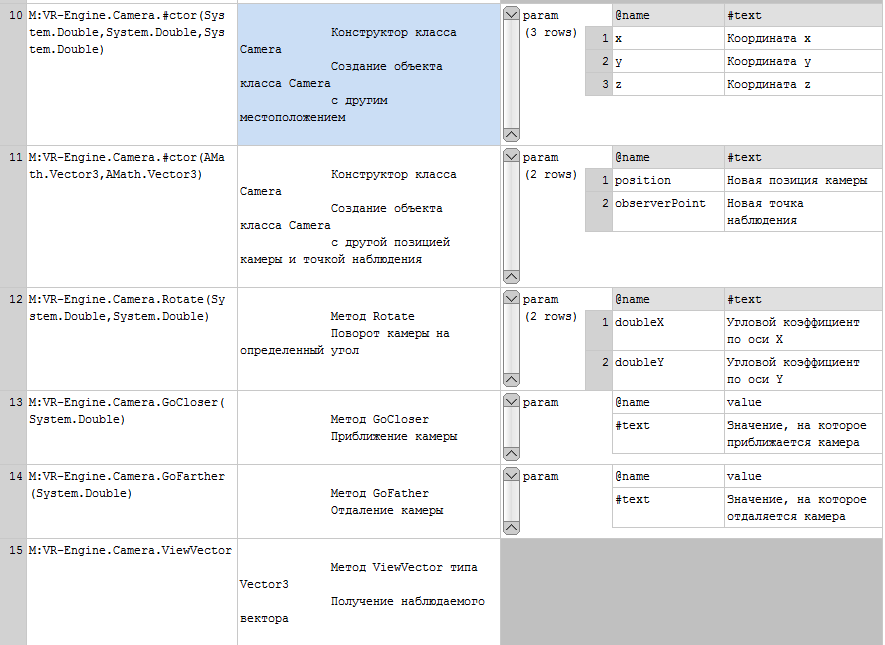


Рисунок 4 – XML-документация в виде таблицы



Рисунок 5 – Отображение DocBook (часть 1)

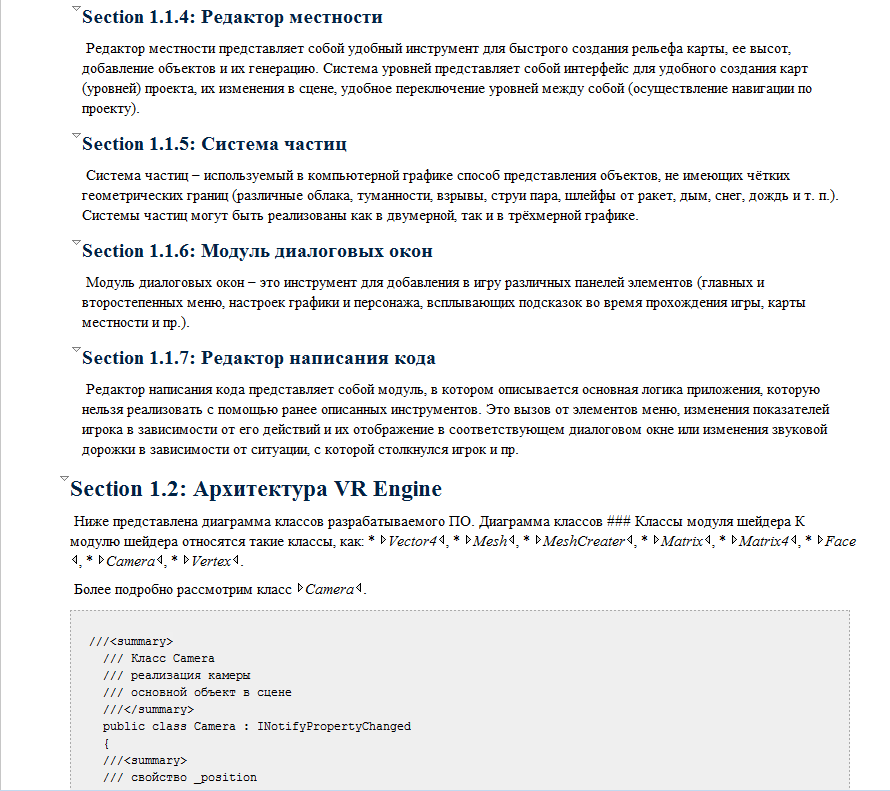
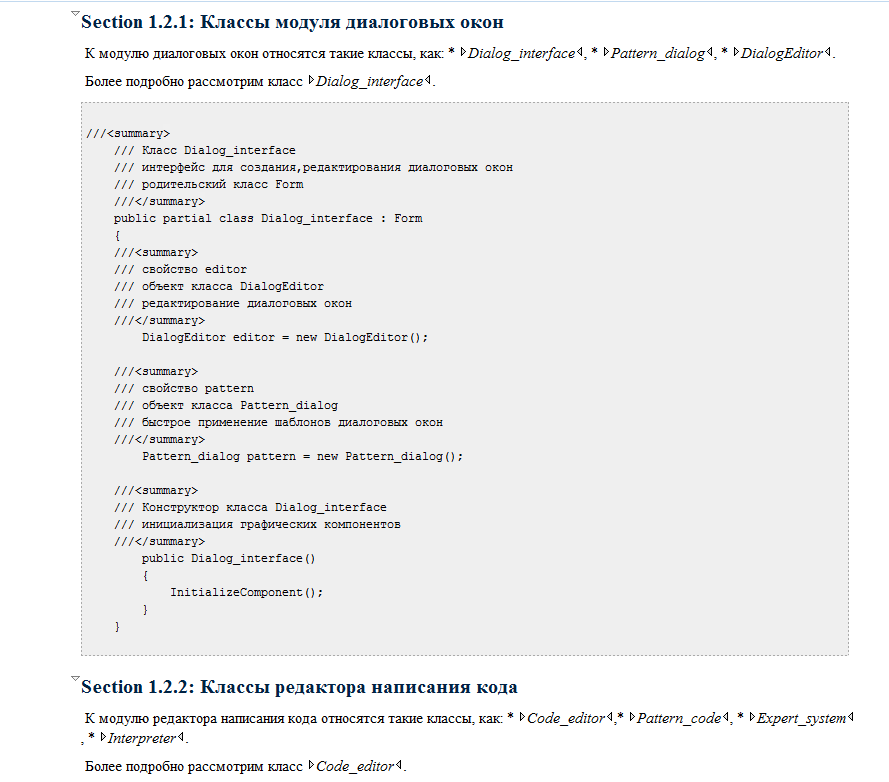
 

Рисунок 6 – Отображение DocBook (часть 2)

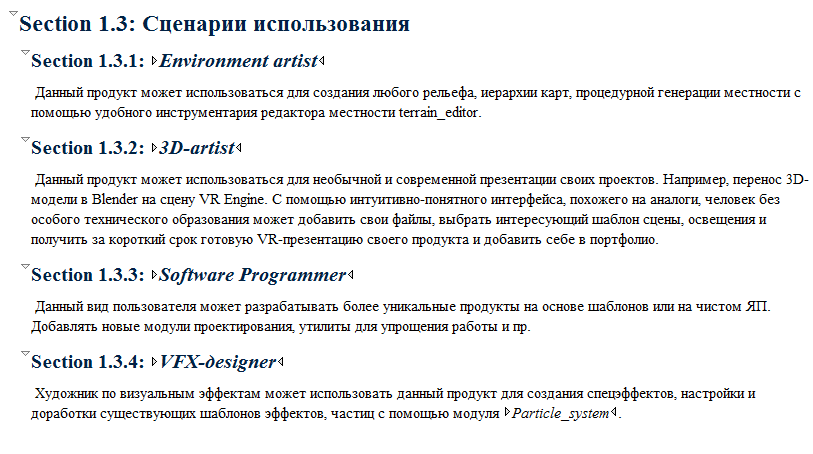


Рисунок 7 – Отображение

<?xml version="1.0" encoding="utf-8" ?>  
<!DOCTYPE article>  
<article  
 xmlns="http://docbook.org/ns/docbook" version="5.0"  
 xmlns:xlink="http://www.w3.org/1999/xlink" >  
 <info>  
 <title></title>  
 </info>  
 <section xml:id="vr-engine">  
 <title>VR-Engine</title>  
 <para>  
 <emphasis role="strong">VR Engine</emphasis> – это программный  
 комплекс, который упрощает разработку игр, предоставляя вам набор  
 необходимых для разработки инструментов. Обобщённо говоря, данный  
 продукт ответственен за организацию и поведение игровых объектов,  
 а также за их отображение на экране. Ваша же задача – выбрать, как  
 они будут выглядеть и как себя вести. Для этого движок предоставит  
 вам возможность создавать и удалять объекты, задавать их  
 параметры, добавлять логику и управлять ресурсами. Данный игровой  
 движок разработан с целью упрощения создания ПО и игр для VR/AR.  
 </para>  
 <section xml:id="модули-которые-предоставляет-vr-engine">  
 <title>Модули, которые предоставляет VR Engine:</title>  
 <itemizedlist spacing="compact">  
 <listitem>  
 <para>  
 Система частиц;  
 </para>  
 </listitem>  
 <listitem>  
 <para>  
 Редактор местности;  
 </para>  
 </listitem>  
 <listitem>  
 <para>  
 Модуль редактирования звуковых дорожек и вызывающих их  
 событий;  
 </para>  
 </listitem>  
 <listitem>  
 <para>  
 Модуль отображения и взаимодействия со сценой в реальном  
 времени;  
 </para>  
 </listitem>  
 <listitem>  
 <para>  
 Модуль шейдера;  
 </para>  
 </listitem>  
 <listitem>  
 <para>  
 Модуль диалоговых окон;  
 </para>  
 </listitem>  
 <listitem>  
 <para>  
 Редактор написания кода.  
 </para>  
 </listitem>  
 </itemizedlist>  
 <section xml:id="модуль-шейдера">  
 <title>Модуль шейдера</title>  
 <para>  
 Программы, работающие с трёхмерной графикой и видео (игры,  
 GIS, CAD, CAM и др.), используют шейдеры для определения  
 параметров геометрических объектов или изображения, для  
 изменения изображения (для создания эффектов сдвига,  
 отражения, преломления, затемнения с учётом заданных  
 параметров поглощения и рассеяния света, для наложения текстур  
 на геометрические объекты и др.).  
 </para>  
 </section>  
 <section xml:id="модуль-отображения-и-взаимодействия-со-сценой">  
 <title>Модуль отображения и взаимодействия со сценой</title>  
 <para>  
 Модуль отображения и взаимодействия со сценой является  
 интерфейсом пользователя и через взаимодействие с ним нажатием  
 горячих клавиш и кнопок мыши можно вызвать методы модуля  
 шейдера, которые перерисуют изображение на сцене. Также на нее  
 можно добавлять объекты, изменять, перемещать и прочее.  
 </para>  
 </section>  
 <section xml:id="модуль-воспроизведения-звука">  
 <title>Модуль воспроизведения звука</title>  
 <para>  
 Модуль воспроизведения звука представляет собой  
 пользовательский интерфейс для добавления звуковых дорожек в  
 создаваемый проект, их изменения, удаления и пр.  
 </para>  
 </section>  
 <section xml:id="редактор-местности">  
 <title>Редактор местности</title>  
 <para>  
 Редактор местности представляет собой удобный инструмент для  
 быстрого создания рельефа карты, ее высот, добавление объектов  
 и их генерацию. Система уровней представляет собой интерфейс  
 для удобного создания карт (уровней) проекта, их изменения в  
 сцене, удобное переключение уровней между собой (осуществление  
 навигации по проекту).  
 </para>  
 </section>  
 <section xml:id="система-частиц">  
 <title>Система частиц</title>  
 <para>  
 Система частиц – используемый в компьютерной графике способ  
 представления объектов, не имеющих чётких геометрических  
 границ (различные облака, туманности, взрывы, струи пара,  
 шлейфы от ракет, дым, снег, дождь и т. п.). Системы частиц  
 могут быть реализованы как в двумерной, так и в трёхмерной  
 графике.  
 </para>  
 </section>  
 <section xml:id="модуль-диалоговых-окон">  
 <title>Модуль диалоговых окон</title>  
 <para>  
 Модуль диалоговых окон – это инструмент для добавления в игру  
 различных панелей элементов (главных и второстепенных меню,  
 настроек графики и персонажа, всплывающих подсказок во время  
 прохождения игры, карты местности и пр.).  
 </para>  
 </section>  
 <section xml:id="редактор-написания-кода">  
 <title>Редактор написания кода</title>  
 <para>  
 Редактор написания кода представляет собой модуль, в котором  
 описывается основная логика приложения, которую нельзя  
 реализовать с помощью ранее описанных инструментов. Это вызов  
 от элементов меню, изменения показателей игрока в зависимости  
 от его действий и их отображение в соответствующем диалоговом  
 окне или изменения звуковой дорожки в зависимости от ситуации,  
 с которой столкнулся игрок и пр.  
 </para>  
 </section>  
 </section>  
 <section xml:id="архитектура-vr-engine">  
 <title>Архитектура VR Engine</title>  
 <para>  
 Ниже представлена диаграмма классов разрабатываемого ПО.  
 Диаграмма классов ### Классы модуля шейдера К модулю шейдера  
 относятся такие классы, как: \* <emphasis>Vector4</emphasis>, \*  
 <emphasis>Mesh</emphasis>, \* <emphasis>MeshCreater</emphasis>, \*  
 <emphasis>Matrix</emphasis>, \* <emphasis>Matrix4</emphasis>, \*  
 <emphasis>Face</emphasis>, \* <emphasis>Camera</emphasis>, \*  
 <emphasis>Vertex</emphasis>.  
 </para>  
 <para>  
 Более подробно рассмотрим класс <emphasis>Camera</emphasis>.  
 </para>  
 <programlisting>  
 ///&lt;summary&gt;  
 /// Класс Camera  
 /// реализация камеры  
 /// основной объект в сцене  
 ///&lt;/summary&gt;  
 public class Camera : INotifyPropertyChanged  
 {  
 ///&lt;summary&gt;  
 /// свойство \_position  
 /// значение типа класса Vector3  
 /// расположения камеры  
 ///&lt;/summary&gt;  
 private Vector3 \_position;  
  
 ///&lt;summary&gt;  
 /// свойство \_observerPoint  
 /// объект класса Vector3  
 /// вектор расположения точки наблюдения  
 ///&lt;/summary&gt;  
 private Vector3 \_observerPoint;  
  
 ///&lt;summary&gt;  
 /// свойство \_focus  
 /// значение типа double   
 /// фокус камеры  
 ///&lt;/summary&gt;  
 private double \_focus;  
  
 }  
 }  
</programlisting>  
 <section xml:id="классы-модуля-диалоговых-окон">  
 <title>Классы модуля диалоговых окон</title>  
 <para>  
 К модулю диалоговых окон относятся такие классы, как: \*  
 <emphasis>Dialog\_interface</emphasis>, \*  
 <emphasis>Pattern\_dialog</emphasis>, \*  
 <emphasis>DialogEditor</emphasis>.  
 </para>  
 <para>  
 Более подробно рассмотрим класс  
 <emphasis>Dialog\_interface</emphasis>.  
 </para>  
 <programlisting>  
///&lt;summary&gt;  
 /// Класс Dialog\_interface   
 /// интерфейс для создания,редактирования диалоговых окон  
 /// родительский класс Form  
 ///&lt;/summary&gt;  
 public partial class Dialog\_interface : Form  
 {  
 ///&lt;summary&gt;  
 /// свойство editor  
 /// объект класса DialogEditor  
 /// редактирование диалоговых окон   
 ///&lt;/summary&gt;  
 DialogEditor editor = new DialogEditor();  
  
 ///&lt;summary&gt;  
 /// Конструктор класса Dialog\_interface  
 /// инициализация графических компонентов  
 ///&lt;/summary&gt;  
 public Dialog\_interface()  
 {  
 InitializeComponent();  
 }  
 }  
</programlisting>  
 </section>  
 <section xml:id="классы-редактора-написания-кода">  
 <title>Классы редактора написания кода</title>  
 <para>  
 К модулю редактора написания кода относятся такие классы, как:  
 \* <emphasis>Code\_editor</emphasis>,\*  
 <emphasis>Pattern\_code</emphasis>, \*  
 <emphasis>Expert\_system</emphasis>, \*  
 <emphasis>Interpreter</emphasis>.  
 </para>  
 <para>  
 Более подробно рассмотрим класс  
 <emphasis>Code\_editor</emphasis>.  
 </para>  
 <programlisting>  
 ///&lt;summary&gt;  
 /// Класс Code\_editor  
 /// интерфейс для редактирования кода  
 /// родительский класс Form  
 ///&lt;/summary&gt;  
 public partial class Code\_editor : Form  
 {  
 ///&lt;summary&gt;  
 /// Конструктор класса Code\_editor  
 /// инициализация графических компонентов  
 ///&lt;/summary&gt;  
 public Code\_editor()  
 {  
 InitializeComponent();  
 }  
 ///&lt;summary&gt;  
 /// свойство code  
 /// объект класса Pattern\_code  
 /// быстрое применение шаблонов  
 ///&lt;/summary&gt;  
 Pattern\_code code = new Pattern\_code();  
 }  
</programlisting>  
 </section>  
 <section xml:id="классы-системы-частиц">  
 <title>Классы системы частиц</title>  
 <para>  
 К модулю системы частиц относятся такие классы, как: \*  
 <emphasis>Particle\_system</emphasis>, \*  
 <emphasis>ParticleEvent</emphasis>, \*  
 <emphasis>Particle</emphasis>, \*  
 <emphasis>Patricle\_DB</emphasis>.  
 </para>  
 <para>  
 Более подробно рассмотрим класс  
 <emphasis>Particle\_system</emphasis>.  
 </para>  
 <programlisting>  
 ///&lt;summary&gt;  
 /// Класс Particle\_system   
 /// интерфейс для системы частиц  
 /// родительский класс Form  
 ///&lt;/summary&gt;  
 public partial class Particle\_system : Form  
 {  
 ///&lt;summary&gt;  
 /// свойство particle  
 /// объект класса Particle  
 /// представление частицы  
 ///&lt;/summary&gt;  
 Particle = new Particle();  
  
 ///&lt;summary&gt;  
 /// Конструктор класса Particle\_system  
 /// инициализация графических компонентов  
 ///&lt;/summary&gt;  
 public Particle\_system()  
 {  
 InitializeComponent();  
 }  
 }  
</programlisting>  
 </section>  
 <section xml:id="классы-редактора-местности">  
 <title>Классы редактора местности</title>  
 <para>  
 К модулю редактора местности относятся такие классы, как: \*  
 <emphasis>terrain\_editor</emphasis>, \*  
 <emphasis>Relief\_map</emphasis>, \*  
 <emphasis>PaintTexture</emphasis>, \*  
 <emphasis>Generation\_rule</emphasis>.  
 </para>  
 <para>  
 Более подробно рассмотрим класс  
 <emphasis>terrain\_editor</emphasis>.  
 </para>  
 <programlisting>  
///&lt;summary&gt;  
 /// Класс terrain\_editor   
 /// интерфейс для редактора местности  
 /// родительский класс Form  
 ///&lt;/summary&gt;  
 public partial class terrain\_editor : Form  
 {  
 ///&lt;summary&gt;  
 /// свойство rm  
 /// объект класса Relief\_map  
 /// карта рельефа, объема и системы уровней  
 ///&lt;/summary&gt;  
 Relief\_map rm = new Relief\_map();  
  
 ///&lt;summary&gt;  
 /// Конструктор класса terrain\_editor  
 /// инициализация графических компонентов  
 ///&lt;/summary&gt;  
 public terrain\_editor()  
 {  
 InitializeComponent();  
 }  
 }  
</programlisting>  
 </section>  
 <section xml:id="классы-модуля-воспроизведения-звука">  
 <title>Классы модуля воспроизведения звука</title>  
 <para>  
 К модулю воспроизведения звука относятся такие классы, как: \*  
 <emphasis>Sound\_interface</emphasis>, \*  
 <emphasis>SoundEditor</emphasis>, \*  
 <emphasis>Sound\_DB</emphasis>, \*  
 <emphasis>SoundEvent</emphasis>.  
 </para>  
 <para>  
 Более подробно рассмотрим класс  
 <emphasis>Sound\_interface</emphasis>.  
 </para>  
 <programlisting>  
///&lt;summary&gt;  
 /// Класс Sound\_interface  
 /// интерфейс для системы редактирования звуковых эффектов  
 /// родительский класс Form  
 ///&lt;/summary&gt;  
 public partial class Sound\_interface : Form  
 {  
 ///&lt;summary&gt;  
 /// свойство se  
 /// объект класса SoundEditor  
 /// редактор звуковых дорожек  
 ///&lt;/summary&gt;  
 SoundEditor se = new SoundEditor();  
 {  
 InitializeComponent();  
 }  
 }  
</programlisting>  
 </section>  
 <section xml:id="классы-модуля-отображения-и-взаимодействия-со-сценой">  
 <title>Классы модуля отображения и взаимодействия со  
 сценой</title>  
 <para>  
 К модулю отображения и взаимодействия со сценой относятся  
 такие классы, как: \* <emphasis>Scene</emphasis>, \*  
 <emphasis>DrawContainer</emphasis>, \*  
 <emphasis>SceneObject</emphasis>, \*  
 <emphasis>Vector3</emphasis>.  
 </para>  
 <para>  
 Более подробно рассмотрим класс <emphasis>Scene</emphasis>.  
 </para>  
 <programlisting>  
//&lt;summary&gt;  
 /// Класс Scene   
 /// взаимодействия пользователя со сценой  
 ///&lt;/summary&gt;  
 public class Scene  
 {  
 ///&lt;summary&gt;  
 /// свойство MainCamera  
 /// объект класса Camera  
 /// проецирование изображения с камеры на сцену  
 ///&lt;/summary&gt;  
 public Camera MainCamera;  
 public ObservableCollection&lt;SceneObject&gt; ObjectList { get; set; }  
</programlisting>  
 <para>  
 Методы класса <emphasis>Scene</emphasis> представлены как: \*  
 конструктор класса <emphasis>Scene</emphasis>, \* геттеры и  
 сеттеры свойств <emphasis>Scene</emphasis> (\*\_far<emphasis>,  
 </emphasis>\_isZbuffer<emphasis>, </emphasis>\_near<emphasis>,  
 </emphasis>\_isPerspective<emphasis>,  
 </emphasis>\_isWireframe<emphasis>,  
 </emphasis>\_isFilled<emphasis>), </emphasis> метод добавления  
 объекта на сцену.  
 </para>  
 <programlisting>  
 ///&lt;summary&gt;  
 /// Метод Far типа double  
 /// получение и отправка текущего   
 /// значения отдаления  
 ///&lt;/summary&gt;  
 public double Far {  
 get  
 {  
 return \_far;  
 }  
 set  
 {  
 \_far = value;  
 ScenePropertiesChanged();  
 }  
 }  
  
 ///&lt;summary&gt;  
 /// Конструктор класса Scene  
 /// Создание объекта класса Scene   
 /// по умолчанию  
 ///&lt;/summary&gt;  
 public Scene()  
 {  
 MainCamera = new Camera(-4, 2, 4);  
 MainCamera.OnPositionChangedEvent += ScenePropertiesChanged;  
 MainCamera.OnObserverPointChangedEvent += ScenePropertiesChanged;  
 MainCamera.PropertyChanged += MainCamera\_PropertyChanged;  
 ObjectList = new ObservableCollection&lt;SceneObject&gt;();  
 ObjectList.CollectionChanged += ObjectCollectionChanged;  
 \_isPerspective = true;  
 \_isFilled = true;  
 \_isWireframe = false;  
 \_isZBuffer = true;  
 \_near = 0.2;  
 \_far = 100.0;  
 }  
 }  
</programlisting>  
 </section>  
 </section>  
 <section xml:id="сценарии-использования">  
 <title>Сценарии использования</title>  
 <section xml:id="environment-artist">  
 <title><emphasis>Environment artist</emphasis></title>  
 <para>  
 Данный продукт может использоваться для создания любого  
 рельефа, иерархии карт, процедурной генерации местности с  
 помощью удобного инструментария редактора местности  
 terrain\_editor.   
 </para>  
 </section>  
 <section xml:id="artist">  
 <title><emphasis>3D-artist</emphasis></title>  
 <para>  
 Данный продукт может использоваться для необычной и современной  
 презентации своих проектов. Например, перенос 3D-модели в  
 Blender на сцену VR Engine. С помощью интуитивно-понятного  
 интерфейса, похожего на аналоги, человек без особого  
 технического образования может добавить свои файлы, выбрать  
 интересующий шаблон сцены, освещения и получить за короткий  
 срок готовую VR-презентацию своего продукта и добавить себе в  
 портфолио.  
 </para>  
 </section>  
 <section xml:id="software-programmer">  
 <title><emphasis>Software Programmer</emphasis> </title>  
 <para>  
 Данный вид пользователя может разрабатывать более уникальные продукты  
 на основе шаблонов или на чистом ЯП. Добавлять новые модули  
 проектирования, утилиты для упрощения работы и пр.  
 </para>  
 </section>  
 <section xml:id="vfx-designer">  
 <title><emphasis>VFX-дesigner</emphasis></title>  
 <para>  
 Художник по визуальным эффектам может использовать данный продукт для создания  
 спецэффектов, настройки и доработки существующих шаблонов  
 эффектов, частиц с помощью модуля <emphasis>Particle\_system</emphasis>.   
 </para>  
 </section>  
 </section>  
 </section>  
</article>

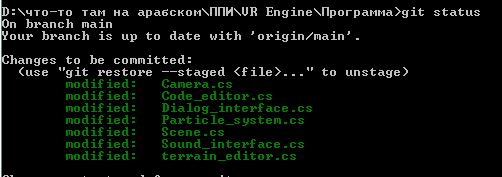


Рисунок 8 – Добавление измененных файлов



Рисунок 9 – Добавление XML-документации и DocBook

Выполненный DocBook был конвертирован в формат markdown и добавлен в файл README.md. С данным файлом можно ознакомиться в репозитории.