ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО

ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

**«КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

**ИНСТИТУТ ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ И ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ**

КУРСОВОЙ ПРОЕКТ

МАКЕТНАЯ ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ДИАЛОГОВОГО ТЕХНОЛОГИЧЕСКОГО КОМПЛЕКА ГЕНЕРАЦИИ СЛОЖНЫХ ПРОГРАММНЫХ СИСТЕМ.

ПРИМЕР РЕАЛИЗАЦИИ МАКЕТА НА ОСНОВЕ МОДЕЛИ ТЕОРИИ ИГР

Выполнил: студент 2 курса

гр.09-411 Губайдуллин А.В.

Проверил: старший преподаватель кафедры

технологий программирования

Георгиев В.О.

Казань

2015

Оглавление

[1. ВВЕДЕНИЕ 3](#_Toc439204086)

[1.1. ПОСТАНОВКА, ПРОБЛЕМАТИКА И ЗАДАЧИ ЗАДАНИЯ КУРСОВОЙ РАБОТЫ 3](#_Toc439204087)

[1.2. ОСНОВНЫЕ ПОНЯТИЯ И ОПРЕДЕЛЕНИЯ 3](#_Toc439204088)

[2. ФОРМАЛЬНОЕ ПРЕДСТАВЛЕНИЕ МОДЕЛИ ДИАЛОГОВОГО ИНТЕРФЕЙСА, ПОСТРОЕННОЙ НА ОСНОВЕ ГРАФОВОЙ (АВТОМАТНОЙ) МОДЕЛИ 4](#_Toc439204089)

[2.1. КОНКРЕТИЗАЦИЯ ТЕОРИИ ИГР 4](#_Toc439204090)

[2.2. РЕАЛИЗАЦИЯ МОДЕЛИ ДЛЯ ИНТЕРАКТИВНОГО ВЗАИМОДЕЙСТВИЯ В ПРЕДСТАВЛЕННОМ ДИАЛОГОВОМ ТЕХНОЛОГИЧЕСКОМ КОМПЛЕКСЕ 7](#_Toc439204091)

[3. ПОСТАНОВКА ТЕХНИЧЕСКОГО ЗАДАНИЯ ДЛЯ ПРОГРАММНОЙ РЕАЛИЗАЦИИ МОДЕЛИ 9](#_Toc439204092)

[3.1. РАЗРАБОТКА И ФУНКЦИИ КОМПЛЕКСА 9](#_Toc439204093)

[3.2. РАЗРАБОТКА СЦЕНАРИЯ ДИАЛОГА 9](#_Toc439204094)

[3.3. ИСПОЛЬЗУЕМЫЕ ИНСТРУМЕНТЫ ДЛЯ РЕАЛИЗАЦИИ 10](#_Toc439204095)

[3.4. СТАНДАРТЫ ГОСТ/ISO 12](#_Toc439204096)

[4. ПРЕДСТАВЛЕНИЕ РЕЗУЛЬТАТОВ РАБОТЫ, ПОЛУЧЕННЫХ В ХОДЕ РАЗРАБОТКЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ 13](#_Toc439204097)

[4.1. ПРИМЕР РАБОТЫ ПРОГРАММЫ 13](#_Toc439204098)

[4.2. ОСОБЕННОСТИ РАЗРАБОТКИ 15](#_Toc439204099)

[5. ЛИСТИНГ КОДА 16](#_Toc439204100)

[Question.cs 16](#_Toc439204101)

[Settings.cs 19](#_Toc439204102)

[Creator.cs 22](#_Toc439204103)

[6. ЛИТЕРАТУРА И ИСТОЧНИКИ 25](#_Toc439204104)

# ВВЕДЕНИЕ

## ПОСТАНОВКА, ПРОБЛЕМАТИКА И ЗАДАЧИ ЗАДАНИЯ КУРСОВОЙ РАБОТЫ

На сегодняшний день существует множество видов моделей для конструирования программных систем. Их использования в большинстве случаев зависит от выбранной предметной области. Одной из таких моделей является графовая (автоматная) модель. В своём курсовом проекте Я рассматривал модель теории игр, основанную на данной построения диалогового технологического комплекса генерации сложных программных систем.

Не для кого не секрет, что графовая (автоматная) модель является одной из доступных и понятных моделей для реализации программных систем. Благодаря её простоте и вариативности можно сконструировать не только отдельную систему, но и целый комплекс, который может служить базой для построения других сложных систем.

Цель курсовой работы:

1. Приобретение навыков в формировании технического задания по требованиям заказчика;
2. Самостоятельная разработка диалогового технологического комплекса создания сложных программных в соответствии с требованиями технического задания;
3. Построение генератора сценариев диалога;
4. Приобретение навыков оформления программной документации для дальнейшей эксплуатации проекта.

## ОСНОВНЫЕ ПОНЯТИЯ И ОПРЕДЕЛЕНИЯ

Формальная грамматика или просто грамматика в теории формальных языков — способ описания формального языка, то есть выделения некоторого подмножества из множества всех слов некоторого конечного алфавита. Различают порождающие и распознающие (или аналитические) грамматики — первые задают правила, с помощью которых можно построить любое слово языка, а вторые позволяют по данному слову определить, входит ли оно в язык или нет.

Конечный автомат — [абстрактный автомат](https://ru.wikipedia.org/wiki/%D0%90%D0%B1%D1%81%D1%82%D1%80%D0%B0%D0%BA%D1%82%D0%BD%D1%8B%D0%B9_%D0%B0%D0%B2%D1%82%D0%BE%D0%BC%D0%B0%D1%82), число возможных внутренних состояний которого [конечно](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BD%D0%B5%D1%87%D0%BD%D0%BE%D0%B5_%D0%BC%D0%BD%D0%BE%D0%B6%D0%B5%D1%81%D1%82%D0%B2%D0%BE).

Автомат Мили — конечный автомат, выходная последовательность которого (в отличие от автомата Мура) зависит от состояния автомата и входных сигналов. Это означает, что в графе состояний каждому ребру соответствует некоторое значение (выходной символ). В вершины графа автомата Мили записываются выходящие сигналы, а дугам графа приписывают условие перехода из одного состояния в другое, а также входящие сигналы.

Формальный язык в математической логике и информатике — множество конечных слов (строк, цепочек) над конечным алфавитом. Понятие языка чаще всего используется в теории автоматов, теории вычислимости и теории алгоритмов. Научная теория, которая имеет дело с этим объектом, называется теорией формальных языков.

Теория автоматов — раздел дискретной математики, изучающий абстрактные автоматы — вычислительные машины, представленные в виде математических моделей — и задачи, которые они могут решать.

Вложенный автомат стека - конечный автомат, который может использовать стек, содержащий данные, которые могут быть дополнительными стеками.

Граф зависимостей — ориентированный граф, отображающий соотношение элементов некоторой совокупности в соответствии с выбранным транзитивным отношением над ней.

Нагруженный граф − ориентированный граф D = (V, X), на множестве дуг которого определена некоторая функция, которую называют весовой функцией.

# ФОРМАЛЬНОЕ ПРЕДСТАВЛЕНИЕ МОДЕЛИ ДИАЛОГОВОГО ИНТЕРФЕЙСА, ПОСТРОЕННОЙ НА ОСНОВЕ ГРАФОВОЙ (АВТОМАТНОЙ) МОДЕЛИ

## КОНКРЕТИЗАЦИЯ ТЕОРИИ ИГР

**Теория диалоговых игр** — это попытка учесть идеи сценарных моделей и диалоговых грамматик в одной структуре. Предположим, что диалоги состоят из череды так называемых игр. Каждая игра составлена из последовательности ходов, которые возможны в соответствии с набором правил (похожих на грамматики), и вся игра запланирована участвующими агентами (как в модели основанной на сценариях). Таким образом, агенты совместно используют знания (представления и цели) в ходе диалога, и игры могут быть вложенными (возможны поддиалоги) для достижения подцелей. В этой структуре ходы часто приравниваются к речевым актам. Модель довольно формально определяет ходы, допустимые для каждого из участников в данный момент игры (по правилам и в соответствии с целью) и таким образом, моделируются диалоги. Эта модель доведена до реализации и используется при проектировании некоторых диалоговых систем.

Термин «диалоговая игра» впервые был предложен Веттгенстейном в 1957 году, чтобы представить язык, как взаимодействие, ограниченное правилами. Но растущий интерес к теории игр привел к появлению новых значений этого термина. В работах Грунига(1989) термин диалоговая игра ссылался не только на работы Веттгенстейна, но и на элементы теории игр. Сейчас существует два подхода к определению этого термина.

Первый подход, который был предложен Левином, Муром и Манном, вводит ДИ, как двусторонняя структура, подразумевающая действия обоих игроков для специального взаимодействия. Например, игра-помощник: нуждающийся в помощи должен объяснить проблему, помощник должен предложить решение. Ходы в игре происходят пошагово. Главный недостаток этого подхода заключается в недостаточной гибкости, т. к. он требует упорядоченных, фиксированных действий. Все дальнейшие попытки решить эту проблему не привели к реальному успеху.

Второй подход гораздо очень близок к принципам теории игр. Высказывания, или действия диалога, представляются в качестве ходов игры. Главная идея — запрещать некоторые последовательности ходов, возвращая на предыдущий ход и в предыдущее состояние.

Эти подходы не взаимоисключающие: они имеют отношения к разным проблемам. Первое решает проблему нахождения согласованного взаимодействия и учитывает глобальную структуру диалога. Вторая формализует нормы специфического взаимодействия и учитывает только локальную структуру диалога.

В данном тексте мы заинтересованы во втором подходе. Для начала отметим, что каждая модель построена на формализмах, построенных строго для специальных типов взаимодействия. Далее ответим на вопрос: существует ли единая теория игровых диалогов? Иначе говоря: несмотря на ее использования в разных ситуациях, существуют ли общие принципы этой модели?

Карлсон предложил диалоговую игру, в которой конечная цель игры четко не определена. Игры МакКензи и Беуна определяют цель в начале игры. В игре МакКензи игроки пытаются доказать свою правоту, при этом их мнения противоположны. Игра Беуна, напротив, нацелена на совместный поиск ответа на вопрос.

Первая игра — типичный представитель игры-конфликта, вторая — игры-кооперации.

В этом тексте приведены показывается, что общие принципы этих игр могут быть найдены.

Для начала определим требования к диалоговым играм, введя несколько определений.

**Игровая доска** определяется двумя ее частями: *общая доска* и *личные доски*.

*Общая доска*. Принято представлять диалог, как взаимодействие на общей доске или общей земле. Эта доска содержит упорядоченную историю ходов, идентификатор игры со связанными правилами, которые были известны до начала или стали известны в процессе диалога. Этот набор можно рассматривать как множество начальных представлений (MBA, B).

*Личная доска* — личные представления каждого игрока, неизвестные другим. Это в частности предполагает, что мы имеем дело с играми с неполной информацией. Представления о мире будем рассматривать, как множество BA. Представления о представлениях другого игрока рассмотрим, как множество BABB.

Теперь нужно определить способ, которым эти доски могут быть обновлены и изменены, и когда эти доски подвергаются одинаковым логическим ограничениям (правилам).

**Ход** — базовое действие игрока в игре, и оно имеет цену. **Шагом игры** назовем интервал между двумя ходами. Игрок, который должен ходить — **активный**.

**Когнитивный контекст** - состояние личных досок и совместно принятые факты на публичной доске.

С каждом шагом игры меняется ее когнитивный контекст из-за добавления или удаления фактов на игровой доске. В этом отношении это понятие может быть отнесено к акту диалога. Формально **акт диалога** определен как коммуникативная функция, применительная к содержательным высказываниям. Например, утверждение «Небо голубое» - это акт диалога assert(p), где p = “Небо голубое». Так же введем **пустой ход**. Этот ход не влияет на когнитивный контекст игры, но увеличивает счетчик ходов, как любое диалоговое действие.

В добавок, будем считать, что активный игрок может сделать логический ход. **Логический ход** изменяет доску, применяя логическое правило. Логический ход ничего не предлагает и не увеличивает счетчик игры. Таким образом может быть определено несколько видов логических ходов. Возможно даже представить ход, который торопит других принят логическое правило.

Цели

Даже если намерения напрямую не реализованы в игре, игроки имеют цели, когда играют в игру. Цели записаны в данной модели структурой игры. Т.к. Обмен информацией в игре ограничен, цель игрока — достичь ментального состояния, которого игрок хочет достичь. В добавок ко всему, для того, чтобы учесть возможность недостижимости целей, введем понятия ошибочного состояния. Достигая этого состояния, игроки демонстрируют их неспособность достичь цели. В этот момент игра прерывается.

Правила

Понятие «правило» сильно связано с сущностью игры. Не существует подхода, который мог бы дать чистое представление различных правил в диалоговых играх. Определим сигма как множество возможных когнитивных контекстов. Тогда начальные правила игры можно просто определить как функцию, определяющую доступные ходы. Назовем такие правила эффективными — имеющими эффект.

**Эффективные правила** (m: Ω → Ω) Эти правила влияют на когнитивный контекст. Влияние на слушателя, в таком случае, это представления о рациональности говорящего.

Далее определим M, как множество ходов игры. Это множество — набор функций типа m, которые можно отнести к множеству ходов диалога(Md) или множеству логических ходов(Mt). M = Md U Mt.

История диалога это строго упорядоченное множество h = { m1, m2, ...mn | mi=1..n= 1..n ∈ Md }. Множество всех возможных историй обозначим H. Теперь определим три новых класса правил:

*Соглашения диалога* (Fdc : H → 2Md) Эти правила определяют, какие ходы допустимы после данной истории диалога.

*Рациональность говорящего* (Fsr : 2Md ⊗ Ω → 2Md) Эти правила запрещают говорящему делать ходы, выдающие его ментальные состояния, предполагая некоторый уровень рациональности.

*Рациональность слушателя* (Fhr : ω → ω) . Эти правила требуют определенной рациональности. В действительности, слушатель должен быть не менее рационален, чем говорящий.

Шаг в игре

Очевидно, не все эти правила работают одновременно.

Пусть h1 – это история диалога (где m1 – последний ход истории), а ω1 – когнитивный контекст в ходе 1. δi — это набор ходов диалога и fdc, fsr, fhr – это функции, определяющие соглашения диалога, рациональность слушателя и говорящего.

1. δ1 = fdc (h1) — множество допустимых по соглашению ходов после h1.
2. ω1' = m1(ω1) – когнитивный контекст после h1.
3. ω1'' = fhr(ω1') – когнитивный контекст после обновления ментального состояния слушателя.
4. δ1' = fsr(δ1, ω1'') – множество рациональных и допустимых по соглашению ходов диалога активного игрока.

Тогда ход ограничен двумя факторами: правилами и текущим когнитивным контекстом. В этом отношении мы требуем, чтобы наша модель была одновременно основана на правилах и ограничениях. Из этого следует, что в одной игре, два одинаковых хода игрока в разные моменты игры могут дать разный δ1' набор возможных ходов.

В итоге получим, что диалоговая игра между A и B – это структура:

DG(A, B, *game*) = (*S, R*, **Ф**). **Ф** — это множество допустимых типов ходов в игре. *S* состоит из (*SA, SB, SF*), где *SA* — это состояние, которое A хочет достигнуть, *SB* – состояние, которое хочет достигнуть B, *SF* – ошибочное состояние. R состоит из (*RA, RB*), где RA – множество правил для A, RB — множество правил для B. Эти множества функционально представимы через (*Fef, fdc, fsr, fhr*), где *Fef* – это множество ходов функционально связанных с элементами δ1, и fdc, fsr, fhr – функции, отражающие соглашения диалога, рациональность говорящего и слушателя. Теперь можем дать два определения:

*Симметрия*. Игры, в которых правила для обоих игроков одинаковы, называются симметричными.

*Кооперация*. Игры, в которых оба игрока пытаются достигнуть одинакового состояния, называются кооперативными.

Важно понимать, что эти понятия не взаимоисключающие.

На основе введенных определений приведем пример простой диалоговой игры.

Соглашения диалога:

*dc(⟨-, assert(p)⟩)* = *{accept(p),deny + (p), empty, question + (x)}*

*dc(⟨-, accept(p)⟩)* = *{assert(x), empty, question + (x)}*

else *dc(⟨-⟩) = Md*

Эффекты:

*assertA*(*p*)(*ω0*) = *ω0* U { *BBBAp, BABBp* }

*acceptA*(*p*)(*ω0*) = *ω0* U { *BBBA¬p* } – { *BBBAp* }

*questionA*(*p*)(*ω0*) = *ω0* U { *BBBA \* p* }

*ignoreA*(*p*)(*ω0*) = *ω0* U { *BBBA \* p* }

Рациональность говорящего:

*sr*(*δ0, ω0*) = *δ0*– *δ1* где *δ1* — конструкция:

if *BAp* ∉ *ω0* then *δ1*= *δ1* U { *assert*(*p*) }

if *BAp* ∈ *ω0* or *BA*¬*p* ∈ *ω0* then *δ1* = *δ1* U *{ question*(*p*)*, ignore*(*p*) *}*

if *BA*¬*p* ∉ *ω0*then *δ1* = *δ1* U { *deny*(*p*) }

Рациональность слушателя:

if *BABBp* ∈ *ω0* and *BA*¬*p* ∉ *ω0* then *hr*(*ω0*) = *ω0* U { *BAp* }

else hr(ω0) = ω0

## РЕАЛИЗАЦИЯ МОДЕЛИ ДЛЯ ИНТЕРАКТИВНОГО ВЗАИМОДЕЙСТВИЯ В ПРЕДСТАВЛЕННОМ ДИАЛОГОВОМ ТЕХНОЛОГИЧЕСКОМ КОМПЛЕКСЕ

Структуру комплекса составляют две части: реализация модели с заданными свойствами с помощью, так называемого, генератора сценария и применение построенного сценария в диалоговой системе.

В качестве прототипной практической реализации графовой (автоматной) модели для обеспечения интерактивного взаимодействия в макетном варианте выше представленная модель будет выглядеть следующим образом:

- Граф задается как пара {Nodes, Arrows}, что является стандартным представлением вершин графа и возможных переходов в следующие состояния;

- Каждой вершине соответствует вопрос диалога;

- Каждому переходу соответствует один из возможных вариантов ответов на вопрос, что по сути предоставляет нам возможность перемещаться в любые доступные состояния;

- Каждому вопросу, аналогично вершине графа, присваивается некоторое состояние.

Таким образом была реализована графовой (автоматной) модель с помощью инструментов определенной среды разработки под названием Microsoft Visual Studio 2015.

# ПОСТАНОВКА ТЕХНИЧЕСКОГО ЗАДАНИЯ ДЛЯ ПРОГРАММНОЙ РЕАЛИЗАЦИИ МОДЕЛИ

## РАЗРАБОТКА И ФУНКЦИИ КОМПЛЕКСА

Создание диалогового технологического комплекса для генерации сложных программных систем является одной из перспективных направлений в данной сфере. Поэтому разработка и усовершенствование такого вида работы позволяет задать вектор направления в развитии проекта.

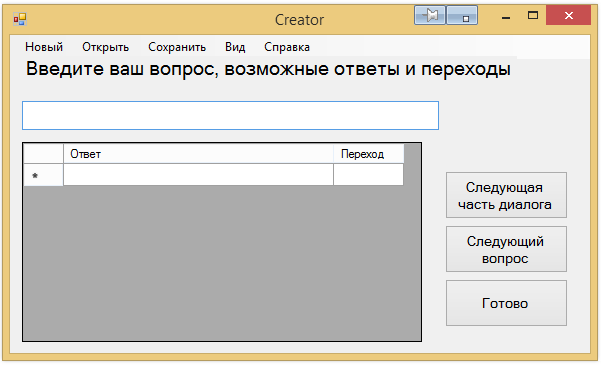
В качестве темы своей курсовой работы я выбрал реализацию модели теории игр, основанной на графовой (автоматной) модели. Но это лишь небольшая часть проекта. Помимо этого мной также был разработан и сконструирован диалоговый технологический комплекс генерации сложных программных систем.

Программа позволяет:

* Сгенерировать диалог на основе графовой (автоматной) модели;
* Отредактировать диалог, добавить в диалог новые вопросы с вариантами ответов, просмотреть записи;
* Использовать уже имеющийся у пользователя сценарий диалога;
* На примере человеко-машинного взаимодействия ответить на вопросы диалога с записью отчётов по каждому из пунктов в файл;

Программа написана на языке C# в среде программирования Microsoft Visual Studio 2015 Professional.

## РАЗРАБОТКА СЦЕНАРИЯ ДИАЛОГА



Создание сценария, построенного на основе графовой (автоматной) модели начинается с создания пустого документа в формате .kur

Для записи состояния (вершины) графа используется инструмент textbox, в который записывается текст вопроса.

Граф реализован таблицей (DataGridView) с полями «Варианты ответов», «Переход в состояние».

После заполнения формы необходимо нажать кнопку «Следующий вопрос». Путем таких последовательных действий с указанием номера перехода на определенное состояние для каждого варианта ответа на вопрос можно сформировать файл, в котором хранится сценарий, построенный на основе нашей модели.

После выполнения этих действий мы можем открыть наш созданный файл в главном окне программе. После таких манипуляций пользователю предлагается пройтись по сценарию диалога и ответить на вопросы. Исходя из ответа пользователя, система переходит в необходимое нам состояние и записывает результат в «Ход диалога».

Как говорилось выше, комплекс построен таким образом, что на его основе можно реализовать любую модель.

В ходе диалога присутствует анализатор ошибок, который выводит в поле «Сообщение об ошибке» информацию пользователю о том, что он ввёл неверный ответ и система вновь просит его ответить на вопрос, выбрав один из предложенных вариантов ответа.

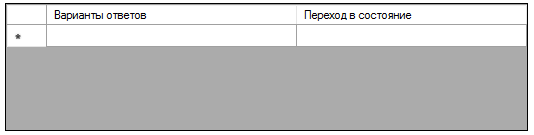
В программе предусмотрен инструмент «Выпадающее меню», с помощью которого пользователь может работать в программе.

## ИСПОЛЬЗУЕМЫЕ ИНСТРУМЕНТЫ ДЛЯ РЕАЛИЗАЦИИ

TextBox – позволяет вводить информацию в поле.



DataGridView – представляет собой таблицу со строками и столбцами.



Buttons – кнопки для управления записями в таблице.



Label – надпись для именования различных объектов.



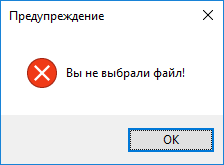
MenuStrip – выпадающее меню с различными вариантами.



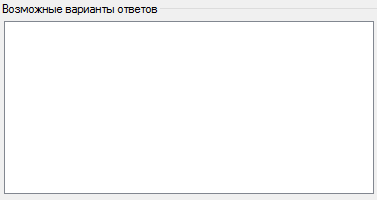
GroupBox – группирует объекты в единую систему.



MessageBox – окно с сообщением (информацией).



ListBox – используется для записи информации построчно.



## СТАНДАРТЫ ГОСТ/ISO

Административное управление определяется:

ISO 12207:1995 – процессы жизненного цикла программных средств

ГОСТ 19.102-77 – стадии разработки ПС.

ISO 16326:1999 – руководство по применению (207 ИСО) при административном управлении проектом.

Тестирование определяется:

ANSI/IEEE 1008-1986 – тестирование программных модулей и компонентов ПС.

ANSI/IEEE 1012-1986 – планирование верификации и подтверждение достоверности качества (валидации ПС).

Документирование определяется:

ГОСТ 19105-78 (общее требование к программным документам)

ГОСТ 19106-78 (требование к программным документам, выполненным печатным способом)

ISO 9294:1990 (Руководство по управлению документированию ПО)

ISO 9127:1990.

Оценка качества:

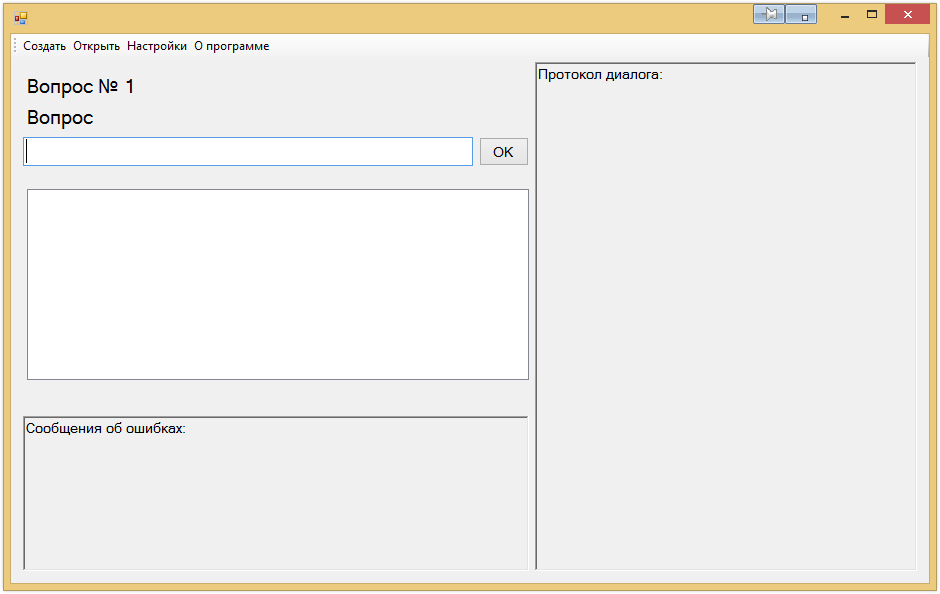
ISO 9126:1991 Оценка программного продукта характеристики качества и руководства по их применению.

ISO 14598-1-6:1998-2000 Оценивание программного продукта.

# ПРЕДСТАВЛЕНИЕ РЕЗУЛЬТАТОВ РАБОТЫ, ПОЛУЧЕННЫХ В ХОДЕ РАЗРАБОТКЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

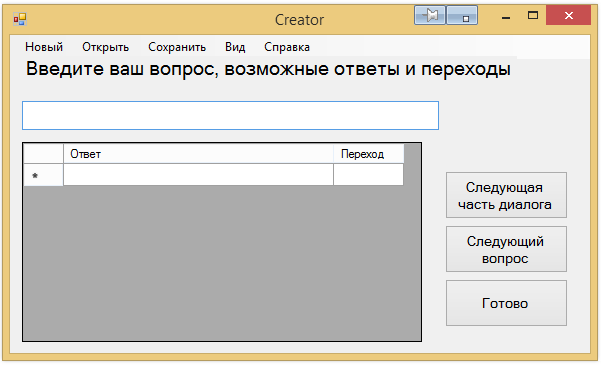
### ПРИМЕР РАБОТЫ ПРОГРАММЫ

Начальное окно программы

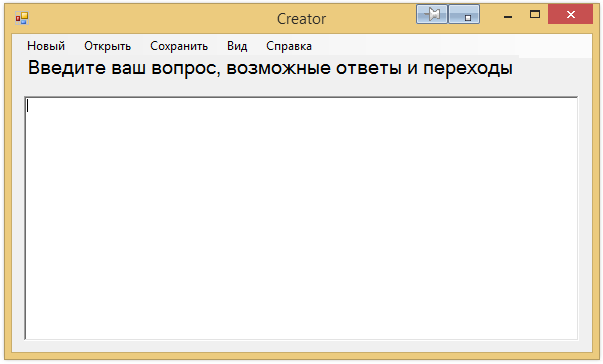


При нажатии кнопки «Создать» мы переходим в режим создания нового сценария диалога. Эта форма имеет 2 вида:

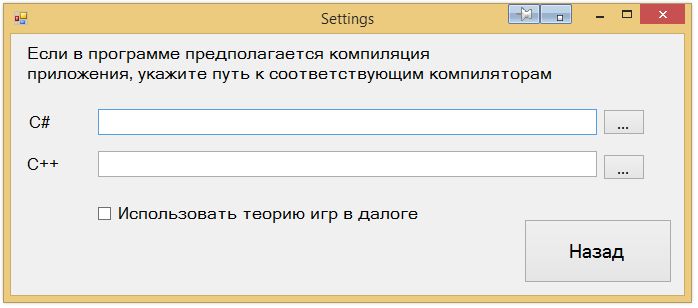
* 1. Простой – в нем можно писать простые диалоги без дальнейшей сборки



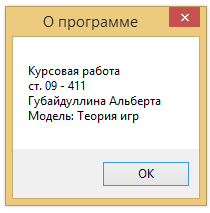
* 1. Расширенный – в нем можно писать сложные диалоги с фрагментами кода на том или ином языке



При нажатии кнопки «Настройки» мы переходим в новую форму, где можно задать местонахождение предустановленных компиляторов представленных языков, а также указать программе необходимо ли вводить Элемент теории игр в диалоге.



Также при нажатии кнопки «О программе» главного окна появляется элемент MesssageBox со следующим содержанием



### ОСОБЕННОСТИ РАЗРАБОТКИ

Структура основных классов:

Класс, в котором хранится сценарий

public class Scenario

{

public int PartsNUM = 0;

public Element[] S;// S = (s1,s2,…,sn) si – элемент сценария.

public class Element // s = (M, P, N, T, R, S’)

{

public int QuestionsNUM = 0;

public Message[] M; //M = (Mk, V, D) – описание сообщения, которым обменивается юзер с ДС.

public Procedures P; //P определяет запускаемые прикладные процедуры пользователя.

public ProgramInfo Prog = new ProgramInfo();

public Scheme T;// T - схемы выбора последующих шагов диалога.

public class Message

{

public int AnswersNUM = 0;

public Format Mk; //Mk = (mk1, mk2,…,mkn) – макета(кадры) экрана, определяющие внешний формат сообщения.mki – формат сообщения заданный в текущий момент заданного интерактивного взаимодействия.

public string V; //V = (v1,v2,…,vn) – вопросы диалога.vi – вопрос, заданный пользователю в текущий момент хода ДВ.

public string D; //D = (d1, d2,…, dn) -справочная информация, позволяющая пользователю в соответствующем пункте диалога получить справку о состоянии диалога, о хар-ках текущего пункта, возможные варианты ответов, значения по умолчанию.

public Answers[] R;//R = (sh1, sh2, …, shn) – диапазон допустимых ответов пользователя, который определяется использованием синтаксиса семантического шаблона(код) АЙЧ

public class Format

{

}

public class Answers

{

public String Answer;//R = (sh1, sh2, …, shn) – диапазон допустимых ответов пользователя, который определяется использованием синтаксиса семантического шаблона(код) АЙЧ

public int Next; //Next = (n1, n2,…, nm) – список положительных чисел mi, определяющий последующий шаг диалога.

public Command[] Source;

public class Command

{

public bool Exist;

public String Language;

public String Type;

public int[] Length = new int[2];

public String[] Text = new String[2];

}

}

}

public class Procedures

{

}

public class Scheme

{

}

}

public class ProgramInfo

{

public static int LanguageNUM;

public Pair Language;

public int Dir;

public Pair Name;

public Pair IN;

public Pair OUT;

// public string[] Utility = new string[5];

public Pair Task;

}

Класс, в котором хранится программы, созданная в процессе диалога с пользователем

public class Program

{

public string Language;

public string Name;

public string Start;

public string End;

public string Dir;

public string[] IN = new string[2];

public string[] OUT = new string[2];

// public string[] Utility = new string[5];

public string[] Task;

# ЛИСТИНГ КОДА

## Question.cs

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.IO;

namespace Kursovaia\_Gubaidullin

{

public partial class Creator : Form

{

public int i = 0, k = 0;

public bool NEW = true;

public bool CanSimpleViewed = true;

Form ParentForm;

int PartsNUM = 0;

int QuestionNUM = 0;

int CurrentLine = 2;

public Creator(Form f)

{

InitializeComponent();

ParentForm = f;

SourceBox.Text += '\n' + '\n';

}

private void SimpleView\_Click(object sender, EventArgs e)//сделано

{

if (NEW)

{

SourceBox.Visible = false;

}

else

{

if (CanSimpleViewed)

{

SourceBox.Visible = false;

}

else

MessageBox.Show("Данный сценарий не может быть представлен в простом виде");

}

}

private void SourceView\_Click(object sender, EventArgs e)//сделано

{

SourceBox.Visible = true;

}

private void NextPart\_Click(object sender, EventArgs e)

{

PartsNUM++;

SourceBox.Lines[0] = Convert.ToString(PartsNUM);

SourceBox.Lines[CurrentLine - 1] = Convert.ToString(QuestionNUM);

QuestionNUM = 0;

SourceBox.Text += QuestionBox.Text + '\n'; // Вопрос

CurrentLine++;

SourceBox.Text += Convert.ToString(SourceGrid.RowCount) + '\n'; //Количество Всеможножных ответов

CurrentLine++;

for (int z = 0; z < SourceGrid.RowCount; z++)

{

SourceBox.Text += Convert.ToString(SourceGrid.Rows[z].Cells[0].Value) + '\n'; //Ответ

CurrentLine++;

SourceBox.Text += Convert.ToString(SourceGrid.Rows[z].Cells[0].Value) + '\n'; //Переход

CurrentLine++;

SourceBox.Text += "false" + '\n'; // Exist-ы кодов

CurrentLine++;

SourceBox.Text += "false" + '\n';

CurrentLine++;

}

QuestionBox.Clear();

SourceGrid.Rows.Clear();

}

private void NextQuestion\_Click(object sender, EventArgs e)//сделано

{

QuestionNUM++;

SourceBox.Text += QuestionBox.Text + '\n'; // Вопрос

CurrentLine++;

SourceBox.Text += Convert.ToString(SourceGrid.RowCount) + '\n'; //Количество Всеможножных ответов

CurrentLine++;

for (int z = 0; z < SourceGrid.RowCount; z++)

{

SourceBox.Text += Convert.ToString(SourceGrid.Rows[z].Cells[0].Value) + '\n'; //Ответ

CurrentLine++;

SourceBox.Text += Convert.ToString(SourceGrid.Rows[z].Cells[0].Value) + '\n'; //Переход

CurrentLine++;

SourceBox.Text += "false" + '\n'; // Exist-ы кодов

CurrentLine++;

SourceBox.Text += "false" + '\n';

CurrentLine++;

}

QuestionBox.Clear();

SourceGrid.Rows.Clear();

}

private void NewDialog\_Click(object sender, EventArgs e)//сделано

{

SourceBox.Clear();

QuestionBox.Clear();

SourceGrid.Rows.Clear();

NEW = true;

}

private void EditDialog\_Click(object sender, EventArgs e)//сделано

{

try

{

OpenFileDialog openFileDialog1 = new OpenFileDialog();

openFileDialog1.Filter = "KUR|\*.kur";

openFileDialog1.Title = "Select a kur File";

if (openFileDialog1.ShowDialog() == System.Windows.Forms.DialogResult.OK)

{

string path = openFileDialog1.FileName;

System.IO.StreamReader file = new System.IO.StreamReader(path, Encoding.Default);

SourceBox.Text = file.ReadToEnd();

file.Close();

file = new System.IO.StreamReader(path, Encoding.Default);

string s = null;

for (int i = 0; i < 7; i++)

s = file.ReadLine();

CanSimpleViewed = Convert.ToBoolean(s);

file.Close();

NEW = false;

}

}

catch (Exception ex)

{

MessageBox.Show("Error: Could not read file from disk. Original error: " + ex.Message);

}

}

private void Help\_Click(object sender, EventArgs e)

{

MessageBox.Show("Правила заполнения см. в Документации к проекту");

}

private void OK\_Click(object sender, EventArgs e)//сделано

{

{

Stream myStream;

SaveFileDialog saveFileDialog1 = new SaveFileDialog();

saveFileDialog1.Filter = "KUR|\*.kur";

saveFileDialog1.FilterIndex = 2;

saveFileDialog1.RestoreDirectory = true;

if (saveFileDialog1.ShowDialog() == DialogResult.OK)

{

if ((myStream = saveFileDialog1.OpenFile()) != null)

{

StreamWriter w = new StreamWriter(myStream);

w.Write(SourceBox.Text);

myStream.Close();

}

}

ParentForm.Show();

this.Close();

}

}

}

}

## Settings.cs

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.IO;

namespace Kursovaia\_Gubaidullin

{

public partial class Creator : Form

{

public int i = 0, k = 0;

public bool NEW = true;

public bool CanSimpleViewed = true;

Form ParentForm;

int PartsNUM = 0;

int QuestionNUM = 0;

int CurrentLine = 2;

public Creator(Form f)

{

InitializeComponent();

ParentForm = f;

SourceBox.Text += '\n' + '\n';

}

private void SimpleView\_Click(object sender, EventArgs e)//сделано

{

if (NEW)

{

SourceBox.Visible = false;

}

else

{

if (CanSimpleViewed)

{

SourceBox.Visible = false;

}

else

MessageBox.Show("Данный сценарий не может быть представлен в простом виде");

}

}

private void SourceView\_Click(object sender, EventArgs e)//сделано

{

SourceBox.Visible = true;

}

private void NextPart\_Click(object sender, EventArgs e)

{

PartsNUM++;

SourceBox.Lines[0] = Convert.ToString(PartsNUM);

SourceBox.Lines[CurrentLine - 1] = Convert.ToString(QuestionNUM);

QuestionNUM = 0;

SourceBox.Text += QuestionBox.Text + '\n'; // Вопрос

CurrentLine++;

SourceBox.Text += Convert.ToString(SourceGrid.RowCount) + '\n'; //Количество Всеможножных ответов

CurrentLine++;

for (int z = 0; z < SourceGrid.RowCount; z++)

{

SourceBox.Text += Convert.ToString(SourceGrid.Rows[z].Cells[0].Value) + '\n'; //Ответ

CurrentLine++;

SourceBox.Text += Convert.ToString(SourceGrid.Rows[z].Cells[0].Value) + '\n'; //Переход

CurrentLine++;

SourceBox.Text += "false" + '\n'; // Exist-ы кодов

CurrentLine++;

SourceBox.Text += "false" + '\n';

CurrentLine++;

}

QuestionBox.Clear();

SourceGrid.Rows.Clear();

}

private void NextQuestion\_Click(object sender, EventArgs e)//сделано

{

QuestionNUM++;

SourceBox.Text += QuestionBox.Text + '\n'; // Вопрос

CurrentLine++;

SourceBox.Text += Convert.ToString(SourceGrid.RowCount) + '\n'; //Количество Всеможножных ответов

CurrentLine++;

for (int z = 0; z < SourceGrid.RowCount; z++)

{

SourceBox.Text += Convert.ToString(SourceGrid.Rows[z].Cells[0].Value) + '\n'; //Ответ

CurrentLine++;

SourceBox.Text += Convert.ToString(SourceGrid.Rows[z].Cells[0].Value) + '\n'; //Переход

CurrentLine++;

SourceBox.Text += "false" + '\n'; // Exist-ы кодов

CurrentLine++;

SourceBox.Text += "false" + '\n';

CurrentLine++;

}

QuestionBox.Clear();

SourceGrid.Rows.Clear();

}

private void NewDialog\_Click(object sender, EventArgs e)//сделано

{

SourceBox.Clear();

QuestionBox.Clear();

SourceGrid.Rows.Clear();

NEW = true;

}

private void EditDialog\_Click(object sender, EventArgs e)//сделано

{

try

{

OpenFileDialog openFileDialog1 = new OpenFileDialog();

openFileDialog1.Filter = "KUR|\*.kur";

openFileDialog1.Title = "Select a kur File";

if (openFileDialog1.ShowDialog() == System.Windows.Forms.DialogResult.OK)

{

string path = openFileDialog1.FileName;

System.IO.StreamReader file = new System.IO.StreamReader(path, Encoding.Default);

SourceBox.Text = file.ReadToEnd();

file.Close();

file = new System.IO.StreamReader(path, Encoding.Default);

string s = null;

for (int i = 0; i < 7; i++)

s = file.ReadLine();

CanSimpleViewed = Convert.ToBoolean(s);

file.Close();

NEW = false;

}

}

catch (Exception ex)

{

MessageBox.Show("Error: Could not read file from disk. Original error: " + ex.Message);

}

}

private void Help\_Click(object sender, EventArgs e)

{

MessageBox.Show("Правила заполнения см. в Документации к проекту");

}

private void OK\_Click(object sender, EventArgs e)//сделано

{

{

Stream myStream;

SaveFileDialog saveFileDialog1 = new SaveFileDialog();

saveFileDialog1.Filter = "KUR|\*.kur";

saveFileDialog1.FilterIndex = 2;

saveFileDialog1.RestoreDirectory = true;

if (saveFileDialog1.ShowDialog() == DialogResult.OK)

{

if ((myStream = saveFileDialog1.OpenFile()) != null)

{

StreamWriter w = new StreamWriter(myStream);

w.Write(SourceBox.Text);

myStream.Close();

}

}

ParentForm.Show();

this.Close();

}

}

}

}

## Creator.cs

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.IO;

namespace Kursovaia\_Gubaidullin

{

public partial class Creator : Form

{

public int i = 0, k = 0;

public bool NEW = true;

public bool CanSimpleViewed = true;

Form ParentForm;

int PartsNUM = 0;

int QuestionNUM = 0;

int CurrentLine = 2;

public Creator(Form f)

{

InitializeComponent();

ParentForm = f;

SourceBox.Text += '\n' + '\n';

}

private void SimpleView\_Click(object sender, EventArgs e)//сделано

{

if (NEW)

{

SourceBox.Visible = false;

}

else

{

if (CanSimpleViewed)

{

SourceBox.Visible = false;

}

else

MessageBox.Show("Данный сценарий не может быть представлен в простом виде");

}

}

private void SourceView\_Click(object sender, EventArgs e)//сделано

{

SourceBox.Visible = true;

}

private void NextPart\_Click(object sender, EventArgs e)

{

PartsNUM++;

SourceBox.Lines[0] = Convert.ToString(PartsNUM);

SourceBox.Lines[CurrentLine - 1] = Convert.ToString(QuestionNUM);

QuestionNUM = 0;

SourceBox.Text += QuestionBox.Text + '\n'; // Вопрос

CurrentLine++;

SourceBox.Text += Convert.ToString(SourceGrid.RowCount) + '\n'; //Количество Всеможножных ответов

CurrentLine++;

for (int z = 0; z < SourceGrid.RowCount; z++)

{

SourceBox.Text += Convert.ToString(SourceGrid.Rows[z].Cells[0].Value) + '\n'; //Ответ

CurrentLine++;

SourceBox.Text += Convert.ToString(SourceGrid.Rows[z].Cells[0].Value) + '\n'; //Переход

CurrentLine++;

SourceBox.Text += "false" + '\n'; // Exist-ы кодов

CurrentLine++;

SourceBox.Text += "false" + '\n';

CurrentLine++;

}

QuestionBox.Clear();

SourceGrid.Rows.Clear();

}

private void NextQuestion\_Click(object sender, EventArgs e)//сделано

{

QuestionNUM++;

SourceBox.Text += QuestionBox.Text + '\n'; // Вопрос

CurrentLine++;

SourceBox.Text += Convert.ToString(SourceGrid.RowCount) + '\n'; //Количество Всеможножных ответов

CurrentLine++;

for (int z = 0; z < SourceGrid.RowCount; z++)

{

SourceBox.Text += Convert.ToString(SourceGrid.Rows[z].Cells[0].Value) + '\n'; //Ответ

CurrentLine++;

SourceBox.Text += Convert.ToString(SourceGrid.Rows[z].Cells[0].Value) + '\n'; //Переход

CurrentLine++;

SourceBox.Text += "false" + '\n'; // Exist-ы кодов

CurrentLine++;

SourceBox.Text += "false" + '\n';

CurrentLine++;

}

QuestionBox.Clear();

SourceGrid.Rows.Clear();

}

private void NewDialog\_Click(object sender, EventArgs e)//сделано

{

SourceBox.Clear();

QuestionBox.Clear();

SourceGrid.Rows.Clear();

NEW = true;

}

private void EditDialog\_Click(object sender, EventArgs e)//сделано

{

try

{

OpenFileDialog openFileDialog1 = new OpenFileDialog();

openFileDialog1.Filter = "KUR|\*.kur";

openFileDialog1.Title = "Select a kur File";

if (openFileDialog1.ShowDialog() == System.Windows.Forms.DialogResult.OK)

{

string path = openFileDialog1.FileName;

System.IO.StreamReader file = new System.IO.StreamReader(path, Encoding.Default);

SourceBox.Text = file.ReadToEnd();

file.Close();

file = new System.IO.StreamReader(path, Encoding.Default);

string s = null;

for (int i = 0; i < 7; i++)

s = file.ReadLine();

CanSimpleViewed = Convert.ToBoolean(s);

file.Close();

NEW = false;

}

}

catch (Exception ex)

{

MessageBox.Show("Error: Could not read file from disk. Original error: " + ex.Message);

}

}

private void Help\_Click(object sender, EventArgs e)

{

MessageBox.Show("Правила заполнения см. в Документации к проекту");

}

private void OK\_Click(object sender, EventArgs e)//сделано

{

{

Stream myStream;

SaveFileDialog saveFileDialog1 = new SaveFileDialog();

saveFileDialog1.Filter = "KUR|\*.kur";

saveFileDialog1.FilterIndex = 2;

saveFileDialog1.RestoreDirectory = true;

if (saveFileDialog1.ShowDialog() == DialogResult.OK)

{

if ((myStream = saveFileDialog1.OpenFile()) != null)

{

StreamWriter w = new StreamWriter(myStream);

w.Write(SourceBox.Text);

myStream.Close();

}

}

ParentForm.Show();

this.Close();

}

}

}

}

# ЛИТЕРАТУРА И ИСТОЧНИКИ

* 1. https://ru.wikipedia.org/wikiТеория\_графов
  2. Орлов С.А., Цилькер Б.Я. Технологии разработки программного обеспечения
  3. Уокер Ройс. Управление проектами по созданию программного обеспечения
  4. Поспелов Д.А. Диалоговые системы в АСУ