Федеральное государственное автономное образовательное учреждение высшего образования «Санкт-Петербургский национальный исследовательский университет ИТМО»

**Отчет по производственной практике**

«Разработка 3D-интерфейса на UWP для системы безопасности»

Студента Пакин М.В.

Группы R3345

Дата начала практики 05.02.2020

Дата окончания практики 04.03.2020

Руководитель практики от кафедры Лукичев Д.В.

Руководитель практики от предприятия Смольский И.А.

Пакин М.В.\_\_\_\_\_\_\_\_\_\_\_\_ (подпись)

Лукичев Д.В.\_\_\_\_\_\_\_\_\_\_\_\_ (подпись)

Санкт-Петербург 2020

**Оглавление:**

[**1.** **Введение** 1](#_Toc34036177)

[**1.1** **Сведения о предприятии ООО «ОКБ «Авгит»** 1](#_Toc34036178)

[**1.2** **Цель работы** 2](#_Toc34036179)

[**1.3** **Постановка задач** 2](#_Toc34036180)

[**2.** **Структура программы** 2](#_Toc34036181)

[**2.1** **Файлы проекта** 2](#_Toc34036182)

[**2.2** **Алгоритм построения 3D-модели** 5](#_Toc34036183)

[**3.** **Итоговый результат для пользователя** 6](#_Toc34036184)

[**3.1** **Вкладка Help** 7](#_Toc34036185)

[**3.2** **Вкладка File** 7](#_Toc34036186)

[**3.3** **Вкладка Camera** 9](#_Toc34036187)

[**3.4** **Использование** 9](#_Toc34036188)

[**4.** **Заключение** 12](#_Toc34036189)

[**5.** **Список литературы** 13](#_Toc34036190)

1. **Введение**
   1. **Сведения о предприятии ООО «ОКБ «Авгит»**

Группа компаний «ТвинПро» создана в 2016 г. [1] и включает производственные предприятия – ООО «ОКБ «Авгит» (г. Санкт-Петербург – основано в 2013 г.), ООО «ЕС-пром», (г. Самара. основано в 2011 г.) и головное предприятие ООО «ТвинПро» (г. Москва, основано в 2013 г.) [6].

Группа предприятий осуществляет разработку, производство, поставку, продвижение и техническую поддержку сложных высокотехнологичных систем безопасности. Название группы компаний образовано от словосочетания «ПРОектныеПРОдукты», к которым относятся технические средства, для комплектации, конфигурирования, монтажа, настройки и обслуживания которых определяющее значение имеют особенности реализуемого проекта. Такие средства требуют высокой специализированной квалификации сотрудников, работающих с ними, на всех этапах реализации проекта, однако при этом обеспечивают максимально достижимые уровни функциональности, информативности, оперативности, надежности, живучести, минимизации влияния человеческого фактора и других потребительских характеристик.

Группа компаний «ТвинПро» поставляет на рынок как продукцию собственного производства, так и оборудование своих зарубежных партнеров, что позволяет предложить пользователям функционально полный спектр систем, способных обеспечить оптимальный уровень безопасности защищаемых объектов различного назначения.

Основными продуктовыми линейками группы компаний являются следующие:

* Система охранного освещения «Заря»
* Системы периметральной сигнализации «Пунктир»

Взаимодействие со своими клиентами ГК «ТвинПро» осуществляет силами своего московского подразделения – ООО «ТвинПро».

* 1. **Цель работы**

Для удобства эксплуатации охранной системы требуется создать понятный интерфейс. Наилучшим вариантом будет 3D изображение плана здания, где можно увидеть место срабатывания сигнализации, а также любую другую информацию, которую пожелает пользователь (допустим, изображение с камеры видеонаблюдения). Также желательно использовать кроссплатформенные фреймворки, так как изначально трудно сказать, какое именно устройство будет использоваться.

В качестве цели поставлена разработка кроссплатформенного 3D интерфейса для системы безопасности.

* 1. **Постановка задач**

В связи с тем, что время ограничено, будет использованы наиболее подходящие фреймворки, чтобы успеть максимум за предоставленное время. Также будем отдавать приоритет фреймворкам с языком программирования C#.

Для первоначальной настройки нам понадобится план здания. Так как зачастую он выполняется в AutoCad’е, мы будем требовать DXF-формат файла с поли линиями в качестве стен.

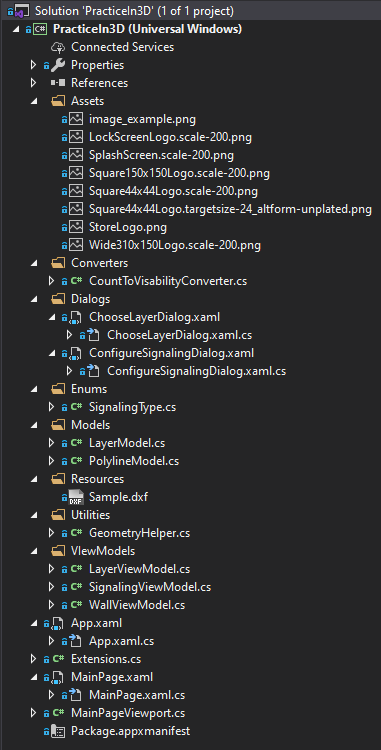
Базовый фреймворк будет **UWP**, так как он кроссплатформенный. Также будут использованы следующие Open-Source Nuget-пакеты:

* **HelixToolkit.UWP** – для отображения 3D графики [2]
* **netDxf.netstandard** – для чтения DXF-файлов [4]
* **Poly2Tri.NetStandard** – для выполнения алгоритма триангуляции [3]

1. **Структура программы**
   1. **Файлы проекта**

Все исходники программы можно найти на git-репозитории по ссылке: <https://github.com/maxPakin/PracticeIn3D>. В файле README.md находится дневник практики, где расписаны все задачи, которые ставились на протяжении практики.

Разберем все файлы и папки проекта, которые можно увидеть на Рис. 1.



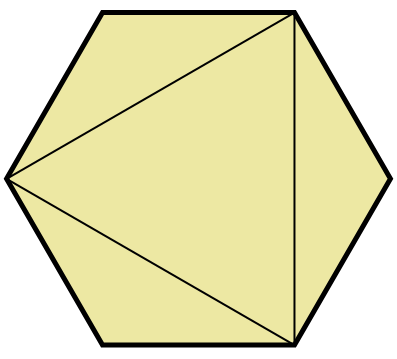
*Рис. 1. Структура проекта*

* **Connected Services** и **Properties** - создаются стандартно, поэтому их можно не оговаривать.
* **References** – список проектов, которые использует данный проект. Туда входят те Nuget-пакеты, о которых говорилось ранее.
* **Assets** – используемые изображения, шрифты и т.д. Данная папка осталась без изменений, в ней находятся стандартные изображения, которые требуются проекту.
* **Converters** – папка с конверторами для xaml страничек, которые использует UWP. Сейчас там только один конвертер, который создан на будущее и не используется.
* **Dialogs** – папка с диалоговыми окнами, где пока находятся только 4 файла для 2 диалоговых окна. Одно из них – уточняет, какой именно слой хочет загрузить пользователь из DXF файла, а другое – изменяет настройки сигнализации.
* **Enums** – папка для общих для проекта enum-структур. Enum-структуры используются для обозначения ограниченного количества дискретных значений.
* **Models** – папка для моделей. Модели используются для хранения минимальной информации о той или иной сущности.
* **Resources** – папка для хранения ресурсов. На данный момент так только DXF файл-пример для загрузки.
* **Utilities** – папка для хелперов, где находится только один класс – *GeometryHelper*. Данный класс выносит очевидную логику геометрии для упрощения понимания кода.
* **ViewModels** – папка для View-моделей, которые используются, чтобы хранить информацию для отображения. Допустим, у нас есть Яблоко. В модели яблока мы можем хранить только дату его созревания. А уже во View-модели будем хранить его цвет, который вычисляется на основе текущей даты и даты его созревания. Таким образом, на диске мы не храним цвет яблока, а только дату его созревания, но при отображении его цвет будет меняться.
* **App.xaml** и **App.xaml.cs** – стандартные файлы для настройки приложения.
* **Extensions.cs** – файл для методов расширения. Здесь хранятся те методы, которые хотелось бы иметь стандартно, но их нет. Поэтому для упрощения понимания кода, эти методы вынесены с этот файл.
* **MainPage.xaml** и **MainPage.xaml.cs** – файлы главной активности. В этом файле соответственно указана логика отображения и логика поведения главной активности.
* **MainPageViewport.cs** – наследник от класса Viewport3DX из пакета HelixToolkit [2]. Было применено наследование, так как была необходимость перегрузить некоторые методы класса Viewport3DX.
* **Package.appxmanifest** – файл для описания самых базовых характеристик кроссплатформенного приложения. Допустим, требуется ли приложению доступ к камере. В нашем случае – нет.

Более подробная информация о коде в программах описана в комментариях в коде программы. Но стоит акцентировать внимание на алгоритме построения 3D модели из чертежа плана здания.

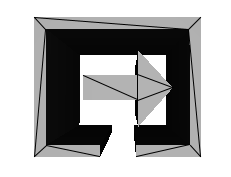
* 1. **Алгоритм построения 3D-модели**

Большая часть 3D графики состоит из треугольников, и HelixToolkit [2] не исключение. После чтения данных из файла мы получаем коллекцию поли линий. Алгоритм пройдет по каждой поли линии. Каждую создадут на плоскости Z = 0, на плоскости Z = 50, а потом соединят соответственные точки между собой. Полученные плоскости прямоугольников поделим на 2 треугольника каждый. Осталось поделить на треугольники верхнюю плоскость, что уже не самая легкая задача [6]. Алгоритм деления многоугольника на треугольники называется триангуляцией [5]. Существуют простые алгоритмы для правильных многоугольников. Допустим, алгоритм «отрезания уха», когда мы отрезаем от многоугольника треугольники, состоящие из соседних точек. Пример результата этого алгоритма изображен на Рис. 2.

**

*Рис. 2. Результат алгоритма триангуляции «отрезание уха»*

Но в нашем случае такой алгоритм не подходит, так как многоугольник будет правильным крайне редко. Поэтому будем использовать более сложные алгоритмы, которые уже протестированы и подходят для любого многоугольника. Такой я нашёл в Nuget-пакете Poly2Tri.NetStandard [3], который как раз кроссплатформенный. На Рис. 3 изображен результат алгоритма, но каждый полученный треугольник намеренно выделен обводкой, чтобы стало видно полученные треугольники.



*Рис. 3. Результат триангуляции*

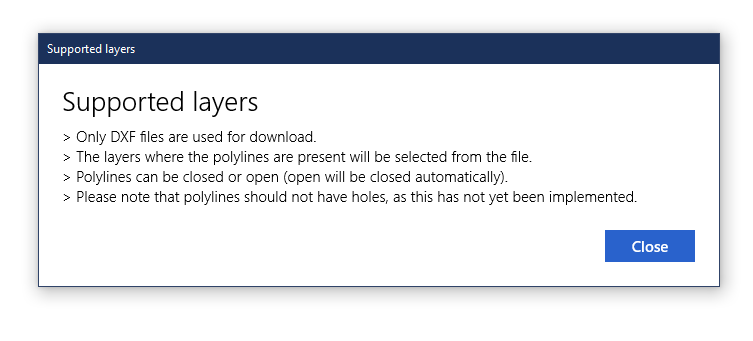
Нижнюю грань нет смысла создавать, так как её не видно.

1. **Итоговый результат для пользователя**

После запуска программы пользователю отображается страница MainPage с пустым пространством и вкладками сверху.

* 1. **Вкладка Help**

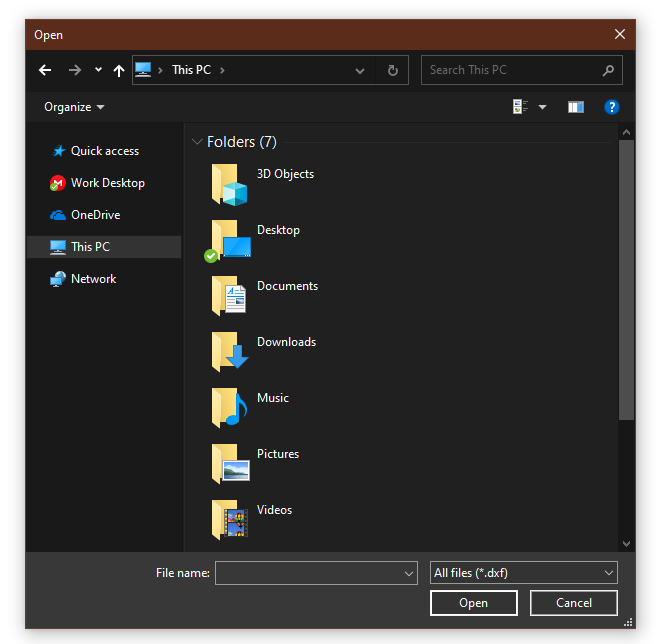
В данной вкладке только один пункт – Supported Layers. При выборе отображает справка о доступных форматах файла, которую можно посмотреть на Рис. 4.



*Рис. 4. Диалоговое окно Supported Layers.*

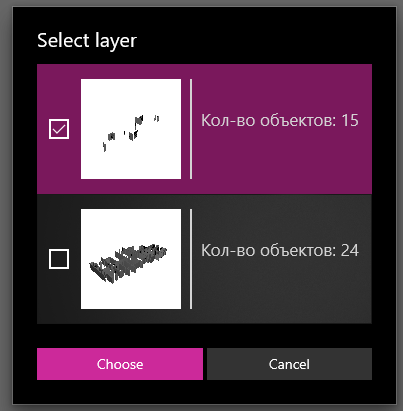
* 1. **Вкладка File**

В данной вкладке есть возможность выбрать файл с диска, либо файл-пример, который лежит в папке Resources. Если выбрать «Open…», то появится окно выбора файла с диска, которое изображено на Рис. 5. Если же выбрать «Open Sample», то мы увидим поведение, будто мы открыли файл с примером с диска. Второй пункт нужен лишь для быстрой демонстрации функционала.



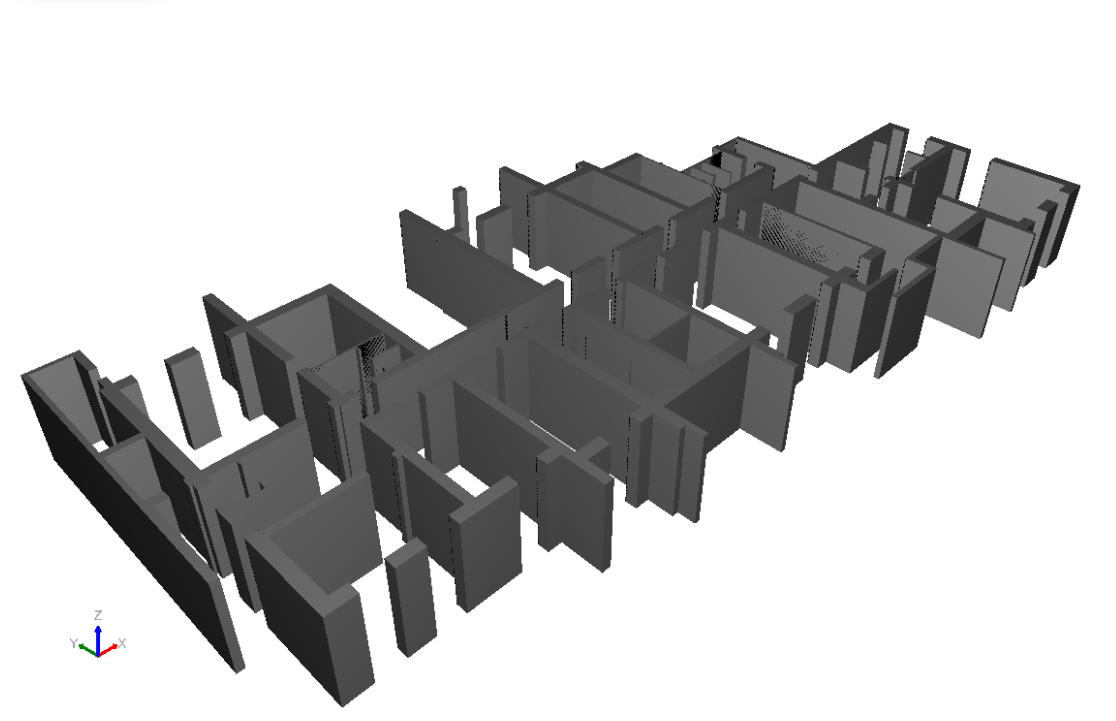
*Рис. 5. Диалоговое окно выбора файла*

В окне сразу настроено, что можно выбрать только файлы DXF. После выбора файла появится окно выбора слоя, если этот выбор имеет место быть. Т.е. если в файле нет подходящих слоёв для загрузки, то программа попросит ознакомиться со справкой Help - Supported Layers. Если в файле подходит только один слой, то программа сразу грузит этот слой. А если несколько слоёв подходят, то программа вывод диалоговое окно ChooseLayerDialog, которое изображено на Рис. 6.



*Рис. 6. Диалоговое окно ChooseLayerDialog.*

В окне будет отображён список подходящих слоёв, а также их превью. Есть возможность выбрать несколько слоёв, так как возможна ситуация, когда на одном слое хранятся стены, а на другом – перегородки. После нажатия на кнопку Choose выбранные слои выводятся перед пользователем, как на Рис. 7.



*Рис. 7. Пример загруженного плана здания*.

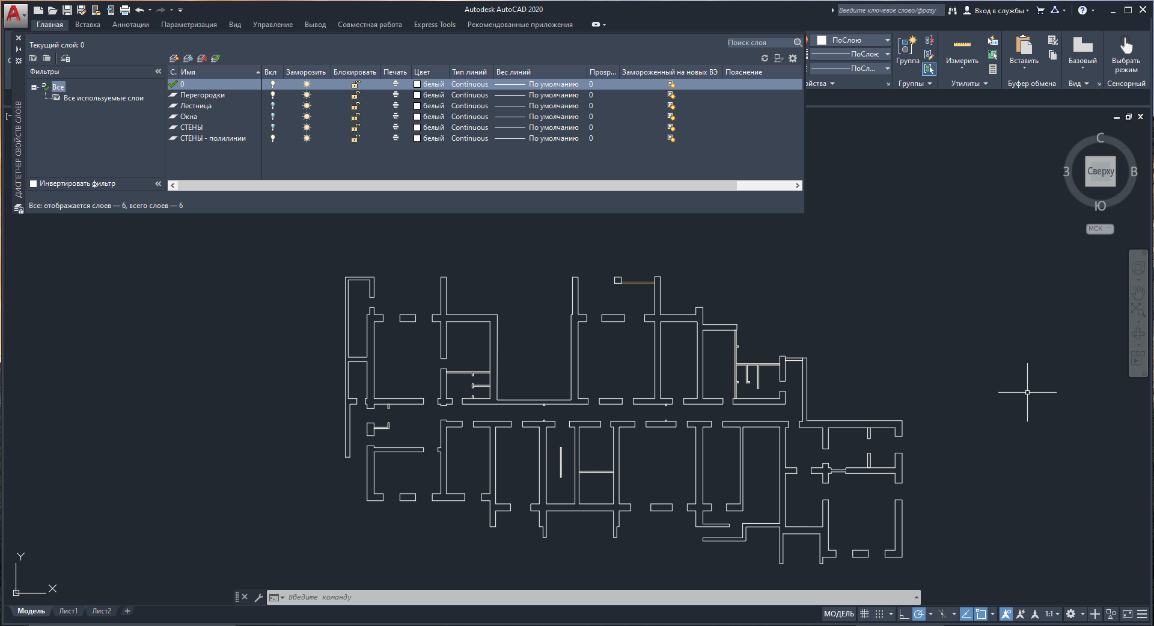
* 1. **Вкладка Camera**

Во вкладке Camera есть лишь два пункта: «Look from the top» и «Look around». На Рис. 8 и Рис. 9 видна разница между этими режимами.

|  |  |
| --- | --- |
| *image*  *Рис. 8. Режим Look from the top* | *image*  *Рис. 9. Режим Look around* |

* 1. **Использование**

Изначально у пользователя должен быть проект с планом здания в AutoCad. Пример такого проекта виден на Рис. 10.



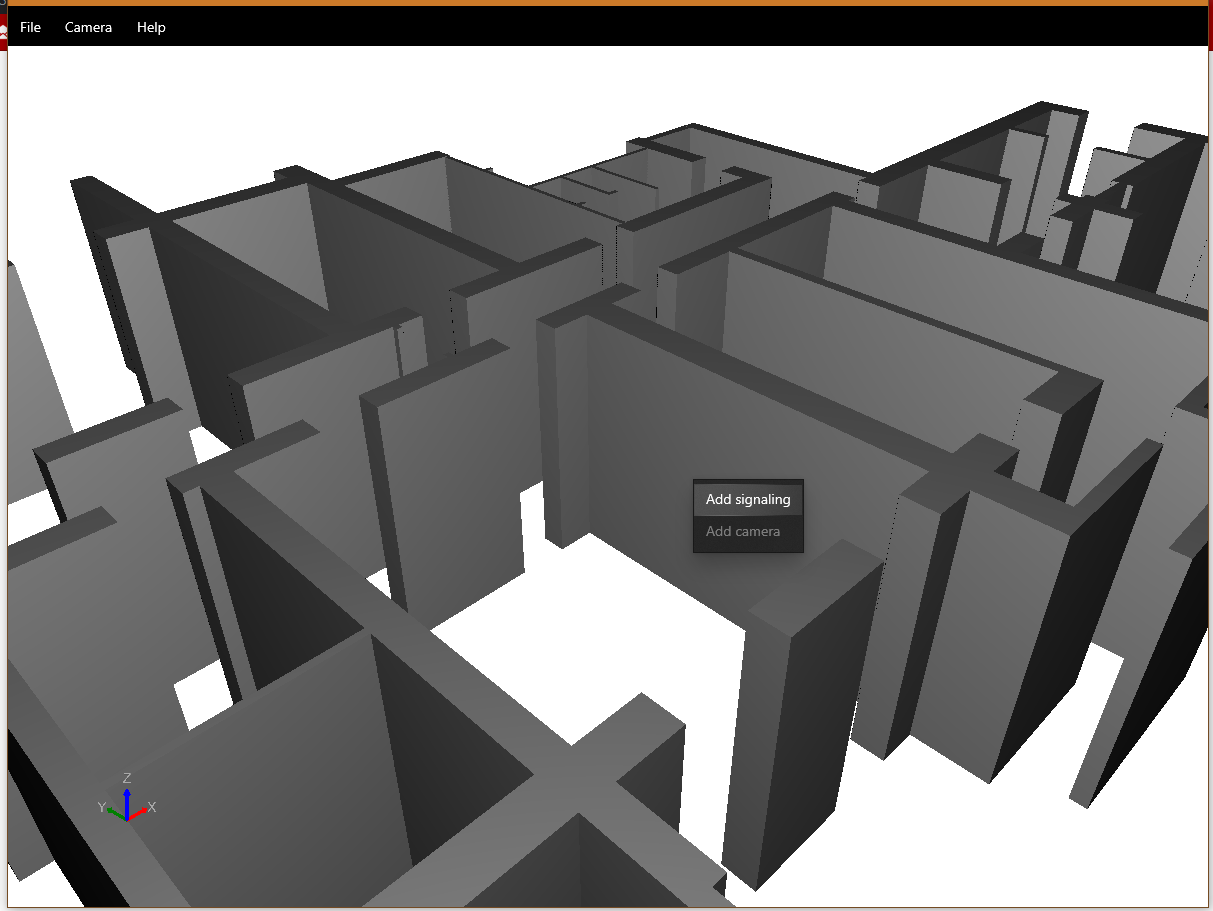
*Рис. 10. Пример плана здания в AutoCad 2020*

В примере создано несколько слоёв для демонстрации функционала, а именно:

* **Перегородки** - слой, на котором размещены перегородки поли линиями, так и обычными линиями. Обычные линии будут игнорироваться
* **Лестница** - слой, на котором нет поли линий, поэтому весь слой будет убран из выбора
* **Окна** - слой, аналогичный лестницам
* **СТЕНЫ** - слой, где стены размещены обычными линиями
* **СТЕНЫ – поли линии** - слой, где стены размещены поли линиями. Этот слой создан обычной обводкой слоя **СТЕНЫ**

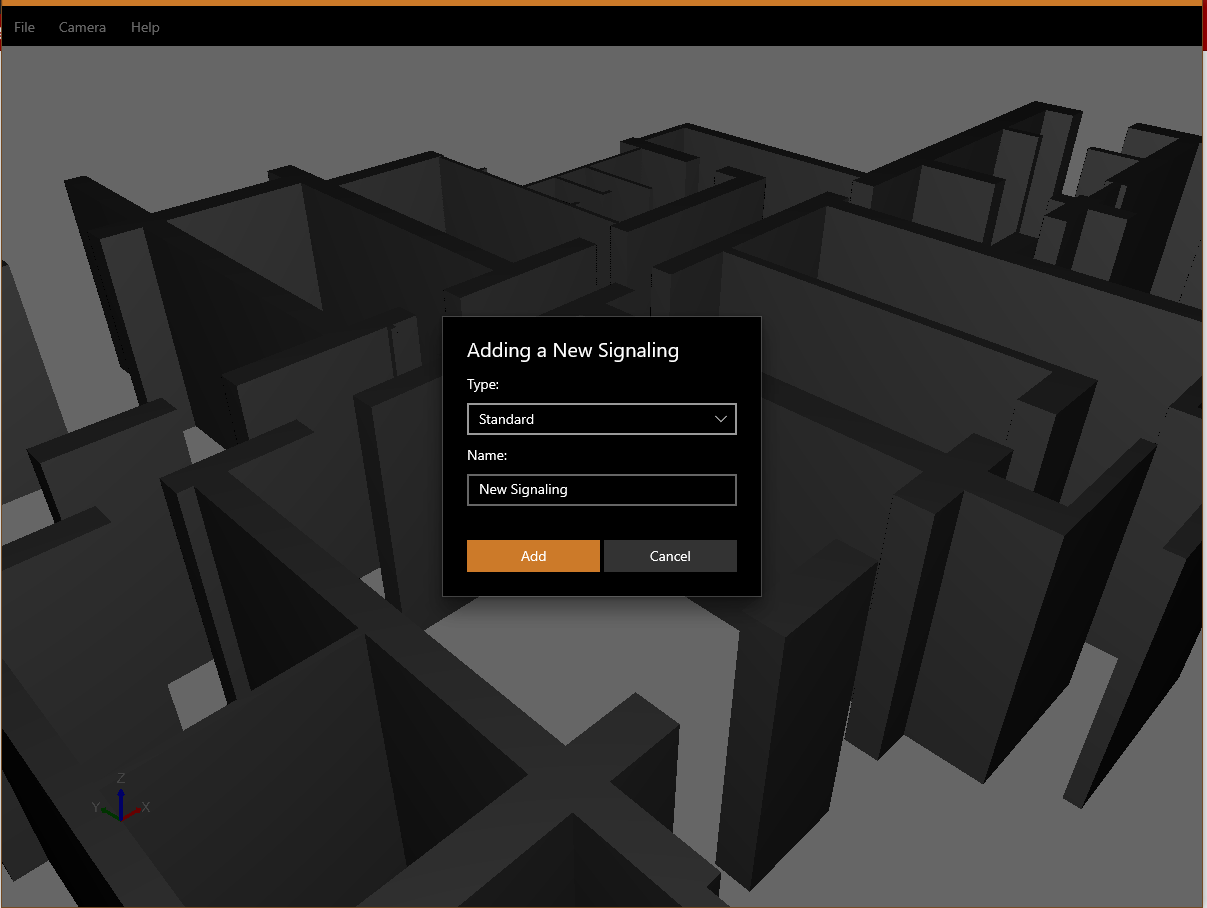
Пользователь должен понимать, что будут загружены только два слоя: **СТЕНЫ – поли линии** и **Перегородки**, т.к. только они удовлетворяют условиям поддерживаемых файлов.

После загрузки пользователь может двигать мышкой (либо пальцем) загруженный макет, а также приближать либо отдалять его. Также есть возможность создать сигнализацию. На данный момент реализован минимальный функционал: добавление на план здания и выбор свойств. Для добавления надо нажать ПКМ на нужном месте и выбрать «Add Signaling», как показано на Рис. 11.

**

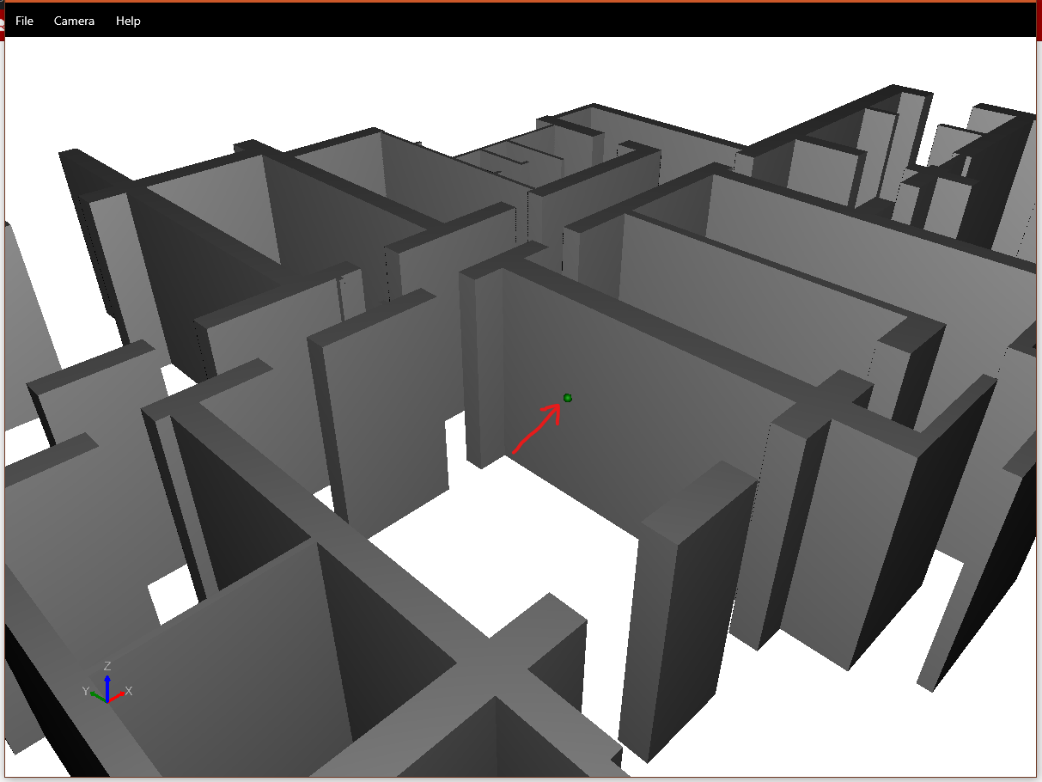
*Рис. 11. Способ добавления сигнализации*

При нажатии появится окно настроек сигнализаций, как на Рис. 12.



*Рис. 12. Диалоговое окно ConfigureSignalingDialog*

Эти свойства практически не имеют функционала. Тип сигнализации влияет на цвет, отображаемый на плане здания, а всё остальное только хранится. Пример созданной сигнализации виден на Рис. 13.

**

*Рис. 13. Пример созданной сигнализации*

В теории, дальше надо написать сохранение проекта в файл и его чтение. Добавление более конкретных свойств к сигнализации и реализация добавления камеры видеонаблюдения.

1. **Заключение**

В результате прохождения производственной практики были получены следующие результаты:

* Произведено ознакомление с фреймворком UWP; с пакетами HelixToolkit [2] и netDxf [4]; с алгоритмами триангуляции
* Приобретены дополнительные навыки работы в пакете «AutoCad» и в составлении систем безопасности

1. **Список литературы**
2. Официальный сайт TwinPro [Электронный ресурс]. – режим доступа: свободный. <http://www.twinpro.ru/about>
3. Git-репозиторий Nuget-пакета HelixToolkit [Электронный ресурс]. – режим доступа: свободный. <https://github.com/helix-toolkit/helix-toolkit>
4. Git-репозиторий Nuget-пакета Poly2Tri [Электронный ресурс]. – режим доступа: свободный. <https://github.com/mapbox/earcut>
5. Git-репозиторий Nuget-пакета netDxf [Электронный ресурс]. – режим доступа: свободный. <https://github.com/haplokuon/netDxf>
6. Википедия – свободная энциклопедия [Электронный ресурс]. – режим доступа: свободный. <https://en.wikipedia.org/wiki/Triangulation_(geometry)>
7. Викиконспекты университета ИТМО [Электронный ресурс]. – режим доступа: свободный. <https://neerc.ifmo.ru/wiki/index.php?title=Триангуляция_полигонов_(ушная_%2B_монотонная)>
8. Официальный сайт ООО «ОКБ «Авгит» [Электронный ресурс]. – режим доступа: свободный. <http://fors.gtechs.ru/>