ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

***«*САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО»**

**ИНСТИТУТ КОМПЬЮТЕРНЫХ НАУК И ТЕХНОЛОГИЙ**

**ВЫСШАЯ ШКОЛА ИНТЕЛЛЕКТУАЛЬНЫХ СИСТЕМ И СУПЕРКОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ**

**Отчет о прохождении   
стационарной производственной преддипломной практики**

**на тему:** **«Разработка модуля процедурной генерации трехмерных моделей зданий»**

Ларионова Алексея Сергеевича, гр. 3530203/60101

**Направление подготовки:** 02.03.03 Математическое обеспечение и администрирование информационных систем

|  |
| --- |
| **Место прохождения практики:** СПбПУ, ИКНТ, ВШИСиСТ |

*(указывается наименование профильной организации или наименование структурного подразделения*

|  |
| --- |
|  |

*ФГАОУ ВО «СПбПУ», фактический адрес)*

**Сроки практики:** с 21.04.2020 по 04.05.2020.

**Руководитель практики от ФГАОУ ВО «СПбПУ»:**

|  |
| --- |
| Туральчук К.А., ст. преп. ВШИСиСТ |

*(Ф.И.О., уч. степень, должность)*

**Оценка:**

Руководитель практики

от ФГАОУ ВО «СПбПУ» Туральчук К.А.

Обучающийся Ларионов А.С.

Дата: \_\_\_\_\_\_\_\_\_\_

ОГЛАВЛЕНИЕ

[ВВЕДЕНИЕ 4](#_Toc40133726)

[РЕАЛИЗАЦИЯ МОДУЛЯ ГЕНЕРАЦИИ 5](#_Toc40133727)

[РЕАЛИЗАЦИЯ ВИЗУАЛИЗАЦИИ МОДЕЛЕЙ 9](#_Toc40133728)

[РЕАЛИЗАЦИЯ АЛГОРИТМА ГЕНЕРАЦИИ МОДЕЛЕЙ ЗДАНИЙ 13](#_Toc40133729)

[ЗАКЛЮЧЕНИЕ 17](#_Toc40133730)

[ПРИЛОЖЕНИЕ 18](#_Toc40133731)

ВВЕДЕНИЕ

В рамках выпускной работы бакалавра (ВКР), рассматривается и реализуется модуль для процедурной генерации трехмерных моделей зданий. Процесс процедурной генерации предполагает ввод нескольких конфигурационных параметров, а затем автоматическую генерацию модели. Модель должна быть осмотрена человеком на предмет удовлетворительного качества или структуры. После этого, модель может быть сгенерирована заново с новыми (или корректированными) параметрами, а подходящая модель может быть экспортирована в стандартный формат файла (OBJ или STL).

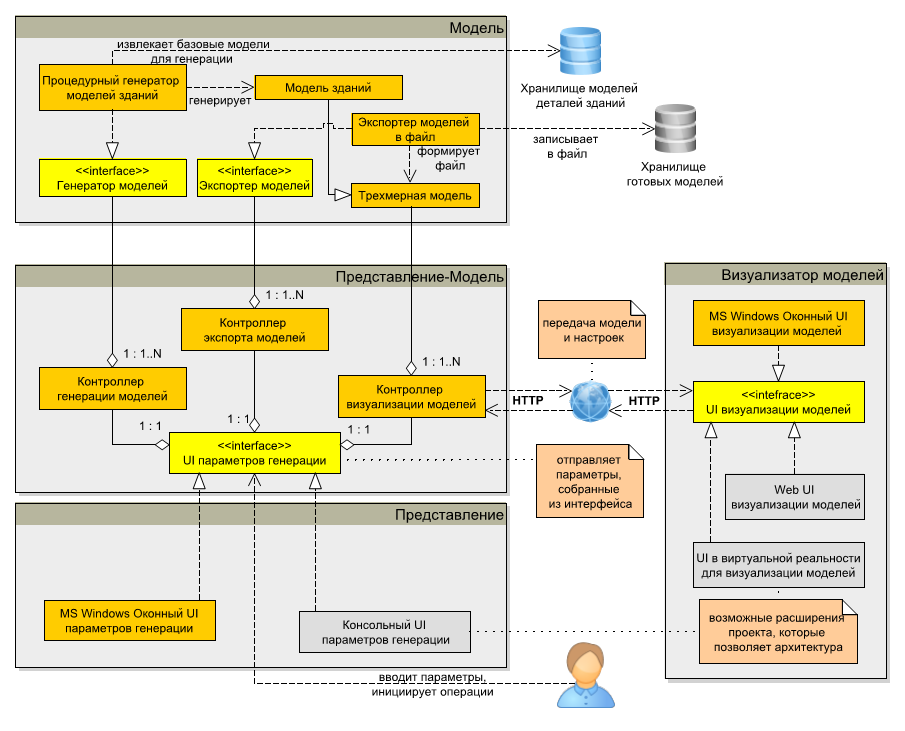
Целью данной работы является практическая реализация модуля в рамках ВКР, с помощью выбранного набора технологий по теоретическим выкладкам, рассмотренным и разработанным в ходе предшествующей работы над проектом (научно-исследовательские работы и пр.).

Для достижения поставленной цели, необходимо решить следующие задачи:

* реализовать общую программную архитектуру модуля и его функционал;
* реализовать графический интерфейс модуля процедурной генерации;
* реализовать дополнительную утилиту для визуализации сгенерированных моделей.

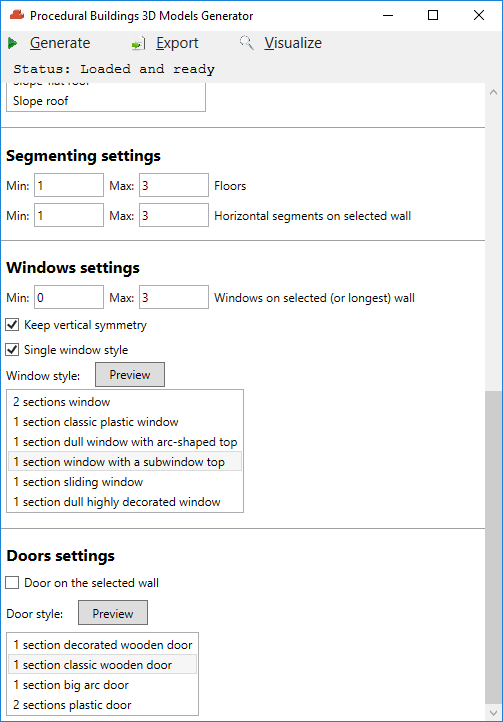
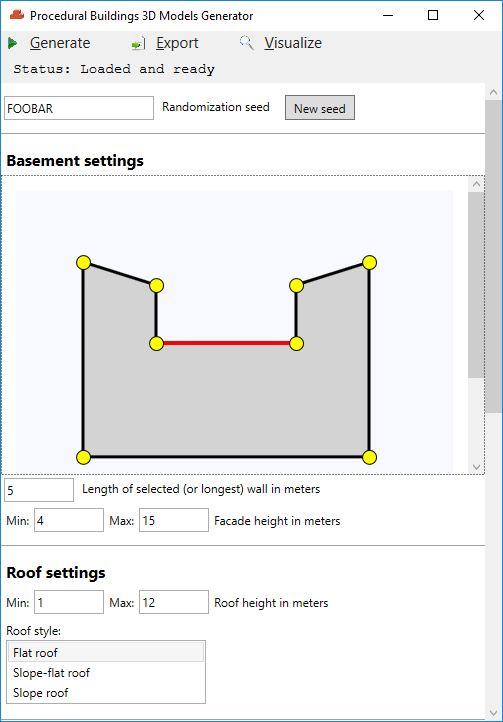
РЕАЛИЗАЦИЯ МОДУЛЯ ГЕНЕРАЦИИ

Модуль генерации, как и предполагалось, реализуется по модели MVVM - вариации модели MVC (Model-View-Controller), которая присуща C# приложениям с графическим интерфейсом, реализованным c помощью технологии Windows Presentation Foundation.



Модуль генерации и визуализация полученных моделей разнесены в два разных приложения, и связываются они с помощью HTTP протокола. Данный подход позволяет обеспечить гибкость встраивания модуля генерации в другие приложения – целевые приложения могут быть реализованы на любом языке и для любой платформы (вплоть до того, что генерация происходит на одной машине, а визуализация на другой, например, в зале виртуальной реальности, или в плагине для системы автоматизированного проектирования). Представление модуля генерации позволяет наличие нескольких реализаций, т.к. представление используется по C# интерфейсу, поэтому ввести реализацию графического интерфейса для другой операционной системы не составляет большого труда.

Реализация графического интерфейса для OS Windows в данной работе состоит из нескольких XAML разметок, каждая из которых представляет секцию настроек процедурного генератора. Эти секции (всего их 5) объединяются в одном окне MainWindow.xaml. Разделение было произведено, с целью возможного дальнейшего расширения, дополнения, изменения внешнего вида интерфейса. Общее окно настроек выглядит следующим образом.

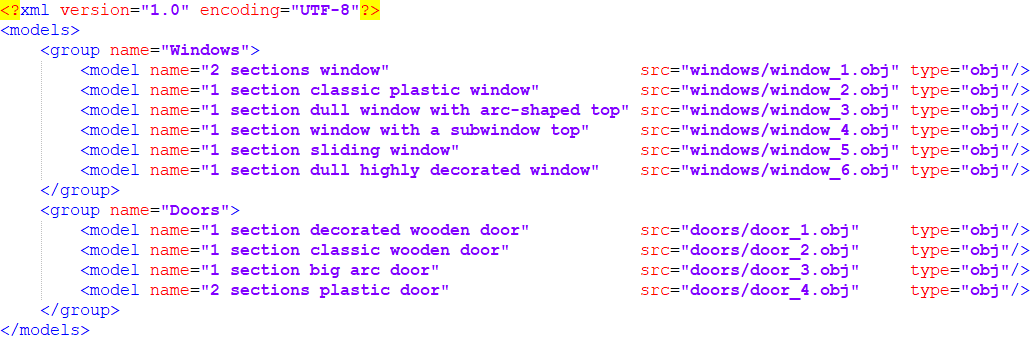


Окно позволяет начать генерацию, экспортировать сгенерированную модель в файл, визуализировать полученную модель (соответственно подключенные по HTTP визуализаторы отобразят модель). Настройки генерации содержат панель для рисования формы фасада здания (можно добавлять, удалять, перемещать точки полигона, выравнивать их). Также можно выбрать одну из сторон, для которой можно указать длину в метрах (чтобы получить модель с необходимым или близким масштабом). Среди других настроек, можно настраивать стиль крыши, окон, дверей (при этом список стилей автоматически согласован с моделями, которыми оперирует генератор, что позволяет легко добавить новую модель в генератор – просто добавить файл, и дать на него ссылку в XML манифесте (см. далее)).

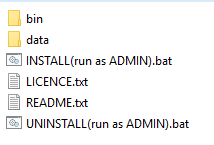
*Представление-модель* модуля вынесено в отдельный проект-библиотеку (опять же, для более гибкой реализации другого интерфейса в будущем). Здесь содержатся контроллеры для генерации, визуализации, экспорта, загрузки моделей стилей с диска. Им делегируются вызовы из графического интерфейса по соответствующим операциям. Также имеются непосредственно представления-модели, т.е. классы, к свойствам (Property) которых осуществляется связывание (Binding) данных из графического интерфейса. Это избавляет от необходимости написания кода по самостоятельному сбору введенных данных из интерфейса. Контроллер генерации, получает все представления-модели и из них формирует параметры генерации в формате для алгоритма генерации (т.е. в виде формальной грамматики).

*Модель* также выделена в отдельный проект, а так как она не использует специальных Windows специфичных операций (как например WPF, WCF), то она компилируется в виде .NET Core библиотеки, которая уже больше подходит для кроссплатформенного использования. Модель реализует генерацию моделей (с помощью библиотеки geometry3sharp) и экспорт моделей (в том числе в файл или в виде потока данных). Логика генерации непосредственно зданий (класс BuildingsModelsGenerator) и их параметров (BuildingsGenerationParameters) отделены абстрактными объектами: интерфейсом IProceduralModelsGenerator и классом GenerationParameters соответственно. Это позволяет в будущем добавить в модель и новые виды генерации другого контента.

Также, *модель* содержит загрузчик моделей деталей (например, конкретных моделей окон, дверей). Для этого используется XML манифест, расположенный по определенному пути относительно приложения. В манифесте перечислены типы моделей (“Doors”, “Windows”), и конкретные модели (их названия в графическом интерфейсе и пути к файлам). Как и обозначалось, для добавления новой модели в генератор достаточно просто добавить строчку в манифест, а для добавления нового вида детали (например, балконы), нужно добавить в манифест новую секцию, а в грамматику генерации новый вид детали и его правила использования.



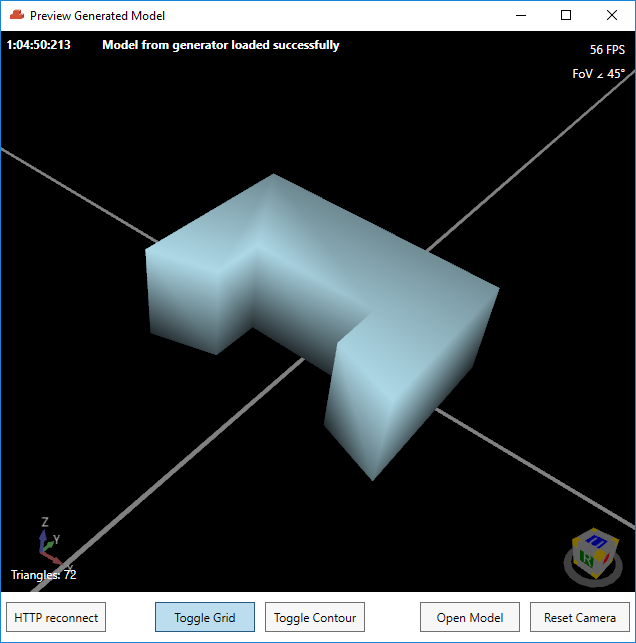
Так как разрабатываемое приложение содержит привязки к внешним файлам, оно не может распространяться как простой исполняемый файл, поэтому написаны .bat скрипты для сбора всех .dll, .exe и файлов данных в виде директории с приложением. Папка bin содержит только исполняемые файлы и динамические библиотеки, data содержит манифест деталей, модели деталей, иконки графического интерфейса. Скрипты INSTALL и UNINSTALL соответственно добавляют (убирают) порт TCP, по которому общаются генератор и визуализатор в список доверенных в Windows (это нужно, чтобы не запускать приложения вручную от имени администратора).



РЕАЛИЗАЦИЯ ВИЗУАЛИЗАЦИИ МОДЕЛЕЙ

Архитектура модуля предполагает реализацию (или использование существующих) программ, которые бы визуализировали сгенерированную модель. При этом созданы они могут быть на любом языке программирования и в том числе запущены на другом компьютере и такая гибкость обеспечена тем, что модуль процедурной генерации поддерживает отправку готовой модели по распространенному протоколу HTTP.

Для выпускной квалификационной работы был разработан графический интерфейс для операционной системы Windows, который реализован на языке C# (средствами библиотеки WPF) и который использует возможности библиотеки Helix Toolkit для визуализации трехмерных моделей. Данный интерфейс является дополнительным инструментом к разрабатываемому модулю, и в будущем может быть реализован любой другой визуализатор (в т.ч. интерфейс для другой операционной системы или плагин для некоторой системы автоматизированного проектирования).



Интерфейс при запуске открывает HTTP сервис (по адресу *http://localhost:64046/wpfVisualizerService*), используя средства встроенной библиотеки WCF языка C#. Удобство данной библиотеки заключается в возможности представлении команд и данных, передаваемых от HTTP сервиса, в виде класса, реализующего интерфейс сервиса. Таким образом с точки зрения программиста не нужно настраивать отдельные HTTP запросы и заниматься их приемом и пересылкой, а вместо этого просто использовать операции как часть обычного класса из языка программирования. Как только сервис визуализатора запущен, он оправляет запрос еще одному сервису – сервису генератора трехмерных моделей, который непосредственно реализует процедурную генерацию и оправку моделей визуализаторам. Данный сервис находится по постоянному адресу *http://localhost:64046/visualizationControllerService*. Сервис визуализатора регистрируется у сервиса генерации и с этого момента сгенерированные модели будут пересылаться для визуализации. Может быть подключено несколько разных визуализаторов (но они должны находиться на разных HTTP сервисах).

Реализованный графический интерфейс также позволяет открывать и визуализировать модели из файловой системы, перемещаться в трехмерном пространстве для осмотра модели, использовать дополнительные элементы визуализации: отображение каркаса модели, сетка X-Y плоскости.

С точки зрения программного кода, от процесса визуализации отделена DLL библиотека языка С#, определяющая общий функционал визуализатора - таким образом можно реализовать этот интерфейс в другом С# проекте и легко заменить существующий визуализатор, или создать проект на другом языке программирования, но с аналогичным функционалом.

public interface IVisualizer

{

string GetDescription();

void VisualizeModel(Stream model, ModelMetaBase modelMeta,

Stream materialLibrary, Stream[] materialFiles);

void Shutdown();

}

Далее, эксклюзивно для модуля WCF С# (для удобного HTTP взаимодействия) выделена DLL библиотека языка С#, определяющая интерфейс сервиса визуализации, который используется конкретно для WCF общения. По факту, сервис определяет такой же функционал, как и приведенный выше интерфейс, но структурирован он специально для WCF.

[ServiceContract]

public interface IVisualizerService

{

[OperationContract]

string GetDescription();

[OperationContract]

void PrepareForModel(ModelMetaBase modelMetadata);

[OperationContract]

void AcceptMaterialLib(Stream model);

[OperationContract]

void PrepareForMaterialFile(string materialFileId);

[OperationContract]

void AcceptMaterialFile(Stream materialFile);

[OperationContract]

void AcceptModel(Stream model);

[OperationContract]

void Visualize();

[OperationContract]

void Shutdown();

}

Сам визуализатор является WPF приложением, которое в XAML разметке содержит элемент <helix:HelixViewport3D> с конфигурационными параметрами, а также элементы управления (кнопки). Программа реагирует на события интерфейса, а также занимается созданием, регистрацией и закрытием HTTP сервисов. При этом необходимо обрабатывать возможность открытия нескольких экземпляров одного и того же визуализатора – принята схема, что только один экземпляр поддерживает открытие и закрытие сервиса (а соответственно и взаимодействие с процедурным генератором), а остальные в это время не делают никакой HTTP обработки (т.е. фактически могут быть использованы только для загрузки моделей из файловой системы). При закрытии этого управляющего экземпляра, можно будет вручную (нажатием кнопки) подключить один из оставшихся визуализаторов к сервису.

РЕАЛИЗАЦИЯ АЛГОРИТМА ГЕНЕРАЦИИ МОДЕЛЕЙ ЗДАНИЙ

С программной стороны, генерация трехмерных моделей зданий состоит из четырех этапов:

* Преобразование параметров генерации, полученных из Представления-Модели в конкретный вид для генератора (как описано в пункте \ref{ch3:module-implementation:generator});
* По параметрам создается грамматика (т.е. начальное слово, грамматические правила), слово последовательно преобразуется правилами и в итоге становится словом модели здания;
* Из слова формируется трехмерная модель здания;
* Трехмерная модель проверяется на корректность формата.

Пользуясь объектно-ориентированным подходом языка C\#, грамматика определяется как набор объектов в классе BuildingsGrammarController. Грамматическое слово представляется как граф-дерево, состоящее из узлов класса GrammarNode. Теоретически, слово может быть произвольной структуры, например, списком из грамматических символов или произвольным графом из символов, однако граф-дерево является наиболее удобным, т.к. в полной мере может представлять отношение разных символов между собой (что и заложено рассмотренной split-грамматикой -- одни символы разбиваются на несколько других, образуя иерархию), а также не является слишком сложным для реализации (так как, для реализации графа необходимо решить вопросы петель, циклов, и не является очевидным то, как данную структуру запрограммировать на визуализацию в виде модели).

Всего представлено 7 конкретных реализаций класса GrammarNode: RootNode является начальным словом процесса генерации, FloorNode обозначает этаж определенной высоты, формы и типа (нижний, средний, верхний), WallStripNode обозначает одну из стен отдельного этажа, RoofNode обозначает крышу определенного стиля и высоты, SegmentNode обозначает кусок стены (который может быть пустым или заполнен дверью, окном), DoorNode обозначает дверь, WindowNode обозначает окно. Каждый из этих грамматических <<символов>> определяет собственные параметры, которые берутся из параметров генерации (так, например, определяется количество этажей у символа RootNode, или высота каждого этажа у символа FloorNode). Помимо данных, все классы вершин определяют метод BuildOnMesh, который принимает некоторую 3D модель и изменяет её некоторым образом, который заложен конкретным символом (например, добавляет модель окна в нужной позиции и определенных размеров, или ничего не изменяет, тем самым являясь абстрактным символом). Таким образом, все символы представляются идентично -- как наследники класса GrammarNode, и поэтому смысл геометрических и абстрактных символов (которые рассмотрены детально в упомянутом параграфе) в рамках практической реализации алгоритма идентичен.

Грамматические символы порождаются с помощью применения грамматических правил (исключением является символ RootNode, который является начальным словом алгоритма и создается явно). Каждое грамматическое правило является наследником класса GrammarRule и переопределяет метод Apply, принимающий все слово целиком, и модифицирующий некоторые из его символов. Введены правила: RootSplitRule -- изменяет символ RootNode, добавляя к его потомкам несколько (в зависимости от настроек последнего) символов-этажей FloorNode (при создании каждому этажу передаются его настройки, т. е. высота, форма и пр.), правило TopFloorToRoofRule целиком заменяет символ верхнего этажа на символ крыши, FloorToWallStripRule разбивает этаж на стены в пределах этажа (с параметрами для визуализации стены -- один из углов стены, высота, ширина, нормаль к плоскости стены), WallStripToSegmentRule разделяет этот кусок стены на сегменты SegmentNode, правило SegmentToDoorsRule заменяет некоторые из сегментов на двери (в данной реализации -- просто один из сегментов на первом этаже), а правило SegmentsToWindowsRule заменяет несколько сегментов на окна (соответственно в зависимости от параметров окна могут иметь разные стили, и располагаться симметрично или произвольно).

Весь набор правил применяется целиком к слову. Глубина рекурсии, которой достигает изменение слова ограничено до запуска генерации (таким образом решается проблема бесконечной рекурсии). Результатом является итоговое грамматическое слово, являющееся иерархией символов с параметрами, которые необходимы при визуализации.

Пользуясь динамическим полиморфизмом -- способом переопределения поведения объектов в языке C#, достаточно легко построить модель. Изначально создается пустая модель, и рекурсивно обходятся все символы слова. Над каждым символом вызывается метод BuildOnMesh, который по параметрам из символа добавляет элемент в модель. Итоговая модель проверяется на корректность, используя функционал библиотеки geometry3sharp. ЗАКЛЮЧЕНИЕ

В ходе преддипломной практики, была проведена работа по практической реализации основной части проекта для выпускной квалификационной работы бакалавра. Основой реализации послужили теоретические выкладки по схеме, архитектуре системы и выбора используемых технологий, которые были рассмотрены в прошлых работах в рамках ВКР.

Были реализованы функциональные подсистемы модуля процедурной генерации трехмерных моделей зданий (с архитектурной точки зрения, а также со стороны взаимодействия с пользователем): генерации модели, экспорта модели в файл и визуализации через внешние приложения. Дополнительно, реализована утилита для визуализации полученных трехмерных моделей. Соответственно обеспечены программные интерфейсы для реализации других визуализаторов для других операционных систем или с другим функционалом и реализован графический пользовательский интерфейс для визуализации в Windows.

Генерация моделей зданий во много требует творческих подходов для построения непосредственно генерации, однако программная основа для генерации была создана. Теоретическое описание системы, которое было спроектировано до реализации проекта, достаточно удачно и гибко было реализовано на практике в виде нескольких программ, из-за чего проект в целом является успешным.

По итогу практики, была достигнута цель программной реализации разрабатываемого в ВКР модуля. Были получены знания в разработке пользовательских интерфейсов на языке C# c библиотекой WPF, разработки HTTP сервисов c библиотекой WCF, а также применения библиотек Helix Toolkit и geometry3sharp языка C#.

ПРИЛОЖЕНИЕ

Блок-схемы алгоритма генерации

