Санкт-Петербургский государственный университет
Математико-механический факультет

Кафедра системного программирования

Визуальный язык для платформы Ubiq Mobile в среде QReal

Курсовая работа студентки 445 группы
Дерипаска Анны Олеговны

Научный руководитель: Литвинов Ю. В.,
старший преподаватель кафедры СП СПбГУ

Санкт-Петербург
2013

Оглавление

Оглавление 2
Введение 3
Визуальное моделирование 3
Разработка мобильных приложений 3
1) Постановка задачи 6
2) Обзор существующих решений 7
2.1) Визуальные среды программирования 7
2.1.1) ДРАКОН 7
2.1.2) HiAsm 8
2.1.3) Scratch 10
2.1.4) App Inventor 12
2.1.5) Stencyl 14
2.1.6) GameSalad Creator 15
2.1.7) DragonRAD 16
2.1.8) Smartface Platform 17
2.1.9) ViziApps 18
2.2) Используемые технологии 19
2.2.1) Платформа Ubiq Mobile 19
2.2.2) Среда визуального программирования QReal 20
3) Реализация 21
3.1) Вариант А. Н. Терехова 21
3.2) Собственный вариант 23
3.3) Генерация кода 30
4) Апробация 32
Заключение 35
Список литературы 36

Введение

Визуальное моделирование

На сегодняшний день разработка программного обеспечения для сложных систем является трудоемкой задачей. Поэтому важную роль играют исследования различных методов повышения производительности труда программистов за счет упрощения процесса разработки. Одним из таких способов может служить визуальное моделирование. Данный подход заключается в том, что программа описывается в виде набора визуальных моделей при помощи некоторого визуального, а не текстового языка программирования, что повышает наглядность и понятность создаваемой программы.

Для удобной работы с такими визуальными моделями существуют специальные среды разработки, а именно CASE-системы, включающие в себя один или несколько редакторов визуальных языков, средства генерации целевого кода по описанным диаграммам и другие дополнительные инструменты разработки ПО. В качестве реализуемых CASE-системой языков выступают обычно или визуальные языки общего назначения, или предметно-ориентированные визуальные языки. Языки общего назначения по определению являются универсальными, тем самым могут быть применены для решения почти любой задачи из различных предметных областей. Однако такие языки довольно сложны и громоздки, что в свою очередь усложняет процесс написания требуемых программ. Поэтому чуть позже появились визуальные предметно-ориентированные языки в рамках DSM - подхода (Domain Specific Modeling, Предметно-ориентированное моделирование), который в последнее время стал набирать популярность. Суть данного подхода заключается в том, что визуальные языки и соответствующие редакторы для них создаются под конкретную задачу или предметную область, в рамках которой были поставлены задачи. Такой подход к разработке ПО значительно упрощает процесс решения требуемых задач и тем самым даёт существенный выигрыш в производительности труда программистов, что подтверждается некоторыми существующими исследованиями, показывающими прирост производительности более чем в 3 раза по сравнению с использованием языков общего назначения [1, 2, 3]. Однако отметим, что создание вручную своего графического редактора некоторого визуального языка чаще всего является трудоемким процессом, а тем самым и реализация новой CASE-системы сама по себе может оказаться сложной задачей. А для разработки программной системы каждый раз необходимо сначала создавать свою узкоспециализированную CASE-систему или переиспользовать уже существующую, а затем только решать требуемую задачу. Тем самым рассматриваемый подход в данном виде реализации может не всегда оказаться экономически оправданным.

Поэтому впоследствии возникла идея автоматизировать процесс разработки своей CASE-системы. Тем самым появились metaCASE-системы, как результат применения подхода визуального моделирования к самим же CASE-системам. Такие системы позволяют удобно и быстро разрабатывать новые визуальные языки, в частности предметно-ориентированные, а также дополнительные инструменты их поддержки. Для этого программисту необходимо сначала описать синтаксис создаваемого языка при помощи специального формального языка (метаязыка) в соответствующем редакторе (метаредакторе). Отметим, что метаязык позволяет описывать язык на уровне абстракций предметной области. Потом можно автоматически генерировать по данному описанию редактор создаваемого визуального языка. При этом также некоторые системы поддерживают различные дополнительные возможности для более удобной работы с новыми визуальными языками. Например, это может быть задание семантики интерпретации языка, что позволит впоследствии отлаживать программы на данном языке, или задание формальным образом ограничений на язык, что позволит минимизировать написание некорректных программ с точки зрения семантики. Тем самым, при помощи metaCASE – систем можно без особых затрат создавать свои собственные редакторы визуальных языков, а значит уже такой подход к разработке ПО не только повышает производительность труда программистов, но и является экономически оправданным.

Разработка мобильных приложений

На сегодняшний день существует множество различных сред визуального программирования, содержащих в себе в качестве реализуемого языка как языки общего назначения, так и предметно-ориентированные (DSL, Domain-Specific Language). Некоторые из них будут рассмотрены ниже, в пункте «2) Обзор существующих решений / 2.1) Визуальные среды программирования». При помощи визуальных

языков общего назначения можно разрабатывать программы различной сложности и из различных предметных областей, что соответственно усложнят и сам процесс разработки. При этом, при решении конкретных задач из определенной области чаще всего не нужна такая громоздкость визуального языка, на котором она будет решаться, а достаточны только языковые конструкции, применимые именно в данной области. Поэтому наиболее популярно использование визуальных сред, поддерживающих предметно-ориентированные языки, которые по определению позволяют разрабатывать программные системы в конкретной области. Тем самым такие системы визуального моделирования являются небольшими и не очень сложными в использовании, по сравнению со средами, реализующие языки общего назначения, что увеличивает эффективность решения задач в конкретной области. Однако на данный момент большинство таких сред позволяют создавать только приложения с невысоким уровнем сложности. Поэтому одной из актуальных задач является создание предметно-ориентированных языков, предназначенных для разработки более сложных программ, например, программ с нетривиальной логикой поведения.

Одной из таких предметных областей, где нужна возможность задания нетривиальной логики поведения, является разработка ПО под мобильные устройства, в частности создание мобильных приложений. На сегодняшний день довольно популярны различные мобильные устройства, такие как обычные мобильные телефоны, смартфоны, планшеты, карманные персональные компьютеры и т.д. Поэтому решение задач в данной области — это одна из наиболее актуальных сегодня проблем. Данная область программирования не новая и уже давно изучаемая, однако, она всё еще непростая и требует специализированных на мобильных приложениях программистов. Особенно сильно усложняется процесс разработки, если требуется создать не просто обычное мобильное приложение, а приложение со сложной логикой поведения, структурами данных, способами их обработки и т.д., что более типично для реального программирования. Поэтому и появилась идея как-то упростить процесс разработки подобных приложений, обладающих нетривиальной логикой.

Одним из таких упрощений, помимо классического визуального программирования, может служить разработка приложений при помощи использования предметно-ориентированных библиотек. Данные библиотеки реализованы и предполагают использование на некотором существующем текстовом языке программирования, например C++, C#, Java и т.п. При этом они предназначены для решения задач именно в конкретной области, в данном случае в области мобильных приложений. Тем самым такие предметно-ориентированные библиотеки скрывают некоторые сложные или специфичные структуры при разработке мобильных приложений, позволяя создавать программы с довольно сложной логикой и поведением, не требуя от программиста специальных знаний и опыта в классической разработке мобильных приложений. Таким образом гораздо большее количество программистов сможет разработать качественное мобильное приложение. Одним из примеров такого подхода является платформа программирования Ubiq Mobile [9], предоставляющая программисту набор предметно-ориентированных библиотек для создания нетривиальных мобильных приложений. Более подробно данная платформа будет рассмотрена далее, в разделе «2) Существующие решения / 2.2) Используемые технологии».

Однако при таких способах упрощения от разработчика мобильного приложения всё равно требуются определённые специфичные знания программирования, такие как классы, наследование и т.п. Появилась идея упростить этот процесс настолько, чтобы при создании таких приложений, помимо прочего, еще не требовались специальные знания программирования, т.е. чтобы почти любой человек мог бы разрабатывать свои мобильные приложения, вне зависимости от его сферы деятельности. Одним из способов достичь этой цели, или по крайней мере приблизиться к ней, является объединение идей визуального программирования и предметно-ориентированных текстовых библиотек. Для данного подхода необходимо наличие некоторого визуального предметно-ориентированного языка, предназначенного для разработки мобильных приложений (как и с простой логикой, так и со сложной) под платформу, содержащую некоторый набор предметно-ориентированных библиотек. Например, можно придумать визуальный язык для платформы Ubiq Mobile, что позволит еще больше упростить процесс разработки мобильных систем, при этом расширяя круг людей, способных самостоятельно создавать свои такие приложения.

Первый вариант такого визуального языка для платформы Ubiq Mobile был создан еще в 2011 году для конференции FRUCT. При этом данный язык был реализован не вручную, а при помощи metaCASE-системы QReal [10], позволяющей быстро создавать редакторы новых визуальных языков. Более подробно данная среда визуального программирования будет опять же рассмотрена далее, в разделе «2) Существующие решения / 2.2) Используемые технологии». Тем самым результаты, представляемые на

конференции, были первой попыткой объединить технологии Ubiq Mobile и QReal для снижения сложности разработки мобильных приложений без специфичных знаний. В целом, данный эксперимент был признан успешным, так как разработанный визуальный язык:

- повысил наглядность и понятность программ;
- не требовал от разработчика специальных знаний объектно-ориентированного программирования и классического программирования мобильных приложений;
- тем самым, расширил аудиторию, которая может самостоятельно создавать свои мобильные приложения, в том числе со сложной логикой поведения.

Однако данный язык всё-таки имел ряд существенных недостатков, таких как:

- необходимость написания в блоках большого количества текстовых кусков кода;
- тем самым, необходимость наличия знаний и опыта программирования на текстовом языке C#;
- требование понимать общее внутреннее устройство библиотек Ubiq Mobile.

Таким образом, было решено придумать и реализовать другой визуальный язык для платформы Ubiq Mobile, который уже смог бы разрешить вышеупомянутые проблемы.

1) Постановка задачи

Тем самым, в качестве моей курсовой работы передо мной были поставлены следующие задачи:

1. Рассмотреть существующие визуальные языки, среды и платформы, в частности различные визуальные способы разработки мобильных приложений;
2. Придумать некоторый визуальный язык для разработки программ с нетривиальной логикой поведения, а именно, мобильных приложений. Однако такая задача может оказаться слишком сложной, поэтому в рамках данной курсовой надо было придумать язык не для всех возможных приложений, а только для некоторого класса мобильных приложений. Тем самым можно рассмотреть класс онлайн игр для двух пользователей с игровым полем, наподобие игры в морской бой или крестики-нолики, и класс приложений-«визиток»;
3. По формальному описанию этого языка сгенерировать в среде визуального программирования QReal соответствующий редактор визуального языка;
4. Написать генератор кода, который по формальному описанию программы на этом специальном визуальном языке будет генерировать текстовый код мобильного приложения. При этом генерацию можно поддержать частично, ограничиваясь только генерацией кода, который можно собрать, запустить и в итоге получить полностью прокликиваемое мобильное приложение. В частности, генерировать можно в код приложения, написанного под платформу Ubiq Mobile;
5. Реализовать на разработанном визуальном языке модельный пример мобильного приложения, например, онлайн игру в крестики-нолики. То есть формально описать программу на визуальном языке и сгенерировать готовый код приложения.

2) Обзор существующих решений

2.1) Визуальные среды программирования

На сегодняшний день существует множество различных визуальных языков и сред разработки мобильных приложений разной сложности, не требующих от разработчика специальных знаний и навыков программирования. Далее рассмотрим некоторые из них.

2.1.1) ДРАКОН

Для начала рассмотрим язык, позволяющий просто создавать какие-либо программы при помощи манипуляции визуальными объектами, а не только мобильные приложения. Одним из таких известных языков является визуальный алгоритмический язык программирования и моделирования ДРАКОН [8] (Дружелюбный русский алгоритмический язык, который обеспечивает наглядность). Этот язык разрабатывался в 1986-1996 гг. в рамках космической программы «Буран» при участии Федерального космического агентства (Научно-производственный центр автоматики и приборостроения им. акад. Н. А. Пилюгина, Москва) и Российской академии наук (Институт прикладной математики им. М. В. Келдыша) под руководством Владимира Пароннджанова [12]. Основной целью разработки данного языка было создание единого универсального языка программирования и моделирования, который смог бы объединить свойства:

- языков программирования реального времени (например, язык ПРОЛ2 для разработки бортовых комплексных программ под руководством Виктора Крюкова);
- проблемно-ориентированных языков (например, язык для разработки программ наземных испытаний ДИПОЛЬ под руководством Владимира Луциковича) ;
- языков моделирования (например, язык ЛАКС под руководством Константина Федорова);

В частности, язык ДРАКОН предназначается для проектирования, программирования, моделирования систем реального времени, а также для обучения. Помимо этого перед разработчиками языка были выдвинуты некоторые специфичные требования [12], такие как:

- предоставить человеку языковые средства для описания алгоритмов и структуры человеческой деятельности, для структуризации, систематизации и формализации императивных знаний;
- упростить общение между программистами и работниками разных отраслей, дисциплин и специальностей;
- улучшить качество создаваемого программного обеспечения по критерию «понятность алгоритмов и программ» за счет использования когнитивно-эргономического подхода к проектированию;
- улучшить производительность человеческого ума в целом;

Тем самым данный язык был построен на основе блок-схем, но с некоторыми улучшениями при помощи формализации, эргономизации и необычной структуризации алгоритмов. Одним из существенных отличий дракон-схем от блок-схем является то, что язык ДРАКОН позволяет создавать сложные алгоритмы «с сохранением наглядности даже для многостраничных схем». При этом рассматриваемый визуальный язык имеет два синтаксиса: графический (сами графические фигуры (иконы), правила их размещения на чертеже и правила взаимосвязи икон между собой при помощи соединительных линий) и текстовый (алфавит символов, правила их комбинирования и привязка к иконам).

Для построения дракон-схем было разработано некоторое исчисление над алфавитом икон, называемое «шампур-методом» и позволяющее создавать алгоритм при помощи последовательного применения правил построения к заготовкам [7]. В его основе лежат определённые базовые понятия [8], например такие как :

- «шампур-блок» — часть дракон-схемы, которая состоит из последовательности безусловно следующих друг за другом икон-операторов, расположенных по одной вертикали, и имеет ровно один вход сверху и один выход снизу;

- «главная вертикаль шампур-блока» — вертикаль, ориентированная сверху вниз и соединяющая его вход и выход;
- «побочная вертикаль шампур-блока» — одна из остальных вертикалей шампур-блока, расположенная правее главной;
- «силуэт» — фигура, полученная путем преобразования заготовки-силуэт за конечное число шагов с помощью некоторого набора операций [11] и представляющая собой разбиение алгоритма на этапы, являющиеся выделенной шампур-веткой и соединенные друг с другом через вершину-соединитель и «петлю силуэта»;
- «лиана» — часть дракон-схемы, имеющая один вход и один выход, называемые «началом лианы» и «концом лианы» соответственно[11];
- «пересадка лианы и заземление лианы» — некоторые операции с лианой, которые позволяют переносить концы побочных маршрутов без изменения структуры программы;

При этом маршруты алгоритма в дракон-схеме для большего понимания располагаются упорядоченно, а именно по принципу «чем правее --- тем хуже» [12], что означает, что маршрут, расположенный правее, описывает более неприятную ситуацию.

На рисунке 3.1 ниже представлен интуитивно понятный пример алгоритма «для тех, кто хочет завести собаку» [11]:

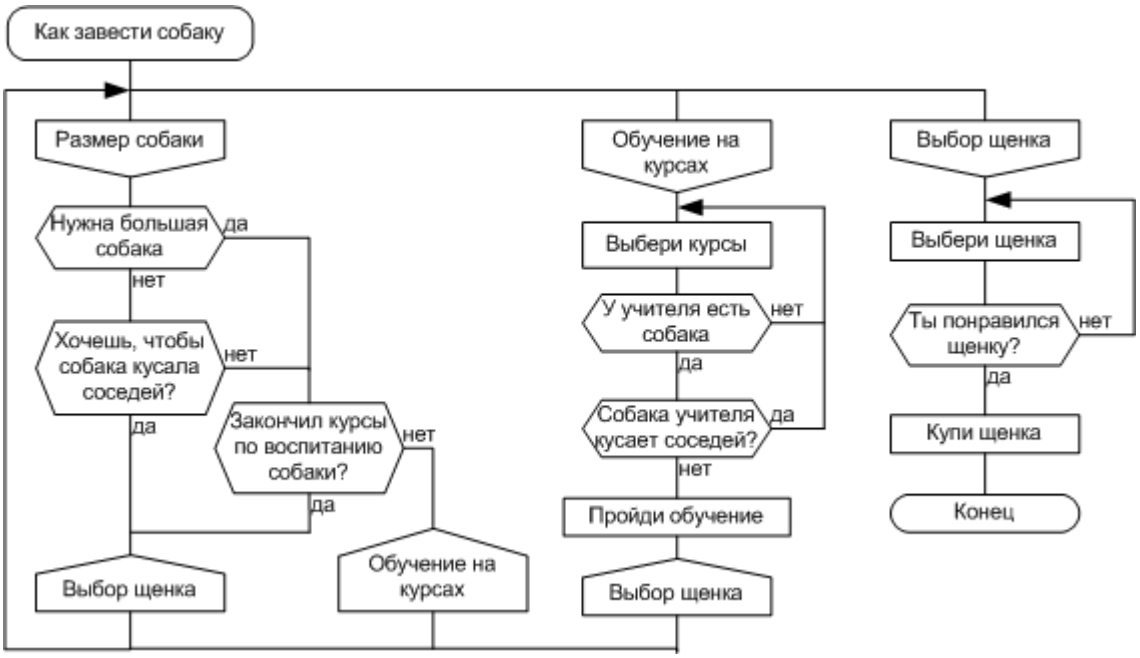

Рис. 3.1

2.1.2) HiAsm

Теперь рассмотрим бесплатную среду визуального программирования HiAsm [15] для разработки определённого класса приложений, таких как: win32, Qt, wxWidgets, сценарии и страницы PHP и JavaScript и приложения для устройств на базе Windows Mobile (например, КПК и некоторые смартфоны) [13]. При этом существует множество дополнительных пакетов для этой среды, позволяющих расширить возможности по созданию определённых приложений.

HiAsm предоставляет пользователю интуитивно понятный графический интерфейс, при помощи которого можно расставлять различные элементы на диаграмме, настраивать их свойства при необходимости и соединять их между собой связями, что позволяет создавать приложения. Тем самым при разработке своего приложения от пользователя не требуется знаний языков программирования или каких-то других специальный знаний, однако разбираться в той предметной области, для которой создается программа, ему всё же необходимо.

Далее представлено главное окно «конструктора программ» HiAsm (рис. 3.2):

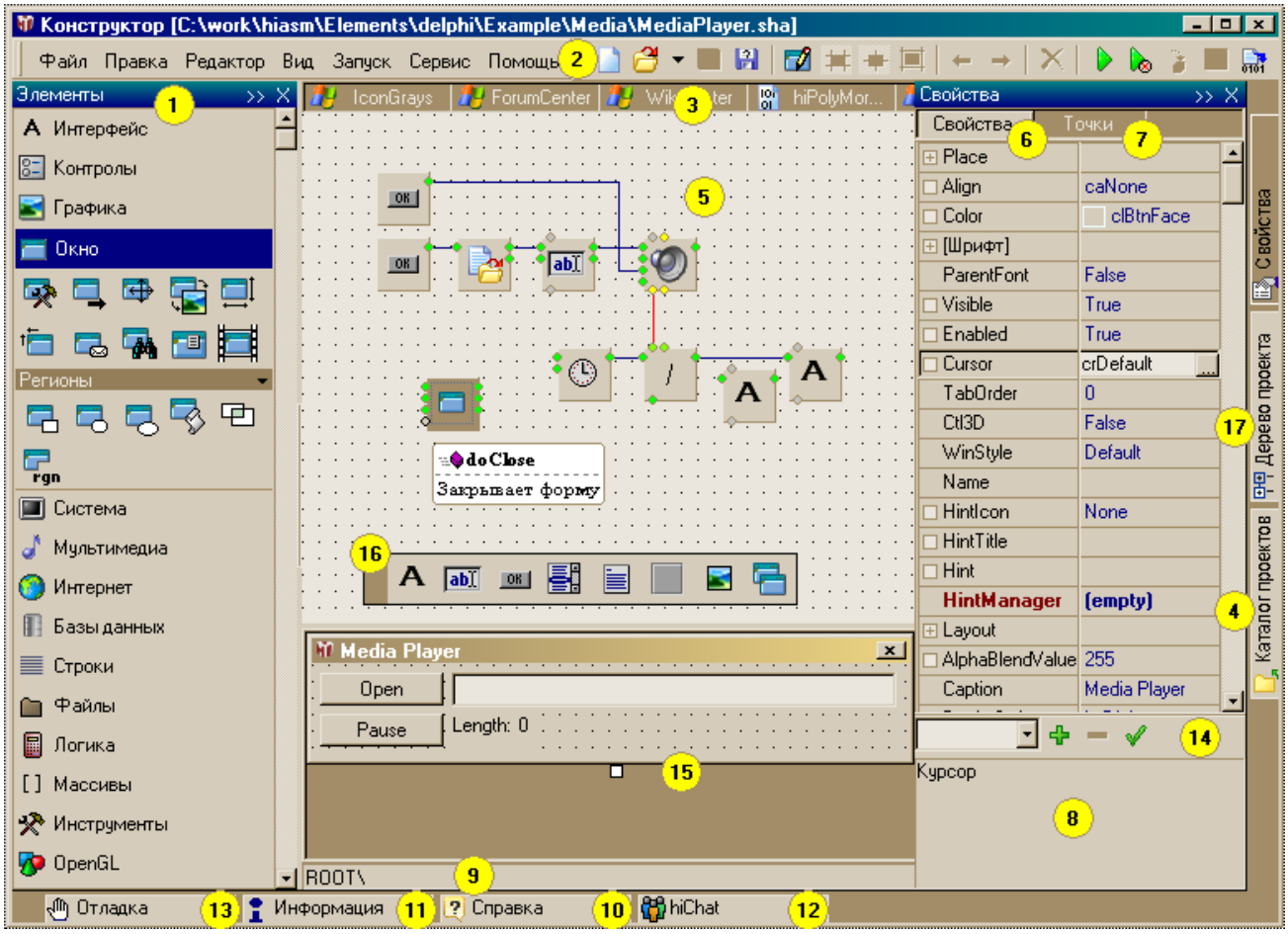

Рис. 3.2

На рисунке видно, что слева расположена «палитра элементов», которая предоставляет доступ к стандартным элементам среды. Каждый такой элемент позволяет выполнить некоторое достаточно узкоспециализированное действие, например «копировать файл», «сохранить файл», «скачать файл из интернета», «сложить два числа», «проиграть звук» и т.д. Именно из них и проектируется требуемая программа. При этом существует некоторый набор основных наиболее часто используемых элементов, например:

- из группы элементов «инструменты»:
  - «разветвитель» (Hub), который позволяет соединять несколько параллельных потоков в один или, наоборот, разделять один поток на несколько;
  - «поток-данные» (DoData), который позволяет помещать некоторые данные в поток;
  - «память» (Memory), который позволяет сохранять данные из потока;
- из группы элементов «логика»:
  - «арифметика» (Math), который позволяет выполнять простые математические операции;
  - «условный блок» (If_else), позволяющий сравнивать два значения между собой;
  - «цикл со счётчиком» (For), позволяющий организовать цикл, телом которого является идущая после него схема;

Справа находится «панель свойств», которая позволяет у каждого элемента настраивать его различные уникальные свойства, определяющие особенности его функционирования. Например, у элемента «кнопка» есть свойства, позволяющие задать его положение на форме, заголовок, используемый шрифт и т.п.

Снизу центральной части представлен «редактор формы» будущего приложения. А по центру находится «редактор схем», который является основным рабочем полем для создания приложений. Именно сюда пользователь перетаскивает с палитры необходимые элементы и строит из них приложение, соединяя

их между собой связями. При этом логику проектируемой программы, т.е. последовательность действий, определяют горизонтальные линии между элементами. А вертикальные линии, в свою очередь, задают элементам, какие данные откуда брать.

Отметим, что помимо стандартных блоков в среде есть некоторый набор стандартных примеров, которые зачастую можно переиспользовать, что еще упрощает и ускоряет процесс конструирования приложений.

Однако в данной среде отсутствуют какие-либо четкие ограничения на расположение элементов, что даёт волю пользователю и тем самым это может привести к тому, что при добавлении новых элементов программа может оказаться более сложной для понимания.

Таким образом, HiAsm [14] хорошо подходит для создания не очень сложных программ, которые не требуют большого количества расчетов, сравнений и других низкоуровневых процедур, которые сложно реализовать при помощи данного визуального средства.

2.1.3) Scratch

Scratch (скретч) [17] — это визуальная объектно-ориентированная среда программирования для обучения школьников младших и средних классов, которая позволяет создавать простые программы, в частности, анимированные и интерактивные презентации, мультики, игры и т.п. [19]. Scratch был разработан на языке Squeak небольшой группой ученых из «Lifelong Kindergarten Group» в Массачусетском технологическом институте. Основной целью является именно обучение детей некоторым основам программирования, развитие их логического мышления и творческих способностей [18], поэтому язык в некотором смысле представляет собой следующее развитие идей языка «Лого» и конструктора «Лего»[16].

Среда Scratch содержит в себе графический язык программирования, который позволяет создавать приложения из стандартных блоков, при этом есть возможность добавлять звук, некоторые стандартные визуальные эффекты и использовать собственные контроллеры через COM-порт [17].

Главное окно рассматриваемой среды программирования выглядит так (рис. 3.3):

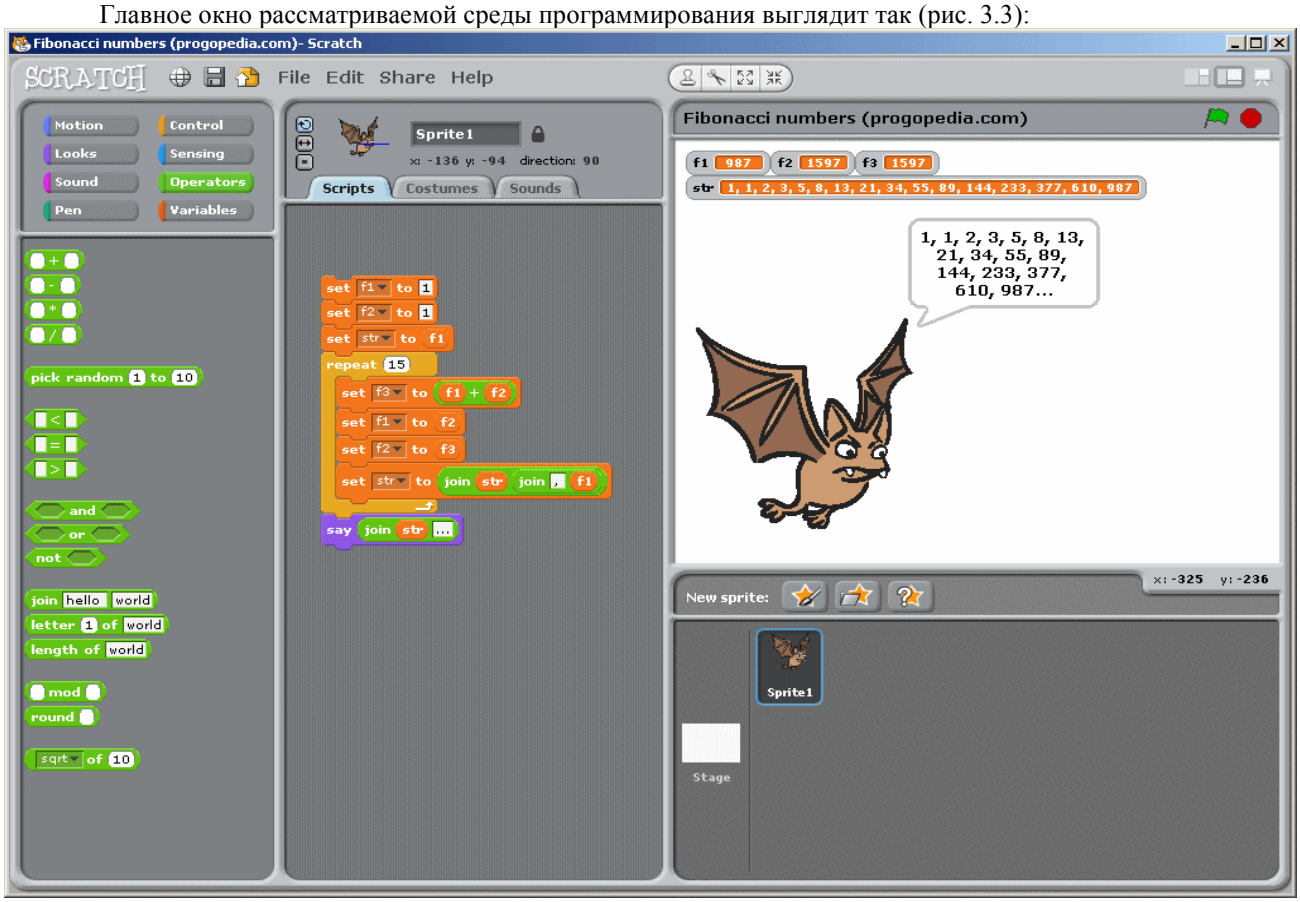

Рис. 3.3. [20]

На рисунке 3.3 показан пример вычисления последовательности чисел Фибоначчи.

Программа на языке Scratch представляет собой просто набор объектов-спрайтов (*sprite*), добавленных в специальное поле внизу справа (рис. 3.3). Каждому спрайту задается его графическое представление при помощи набора кадров-костюмов (*costume*) и логика поведения и взаимодействия при помощи сценария-скрипта (*script*). Костюмы спрайтов редактируются в специальном встроенном графическом редакторе (*Paint Editor*) (рис. 3.4), похожем на стандартные графические редакторы, такие как Paint.

Рис. 3.4.

Сценарии-скрипты программируются при помощи перетаскивания стандартных блоков из «окна блоков» в специальную «область скриптов» (первая и вторая область слева на рис. 3.3 соответственно). Блоки в зависимости от их типа имеют разные формы, тем самым не позволяя соединять вместе несовместимые блоки, что гарантирует синтаксическую корректность создаваемой программы. При этом блоки бывают трёх типов :

- блоки стека (*Stack Blocks*) — для объединения блоков в группу блоков, называемую стеком, которой можно манипулировать как отдельным элементом (например, циклы)
- блоки заголовков (*Hats*) — для создания заголовков к скретч-стекам (напр., блоки «когда ...»)
- блоки ссылок (*Reporters*) — для заполнения полей других блоков

Помимо этого блоки в языке Scratch еще делятся на 8 разных цветовых групп в зависимости от их функционального назначения :

- синий — «движение» (movement)
- сиреневый — «внешность» (looks)
- розовый — «звук» (sound)
- зелёный — «перо» (pen)
- жёлтый — «контроль» (control)
- голубой — «сенсоры» (sensing)
- салатовый — «арифметико-логические операции» (operators)
- оранжевый — «переменные» (variables)

Также многие блоки имеют белое редактируемое поле для ввода текстовых данных программистом.

Пример простой программы показан на рис. 3.5:

Рис. 3.5. [19]

Само же выполнение скретч-программы происходит в специальной области для этого сверху справа (рис. 3.3), называемой сценой (*stage*).

Впоследствии на основе подобной идеи создания программ при помощи стыковки визуальных логических блоков в виде«мозаики» было разработано множество других сред визуального программирования, в том числе более узкоспециализированных, таких как экспериментальные системы визуального программирования для платформы Android App Inventor и Google Blockly, платформы создания игр для мобильных устройств Stencyl и GameSalad Creator и т.д.. Некоторые из них опишем чуть подробнее.

2.1.4) App Inventor

App Inventor [21] — среда визуального программирования, разрабатываемая сначала в Google Labs, а потом в Массачусетском технологическом институте. App Inventor позволяет разрабатывать приложения для телефонов Android при помощи веб-браузера и подключенного телефона (или его эмулятора). При этом создавать свои приложения при помощи этой среды не очень сложно, так как от пользователя требуется только минимальный набор знаний программирования.

Для разработки своего приложения в рассматриваемой среде есть [21]:

- *The App Inventor Designer* (для выбора компонент создаваемого приложения);
- *The App Inventor Blocks Editor* (для визуальной сборки блоков программы, определяющих поведение компонент)

Основное окно среды (*The App Inventor Designer*) разделено на три части (рис. 3.6):

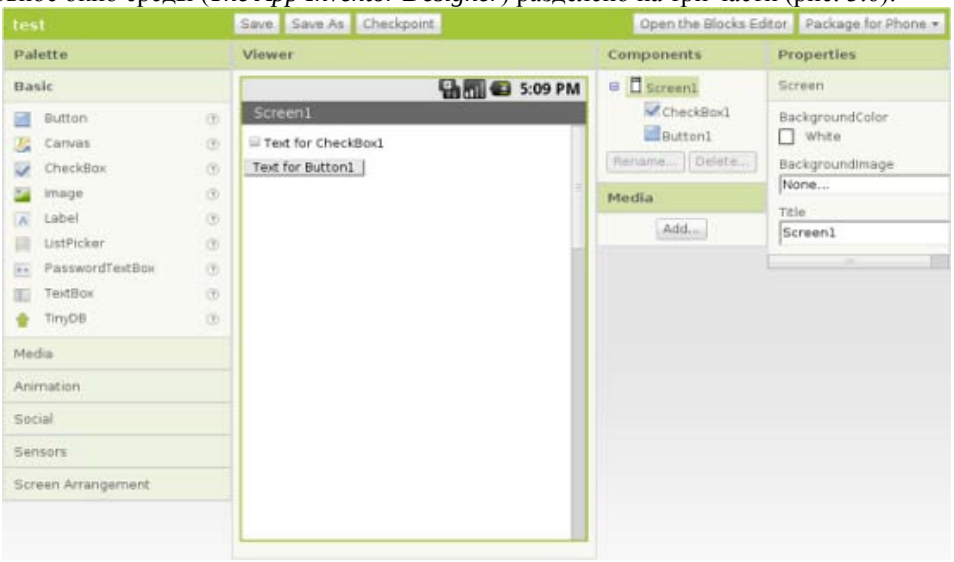

Рис. 3.6 [22]

Слева располагается специальная область «Палитра элементов», предоставляющая набор стандартных блоков для создания приложений (например, элементы интерфейса, функциональные блоки для работы с социальными сетями, веб-сайтами, сенсорами устройства, медийными устройствами и т.п.). В

центре расположено главное рабочее поле, похожее на экран телефона, куда и перетаскиваются выбранные компоненты. А справа расположена область для редактирования свойств выделенных элементов.

Еще одно окно *The App Inventor Blocks Editor* (рис. 3.7) представляет собой собственно «редактор приложения«, в котором пользователь может описывать логику и поведение программы при помощи специального визуального языка.

Рис. 3.7 [22]

Этот визуальный язык среды App Inventor очень похож на ранее рассмотренный язык Scratch, предоставляя пользователю набор стандартных логических блоков с готовыми действиями, которые можно соединять друг с другом, как куски мозаики. Разработчики среды создали большое количество разнообразных готовых блоков (например, блоки для хранения информации, для доступа к функциям телефона, для выполнения периодических действий и т.п.), что позволяет создавать довольно сложные приложения. Поэтому данная среда может быть полезна для обучения учащихся разных возрастов и преподавателей или даже для создания прототипов приложений опытными программистами (однако для разработки полноценного приложения серьёзным разработчикам такая среда будет недостаточна).

После этого программу можно скомпилировать при помощи встроенного специального компилятора App Inventor, который по визуальному блочному представлению генерирует каркас программы в байт-код Android, напоминающий Scheme. Далее можно упаковать разработанное приложение и сделать из него автономное приложение для установки на телефон (или его эмулятор).

Тем самым общая схема разработки выглядит примерно так (рис. 3.8):

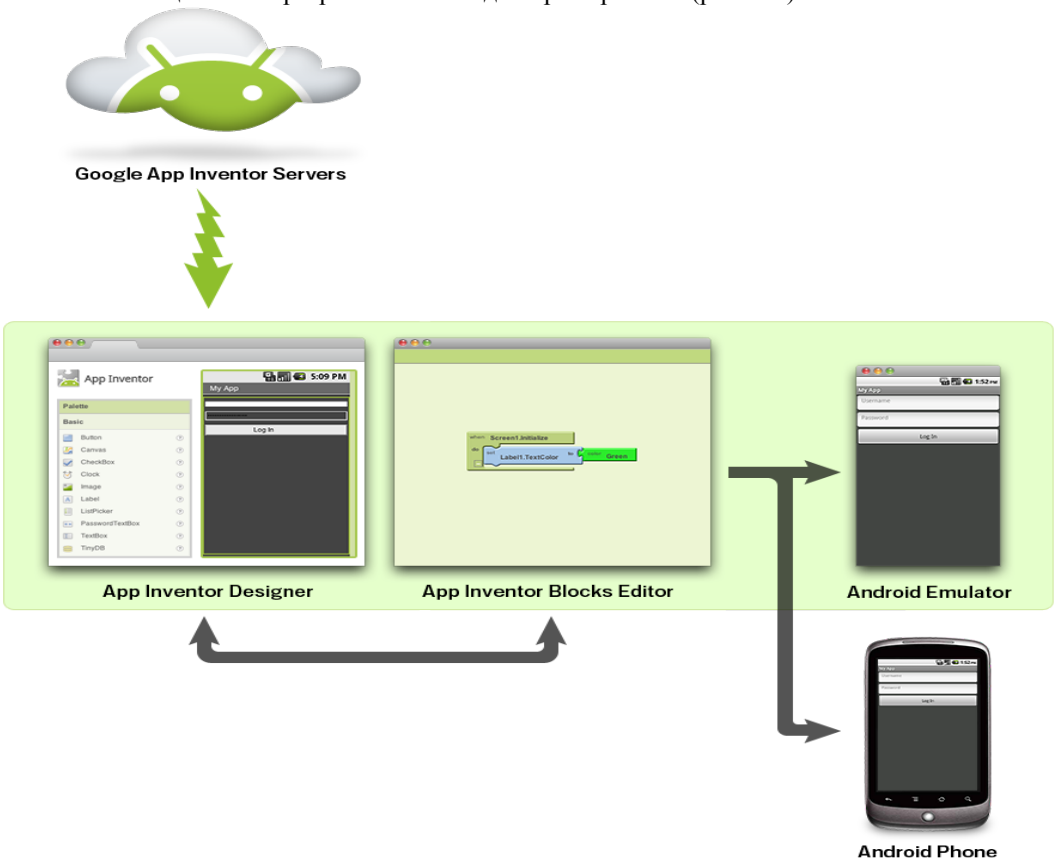

Рис. 3. 8 [21]

Существует еще и другая, основанная на Scratch среда визуального программирования для Android — **Google Blockly**, написанная на JavaScript. Она очень похожа на только что рассмотренный App Inventor и представляет собой веб-приложение, позволяющая создавать приложения на JavaSript, Dart или Python.

2.1.5) Stencyl

Stencyl [25] — это платформа для создания игр, позволяющая разрабатывать 2D-игры для мобильных устройств и обычных компьютеров (в качестве исполняемых программ) и онлайн-игры в браузерах (при помощи экспортирования в Сеть через Adobe Flash Player и HTML5). Платформа поддерживает управление игровой физикой и физикой столкновений при помощи Box2D, которые можно отключать частично или полностью в зависимости от требований пользователя.

Платформа Stencyl включает в себя некоторый набор модулей:

- The Behavior Editor (редактор поведения) — для создания и редактирования логики игры;
- The Tileset Editor (редактор тайлов) — для редактирования наборов тайлов, т.е. внешнего вида, анимации, внешнего вида столкновений и т.п. отдельных блоков игрового поля;
- The Actor Editor (редактор актёра) — для редактирования игровых персонажей, т.е. их внешнего вида, поведения, физики и т.д.;
- The Scene Designer (конструктор сцены) — для создания и редактирования основных игровых сцен, т.е. уровней и игровых состояний, используя уже созданные персонажи, тайлы и элементы поведения;

Помимо этого платформа поддерживает дополнительные инструменты, предоставляющие пользователю полезные возможности, такие как импорт изображений, шрифтов и звуковых файлов, редактирование настроек игры (например, управление игроком, разрешение и т.п.). Также в Stencyl входит «библиотека общих поведений», которая содержит некоторый набор готовой функциональности, обеспечивающий стандартное игровое поведение, общее для большинства 2D игр.

Создавать и редактировать поведение в игре можно в двух режимах: режиме кода (Code Mode) и режиме конструктора (Design Mode). В первом случае пользователь описывает логику игры на текстовом языке, что предоставляет больше возможностей для проектирования приложения. Во втором режиме пользователю предоставляется графический редактор, в котором он может манипулировать визуальными логическими блоками, реализующими некоторую готовую функциональность, соединяя их как мозаику. Тем самым визуальный язык в Stencyl идейно похож на Scratch, поэтому не требует дополнительный пояснений. Отличием же Stencyl от подобных сред является только сам набор элементов, который в данном случае специфичен для создания именно игровых приложений.

Ниже показаны примеры главных окон конструктора сцены (рис. 3.9) и редактора поведения (рис. 3.10):

Рис. 3.9 [24]

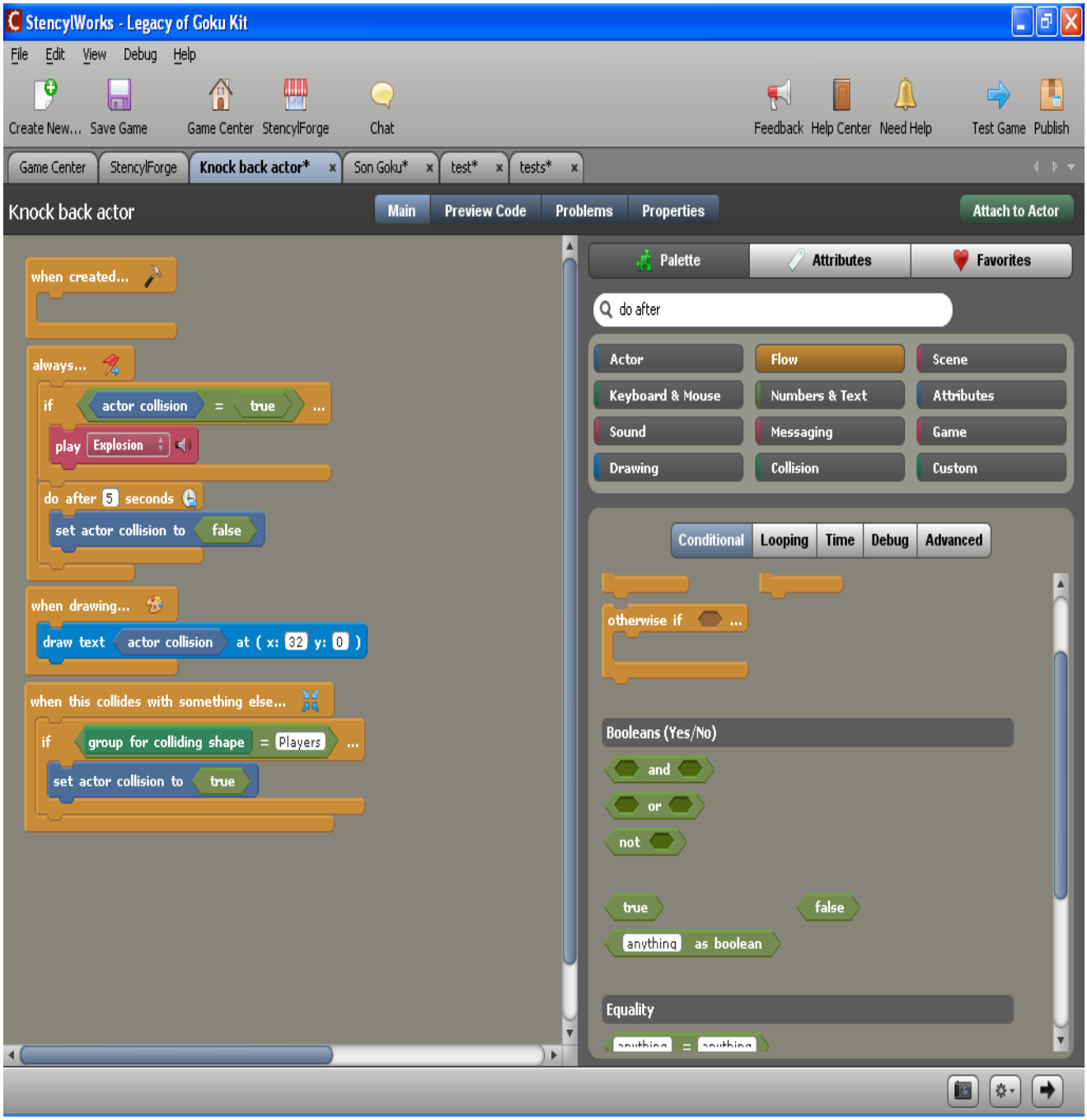

Рис. 3.10

2.1.6) GameSalad Creator

GameSalad [27] — специальный инструмент для создания игр для iPhone, iPod Touch и Android, разработанный компанией Gendai Games. Приложение предоставляет интуитивно понятный, визуальный интерфейс (рис. 3.11), похожий на рассмотренные ранее, но работает только на Mac OS X и Windows.

Рис. 3.11

GameSalad позволяет создавать несложные игры, анимации, интерактивные медиа приложения и т.п., тем самым предназначается в первую очередь графическим дизайнерам, аниматорам и разработчикам игр для быстрого прототипирования [26]. При этом у рассматриваемой среды есть некоторые особенности, такие как:

- использование таблицы для эффективного доступа к данным;
- редактирование игровых сцен в реальном времени, т.е. во время, как игра запущена и работает (при этом прозрачно отображается начальное состояние актеров игры);
- наличие специального режима предварительного просмотра игры для отладки и тестирования производительности и используемой памяти;
- специальная панель со списком загруженных игр и набором общих шаблонов игр, которые можно загружать в среду для переиспользования в своих приложениях;
- поддержка симуляции физики твердых тел для обработки реалистичных движений и столкновений (при этом физика обрабатывается в реальном времени);
- редактор математических выражений для описания сложного поведения и состояний игры (специальный редактор для «продвинутых» пользователей)

2.1.7) DragonRAD

DragonRAD [29] — кросс-платформенный инструмент для разработки, развертывания и управления мобильными приложениями предприятий для различных смартфонов и планшетов. Инструмент был разработан в 2010 году компанией Seregon Solutions Inc., как развитие SeregonMAP. Сейчас DragonRAD поддерживает создание приложений для широкого спектра мобильных устройств и операционных систем, таких как BlackBerry, Android, Windows Mobile и BlackBerry PlayBook (Beta), при этом инструмент продолжает увеличивать этот список [28].

Рассматриваемый инструмент включает в себя визуальную среду разработки, описания базы данных и правила синхронизации.

Графический редактор среды DragonRAD опять же представляет собой:

- рабочую область, внешне похожую на модель телефона, для которого разрабатывается приложение;
- палитру стандартных элементов, которые можно перетаскивать в эту рабочую область (слева от рабочей области);
- встроенный редактор свойств элементов (справа от рабочей области);

Ниже представлено главное окно среды (рис. 3.12) с открытым графическим редактором:

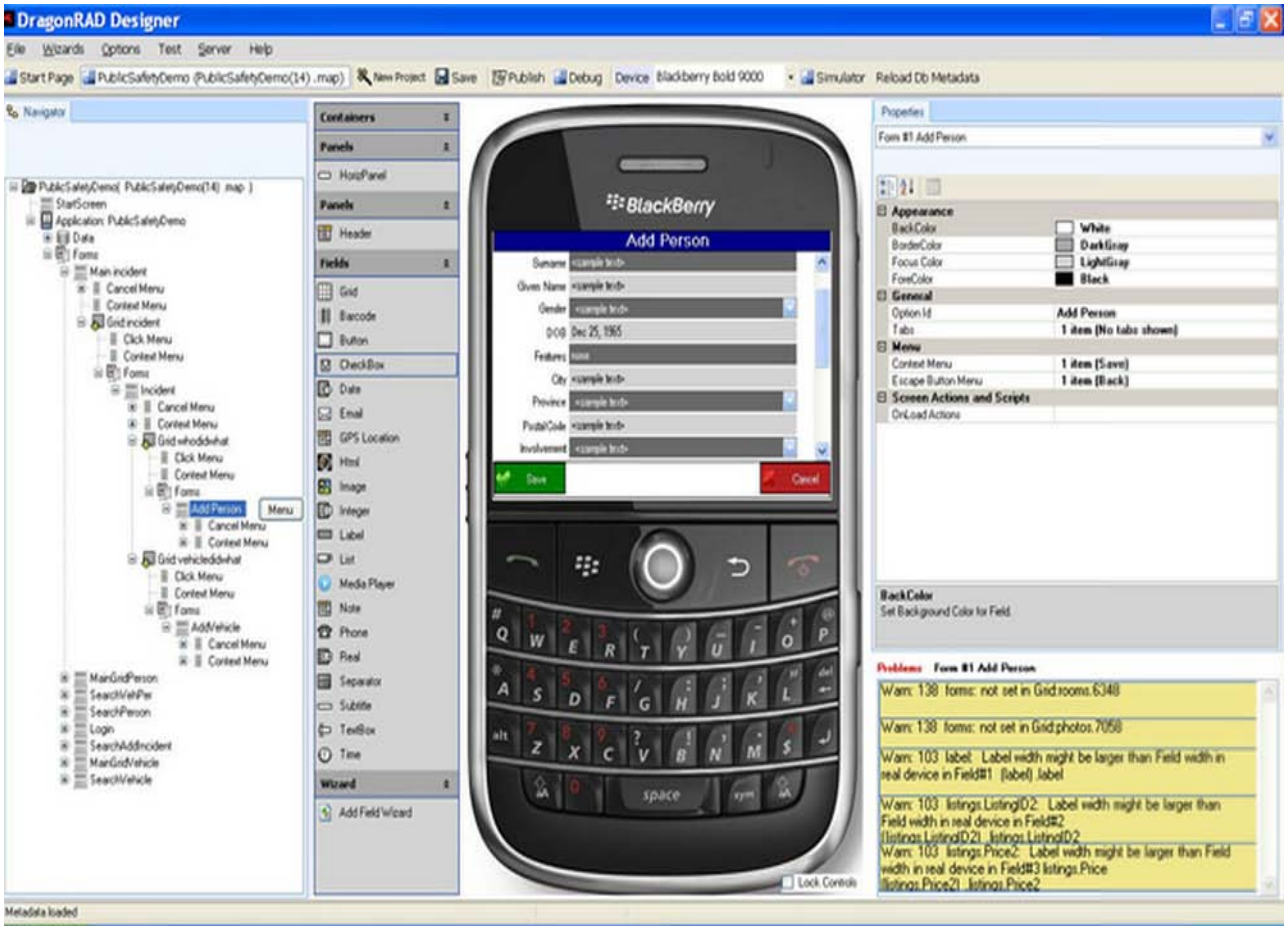

Рис. 3.12

Таким образом, DragonRAD позволяет быстро создавать приложения для широкого класса мобильных устройств без специальных знаний программирования, однако только довольно простые и без особой логики поведения, как было в рассмотренных ранее средах (хотя во многих случаях этого оказывается достаточно). Тем самым сейчас подобных сред программирования мобильных приложений, помимо DragonRAD, стало очень много, например: Smartface Platform, ViziApps и др.

2.1.8) Smartface Platform

Smartface [31, 32] — кросс-платформенный инструмент разработки и управления мобильными приложениями при помощи принципа «drag-and-drop» (перетаскивание графических элементов), который был разработан в 2006 году компанией Mobinex.

Платформа Smartface содержит три основные компоненты [30]:

- *Smartface Designer* — гибкий визуальный редактор, позволяющий быстро разрабатывать интерфейс мобильных приложений для широкого диапазона мобильных устройств с помощью перетаскивания в рабочую область элементов из набора стандартных элементов;
- *Smartface Server* — основной компонент платформы, который обеспечивает и управление жизненным циклом приложения и данными (с возможностью распределения и отслеживания различных этапов), и обновления для мобильных телефонов, и т.д.;
    - *Smartface Enterprise Manager* — веб-модуль для управления Smartface Server, предоставляющий доступ на основе ролей и с расширенными возможностями (например, отчетность);
- *Smartface Player* — компонента платформы, представляющая собой своего рода «прослойку» между Smartface-платформой и операционной системой мобильного устройства, необходимая для запуска Smartface-приложений;

При этом, как говорилось ранее, упомянутый графический редактор внешне и функционально похож на соответствующий редактор DragonRAD, поэтому не будем отдельно описывать его.

2.1.9) ViziApps

Помимо вышеупомянутых платформ визуального программирования существует еще отдельный класс сред для быстрой разработки простых мобильных приложений в *браузере*. Такие среды очень похожи на описанные ранее DragonRAD и Smartface, так как они предназначены для похожего класса приложений. В качестве примера можно рассмотреть следующую систему: ViziApps.

ViziApps [34] — веб-приложение для визуальной разработки нативных, гибридных и веб мобильных приложений для iPhone, IPad и Android смартфонов и планшетов, разработанное в 2010 году компанией ViziApps, Inc. (ранее MobiFlex, Inc.).

Данное приложение предназначено помочь предприятиям малого и среднего бизнеса избежать высоких затрат на разработку своих мобильных приложений. ViziApps предоставляет пользователям возможность создания приложений либо с нуля, либо при помощи настройки одного из уже существующих шаблонов (например, шаблоны приложений для конференций, недвижимости, финансовых услуг и т.д.). Графический редактор среды представляет собой холст с изображением мобильного устройства, для которого создается приложение, и палитру доступных элементов. При этом среда позволяет[33]:

- добавлять пользовательские фоны, изображения и кнопки;
- сопоставлять элементы (например, кнопки и таблицы) со стандартными действиями и функциями (например, электронная почта, SMS-сообщения, интеграция с социальными сетями, «переход на некоторую страницу»);
- создавать приложения, имеющие доступ к различным серверным данным через сеть (например, Google Docs таблицы, веб-сервисы, базы данных SQL и т.п.);
- создавать приложения, имеющие доступ ко всем родным ресурсам целевых устройств (например, камеры, аудио-файлы, список контактов, календарь, GPS и т.п.);
- и многое другое.

Таким образом, общая схема разработки собственного работающего мобильного приложения включает в себя сначала создание и/или настраивание приложения в специальном интуитивно понятном графическом редакторе в браузере, а потом просто скачивание готового приложения на мобильное устройство (рис. 3.13):

Рис. 3.13

2.2) Используемые технологии

2.2.1) Платформа Ubiq Mobile

Платформа Ubiq Mobile [4] разрабатывается на кафедре системного программирования Математико-Механического факультета СПбГУ с 2008 года при участии студентов и аспирантов из СПбГУ и СПбУ ИТМО. Рассматриваемая программная платформа предназначена для разработки современных кросс-платформенных распределенных мобильных сервисов. Основной целью проекта является как быстрое и эффективное создание программных систем, так и расширение круга программистов, способных написать эти мобильные приложения. Последнее достигается за счет того, что использование данной платформы не требует от программистов специального опыта и знаний в проектировании мобильных сервисов. При этом важной особенностью платформы является то, что созданные на её основе мобильные приложения могут работать практически в любых условиях мобильных сетей, в том числе в условиях слабо развитой мобильной инфраструктуры, в сетях с медленной передачей данных (GPRS, EDGE), а также в условиях неустойчивой связи, то есть при частых разрывах соединений (например, при движении в поезде).

На данный момент платформой Ubiq Mobile поддерживается широкий круг разнообразных мобильных устройств, начиная с простых Java-телефонов и заканчивая устройствами класса hi-end, в том числе поддержаны следующие мобильные платформы:

- Java ME (Symbian S40, Sony Ericsson)
- Symbian S60
- Andriod
- iOS

При этом разработка среды Ubiq Mobile продолжается и сейчас, поддерживая возможность работы с всё новыми мобильными устройствами.

Отметим, что данная платформа ориентирована не на всё множество приложений для мобильных устройств, работающих с Интернетом, а только на те программы, которые являются компонентами больших распределенных систем. Такие системы чаще всего характеризуются большим количеством пользователей, сложной бизнес-логикой, реализуемой на сервере и поддержкой различных видов мобильных платформ (то есть мультиплатформенностью).

В качестве примеров таких распределенных мобильных сервисов, которые поддерживает платформа Ubiq Mobile, могут выступать следующие виды приложений:

- многопользовательские онлайн игры (например, «Морской бой» или «Крестики-нолики»);
- приложения, взаимодействующие с «интеллектуальными» устройствами (например, системы передачи данных от камер видеонаблюдения);
- мобильные интерфейсы к корпоративным распределенным бизнес-приложениям;
- мобильный банкинг;
- интерактивные информационные сервисы со сложной серверной логикой и/или структурами данных и способами их обработки;
- Web 2.0-сервисы, которые используют данные, предоставляемые социальными сетями.

Архитектура, реализованная для платформы Ubiq Mobile, похожа на терминальную архитектуру больших mainframe-компьютеров, а именно включает в себя:

- сервер, на котором выполняется вся бизнес-логика сервисов;
- мобильные клиенты (то есть экземпляры клиентской программы, запущенные на мобильном устройстве), которые выступают простыми графическими терминалами.

Мобильное устройство на сервере представлено в виде виртуального хоста, на котором программист может рисовать при помощи предоставляемого платформой графического API. Однако сам мобильный терминал воспринимается сервисом просто как источник внешних событий (например, нажатие кнопок, ввод текста и т.д.,) которые обрабатываются приложением через API платформы в синхронном или асинхронном виде. Таким образом, вся работа сервисов происходит на сервере, а на мобильные устройства отправляются только полученные результаты.

При этом на сервере могут находиться и работать сразу несколько мобильных приложений, а тем самым конечному пользователю прозрачно предоставляется список всех доступных ему сервисов с выбранных серверов. Для возможности работы с такими приложениями мобильному клиенту необходимо

только заранее единожды зарегистрироваться и получить свой идентификационный номер, с которым в дальнейшем он будет работать.

Для создания своего мобильного приложения платформа Ubiq Mobile предоставляет программистам набор предметно-ориентированных библиотек, то есть программных интерфейсов API, например:

- Graphics API (базовое API для всех пользовательских приложений, предназначенное для обмена информацией между сервером и мобильным клиентом через холст);
- Extended Graphics API (графическое API высокого уровня, предоставляющее удобную работу с различными иерархическими структурами графических объектов, управляющими элементами и т.п.);
- Messaging API (внутреннее API, позволяющее установить синхронное и асинхронное взаимодействие между двумя приложениями через сообщения и почтовые ящики);
- и многие другие.

Таким образом, платформа Ubiq Mobile позволяет быстро и эффективно создавать собственные мобильные приложения со сложной логикой поведения, требуя от программиста только базовые знания языка C# и библиотек API самой платформы, что увеличивает производительность разработки сервисов для мобильных устройств.

2.2.2) Среда визуального программирования QReal

Среда визуального программирования QReal [5, 6] тоже разрабатывается на кафедре Системного программирования Математико-механического факультета СПбГУ уже в течение нескольких лет. QReal является CASE-системой, то есть позволяет при помощи набора визуальных моделей разрабатывать ПО, и metaCASE-системой, то есть даёт возможность быстро и удобно создавать свои собственные визуальные языки, редакторы и другие инструменты поддержки. Для разработки своего визуального языка необходимо сначала описать модель языка в метаредакторе при помощи специального визуального языка (метаязыка), а потом по этому формальному описанию сгенерировать редактор. После этого в среду можно подгрузить сгенерированный редактор, как плагин, и уже в нем решать свои задачи из области, для которой создавался этот визуальный язык. Метаязык позволяет формально описывать синтаксис визуального языка на более высоком уровне абстракции, а именно уровне «узел» - «связь». Также в среде есть возможность задавать ограничения на создаваемый язык, причем тоже при помощи визуального языка (языка задания ограничений). Помимо этого, данная система позволяет применять рефакторинги по описываемым шаблонам к программам на визуальном языке, а также визуально отлаживать свои программы.

Таким образом, для решения поставленной передо мной задачи в рамках данной курсовой, а именно реализация визуального языка для разработки мобильных приложений, было решено использовать рассмотренную metaCASE-систему QReal.

3) Реализация

3.1) Вариант А. Н. Терехова

Первым реализованным при помощи технологии QReal решением был визуальный язык, предложенный заведующим кафедры Системного программирования Математико-механического факультета СПбГУ Андреем Николаевичем Тереховым. Данный язык позволяет визуальным образом в реализованном редакторе формально описывать программы для мобильных устройств.

Основная идея рассматриваемого языка заключается в том, что программа собирается из блоков, каждый из которых реализует кусок готовой функциональности, часто специфичной для конкретной задачи. Так например, для игры в «Морской бой» есть блок «Размещение кораблей». То есть для каждой новой задачи помимо создания самого описания программы, необходимо также дополнять данный язык новыми блоками, требуемых конкретно для рассматриваемой задачи. Тем самым при разработке нового приложения требуется работа с метаредактором среды QReal, который и позволяет создавать новые и/или изменять существующие визуальные языки. При этом, если впоследствии потребуется генерация готового приложения по этому формальному описанию, то при дополнении языка новыми блоками придется также каждому из них задавать большие куски кода, в которые этот блок должен раскрываться при генерации.

Таким образом, для полноценной работы с данным визуальным языком, помимо знаний предметной области, необходимо также базовое владение средой QReal как metaCASE-системой и знание целевого текстового языка, в который будет происходить генерация приложения. Такие требования от пользователя могут заметно сузить круг программистов, способных эффективно создать своё приложение. Отметим также, что в данном случае язык не ограничивается только областью разработки мобильных приложений, а может быть применен и в любой другой области, так как все необходимые, но отсутствующие блоки нужно будет каждый раз создавать под конкретную задачу.

Ниже опишем те элементы рассматриваемого языка, которые были реализованы в рамках данной курсовой (табл. 4.1) :

№	displayedName / name	Описание элемента
1	QUbiq Diagram / qUbiqDiagram	Основная диаграмма, на которую непосредственно будут кидаться элементы для разработки программ. Позволяет формально описывать поведение создаваемого приложения. Содержит поля : • Name — собственное имя диаграммы, которое и будет именем будущей программы (задавать обязательно)
2	Send message to dispatcher / sendToDispatcher	Элемент языка, описывающий отправление сообщения диспетчеру. *Замечание*: диспетчер — это программный объект, устанавливающий связь между клиентом текущего пользователя с подключенным сервером, при этом для генерации приложения соответствующий код данного объекта реализуется заранее создателем данного языка. Поля: • Text — содержимое или тип сообщения
3	Receive message from dispatcher / receiveFromDispatcher	Элемент языка, описывающий ожидание сообщения от диспетчера. Поля: • Text — содержимое или тип сообщения
4	Stereotype node / stereotypeNode	Основной элемент языка, описывающий почти любое действие при помощи различных стереотипов и значений его свойств. Поля: • Text — дополнительное поле, для уточнения действия • Stereotype — стереотип элемента, который отображается на блоке в специальных ограничителях, а именно в «@».

5	Receive message from keyboard / receiveFromKeyboard	Элемент языка, описывающий ожидание сообщения от клавиатуры, то есть ожидается нажатие некоторой клавиши на клавиатуре. Поля: • Text — название клавиши клавиатуры, нажатие которой ожидается
6	Input splitter / inputSplitter	Элемент языка, позволяющий соединить несколько «связей» (потоков выполнения) в одну. Полей нет.
7	Output splitter / outputSplitter	Элемент языка, позволяющий одну «связь» (поток выполнения) разбить на несколько. Полей нет.
8	Link / commonEdge	Элемент языка, обозначающий поток выполнения программы. Полей нет.

Табл. 4.1. Описание элементов рассматриваемого визуального языка.

Далее приведён пример диаграммы, полностью описывающий онлайн игру в «Морской бой» между двумя игроками на мобильных устройствах и написанной на рассматриваемом визуальном языке (рис. 4.1) :

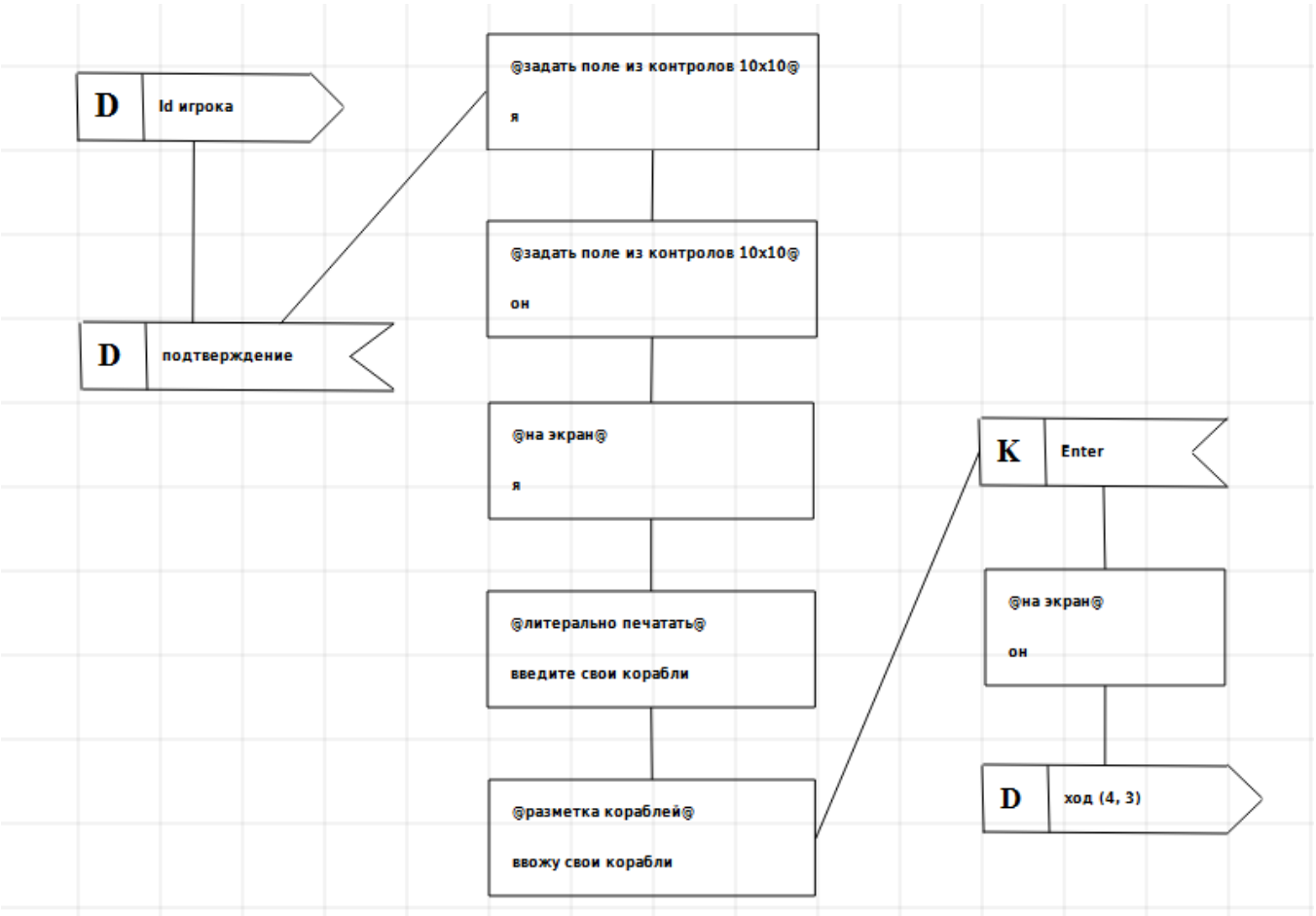

Рис. 4.1. Формальное описание игры «Морской бой» между двумя игроками, написанное на рассматриваемом визуальном языке.

Текущий игрок через диспетчер отправляет свой идентификационный номер и ждет подтверждения. Затем задаются два игровых поля 10 на 10 ячеек, которые будут соответственно отображать текущее

состояние кораблей текущего игрока и потопленных и подбитых кораблей его противника. Далее показывается игровое поле текущего игрока и от этого игрока требуется разместить его корабли, разметка которых закончится по нажатию клавиши Enter. Потом показывается поле с подбитыми текущим игроком кораблями и ожидается его ход, который сразу же будет отправлен противнику через диспетчер.

Также из примера видно, что в данном языке отсутствует специальный способ задания представления, то есть то, как приложение будет выглядеть для конечного пользователя. Внешний вид можно задать (но только частично) при помощи определенных элементов вида «stereotypeNode», которые нужно отдельно реализовывать. Однако, при этом описание логики и экранных форм приложения перемешаны на одной и той же диаграмме и неотделимы друг от друга, что снижает наглядность создаваемой программы.

Таким образом, после реализации данного визуального языка, появилась идея несколько улучшить предложенное решение, разрешив некоторые неточности рассматриваемого языка. В качестве результатов этого улучшения был реализован свой визуальный язык, описанный далее.

3.2) Собственный вариант

Основными целями, которые мы преследовали при разработке данного языка, являлись:

- сохранение наглядности разработки мобильных приложений (как и при визуальном языке «3.1) Вариант А.Н. Терехова»);
- отказ от требований к программисту уметь работать со средой QReal как metaCASE-системой;
- отказ от требований к программисту обладать знаниями текстового языка, в который будет генерироваться приложение (в данном случае это C# с использованием библиотек Ubiq Mobile);
- разделение задания логики поведения приложения и описания его внешнего вида (то есть задание экранных форм).

В соответствие с описанными целями в итоге был реализован рассматриваемый визуальный язык. Наш разработанный язык позволяет отдельно описывать экранные формы с правилами перехода и нетривиальную логику поведения мобильного приложения. Для этого в рамках одной метамодели были реализованы несколько редакторов для разных визуальных языков, а именно для языков представления «QUbiq Presentation Editor» и логики «QUbiq Logic Editor» соответственно. При этом язык задания логики, по сути, позволяет задавать логику поведения обработчиков различных событий, например при переходе на некоторую экранную форму или при нажатии клавиши или кнопки. Также дополнительно был реализован язык и соответствующий ему редактор «QUbiq Condition Editor», являющийся подъязыком языка описания логики программы и позволяющий, в свою очередь, задавать более сложные логические условия, которые сами по себе выделяются в отдельные диаграммы, а впоследствии могут быть использованы в одной или нескольких диаграммах обработчиков, чтобы повысить наглядность и понятность описания создаваемой программы.

Для того, чтобы разработать своё мобильное приложение, необходимо в первую очередь создать корневую диаграмму представления, которая является основной диаграммой и должна быть единственной для одной программы, так как именно в ней указываются все параметры создаваемого приложения, а именно его имя и путь до папки, куда оно будет генерироваться. Тем самым она сама по себе является самодостаточной, а значит по ней можно уже генерировать прототип искомого приложения. Диаграмма строится из специальных элементов языка представления, которые позволяют описывать набор экранных форм. Внутри каждой экранной формы при помощи других элементов данного языка задается, как собственно она выглядит, то есть какие кнопки, списки, сетки и текст на ней расположены. Каждым контролам и самим формам можно указать имя обработчика, который описывается отдельной диаграммой и будет вызываться при нажатии или переходе на них соответственно. При этом простые правила перехода между экранными формами можно задавать сразу на диаграмме представления при помощи соединения специальными связями контрола, при нажатии на который необходимо сменить отображения экрана, и самой формы, на которую следует перейти. Также в данном языке есть возможность задания и использования переменных, чьим типом может выступать целое число, строка, список или сетка (аналог двумерного массива) . Отметим, что все описываемые в программе переменные являются глобальными, то есть в рамках одного приложения всех их могут использовать все диаграммы. Поэтому было решено принять соглашение, что все переменные следует описывать на данной главной диаграмме представления, а на других диаграммах только ссылаться на них.

Для описания поведения одного обработчика необходимо тоже создать одну корневую диаграмму, но уже для задания логики. Однако всего в программе может быть несколько обработчиков. Каждая такая диаграмма представляет собою набор некоторых действий с указанием последовательности их выполнения, а также набор используемых переменных по необходимости. Таким образом язык позволяет переходить на другие экранные формы, запоминать значения в переменные, отправлять и ждать сообщения от диспетчера, ждать нажатие клавиши клавиатуры, брать элементы из списка или сетки по некоторому логическому условию, изменять параметры контролов на разных экранных формах и так далее. При этом также данный язык предоставляет возможность работы с условиями (вида оператора if), обычными циклами (вида while) и циклами по элементам списка или сетки (вида foreach), для указания которых требуются различные логические условия. Простые условия можно сразу задавать в текстовом виде внутри описываемого оператора, но сложные необходимо описывать на отдельной специальной диаграмме, а в операторе уже в качестве логического условия указывать имя этой диаграммы.

Диаграмма задания логических условий описывается при помощи некоторого подмножества визуального языка задания логики, исключая ненужные для логических условий блоки действий, например, посылка и ожидание сообщений, переход на другую форму и т.п.. Тем самым такие диаграммы так же состоят из некоторого набора переменных, допустимых действий и порядка их выполнения. При этом каждая диаграмма условий, так же как и диаграмма задания обработчиков, имеет начальный блок, указывающий, с какого действия следует начать выполнение данной части программы (то есть рассматриваемой диаграммы). Завершение же выполнения для них обеих происходит после того, как отработались все блоки. Однако, если для диаграммы задания обработчиков нету ограничений на последний элемент, то для диаграммы задания условий поток действий должен обязательно заканчиваться специальным блоком, который реализует возвращение результата, а именно логического значения. Отметим, что для данного визуального языка логическими значениями считаются «да» (истина), «нет»(ложь) и «не знаю» (нельзя с уверенностью сказать, истинно или ложно описываемое условие).

Далее более подробно опишем элементы всех этих трех визуальных языков, которые вместе позволяют разрабатывать онлайн игры на мобильных устройствах. В таблице ниже (табл. 4.2) для каждого элемента указываются отображаемый пользователю тип элемента (displayedName), уникальный идентификатор типа (name), а также общее описание и описание сего свойств. В среде QReal при обычной работе с визуальным языком пользователю отображается тип элемента, указанный в поле «displayedName». А фактический тип элемента, то есть значение поля «name», используется внутри самого редактора и дополнительных инструментов поддержки данного языка. В нашем случае такие имена типов элементов (то есть значения полей «name») не используются. Отметим, что у каждого экземпляра элемента также имеется поле «name», которое по логике использования аналогично соответствующему полю для типа элемента (то есть для элемента метаредактора при создании нового языка). Тем самым, имена элементов(значение поля «name») наших языков должны быть уникальными и именно они указываются при задании контролам или экранным формам их обработчиков и при использовании отдельных логических условий в этих обработчиках.

№	displayedName / name	Описание элемента
QUbiq Presentation Editor		
1	PresentationDiagram / qUbiqPresentationDiagram	Корневой элемент редактора для задания экранных форм. Диаграмма представления — основная обязательная диаграмма для создания программы. Одной этой диаграммы достаточно для генерации прототипа приложения. Содержит поля: • Name — собственное имя диаграммы • Program name — имя создаваемой программы • Path to generate — абсолютный путь до папки, в которую будет генерироваться код готового приложения Все поля обязательны для заполнения. *Замечание:* далее поля остальных элементов по умолчанию считаются обязательными.

2	Slide / slide	Основной элемент диаграммы представления. Позволяет описывать одну экранную форму. Поля: • Name — имя элемента (должно быть уникальным) • Size of slide — размерность экранной формы в ячейках, считая экранную форму некоторой таблицей. По умолчанию, поставлена размерность 1x1 (одна ячейка) • Description — некоторое текстовое описание экранной формы (НЕ обязательно) • Handler — имя обработчика, т. е. значение поля «Name» диаграммы, реализующий подпрограмму обработки данной экранной формы (НЕ обязательно)
3	Button / button	Элемент, представляющий собой кнопку. При этом необходимо, чтобы для кнопки обязательно либо был указан обработчик, либо было указано при помощи специальной связи (см. «10. Transition Link») правило перехода на другую форму. Поля: • Name — идентификатор элемента • Position on slide — позиция кнопки на экранной форме, а именно указание номера ячейки в соответствующей таблице. По умолчанию, указано значение «1:1», т. е. кнопка располагается в первой ячейке • Text on button — текст на кнопке (НЕ обязательно) • Handler — имя обработчика, т. е. значение поля «Name» диаграммы, реализующей подпрограмму обработки нажатия на данную кнопку (НЕ обязательно)
4	Exit Button / exitButton	Элемент, представляющий собой кнопку выхода из приложения. Все поля идентичны полям обычной кнопки (см. «3. Button»). Но при этом задавать обработчик или правило перехода не нужно.
5	List / list	Элемент, представляющий собой список элементов некоторого типа, а именно либо текста, либо картинок, либо кнопок. Поля: • Name — идентификатор элемента • Count of elements — количество элементов в списке • Id of element — идентификатор(-ы) представителя(-ей) элементов списка (НЕ обязательно, вместо этого можно нужные элементы просто положить в данный блок, как в контейнер). При этом, данные элементы будут заполнять список, пока не наберется заданное число элементов. • Is Null in begin — логическое значение (true / false), означающее надо ли скрывать элемент при отображении. По умолчанию задано значение «false», т. е. элемент будет отображаться. • Handler — аналогично элементу «кнопка» (см. «3. Button») • Position on slide — аналогично элементу «кнопка» (см. «3. Button»).
6	Text / text	Элемент, представляющий собой текст на экранной форме. Поля: • Name — идентификатор элемента • Text — сам текст (НЕ обязательно) • Is Null in begin — аналогично элементу «список» (см. «5. List»)

7	Image / image	Элемент, представляющий собой картинку на экранной форме. Поля: • Name — идентификатор элемента • Path to image — абсолютный путь до файла с картинкой • Is Null in begin — аналогично элементу «список» (см. «5. List»)
8	Grid / grid	Элемент, представляющий собой сетку (аналогично таблице) на экранной форме. Поля: • Name — идентификатор элемента • Dimension — размерность сетки, аналогично свойству «Size of slide» у элемента «экранная форма» (см. «2. Slide») • Handler — аналогично элементу «кнопка» (см. «3. Button») • Id of element — аналогично элементу «список» (см. «5. List») • Is Null in begin — аналогично элементу «список» (см. «5. List»)
9	Variable / variable	Элемент, представляющий собой переменную. Поля: • Name — идентификатор элемента (т. е. просто имя, по которому впоследствии будем обращаться к этой переменной) • Type of variable — тип переменной. Типом может быть: text (строка символов), list (список), image (картинка), grid (сетка), number (целое число). По умолчанию указано значение «text» • Current value — текущее или начальной значение переменной(НЕ обязательно) • Description — описание элемента (НЕ обязательно)
10	Transition Link / transitionLink	Связь, позволяющая задавать правила перехода. Задается ОТ контрола, при нажатии на который необходимо перейти на другую экранную форму, К соответствующей экранной форме. Поля: • Name — идентификатор элемента (НЕ обязательно)
QUbiq Condition Editor		
1	Condition Diagram / qUbiqConditionDiagram	Корневой элемент редактора для задания логических условий. Поля: • Name — идентификатор элемента • Description — описание элемента (НЕ обязательно)
2	Change element / change	Элемент, позволяющий менять значения свойств некоторого другого элемента. Поля: • Name — идентификатор элемента (НЕ обязательно) • Element ID — идентификатор элемента, чьё свойство необходимо поменять • Name of property — имя свойства элемента (в точности, как записано в элементе) • New property value — новое значение данного свойства, на которое мы меняем
3	Take elements of grid or list / takeGridOrListElements	Элемент, позволяющий запоминать в переменную список элементов, выбранных из другого списка или сетки по некоторому логическому условию. Поля: • Name — идентификатор элемента (НЕ обязательно)

		• Save in variable — идентификатор переменной, в которую запоминаем результат. Тип переменной должен соответствовать типу ожидаемого результата • ID of grid or list — идентификатор списка или сетки, откуда выбираются элементы • By condition — логическое условие, по которому выбираются элементы списка или сетки
4	Return (bool value) / return	Элемент, представляющий собою оператор «return», т. е. позволяет возвращать логическое условие. Поля: • Name — идентификатор элемента (НЕ обязательно) • Return value — возвращаемое логическое значение, которое может равняться либо none(не определено), либо false(ложь), либо true(истина). По умолчанию задано значение «none».
5	IF / ifNode	Элемент, представляющий собою оператор «if». С этим элементом используется специальная связь (см. далее «12. Conditional Link»), при помощи которой указывается, куда далее передается управление в зависимости от результата условия. Поля: • Name — идентификатор элемента (НЕ обязательно) • Condition — само логическое условие. При этом в качестве условия можно либо задать условие сразу в текстовом виде, либо указать идентификатор диаграммы некоторого условия.
6	While / while	Элемент, представляющий собою оператор «while». С этим элементом используется специальная связь (см. далее «12. Cyclic Link»), при помощи которой указывается, куда далее передается управление в зависимости от результата условия в блоке. Поля: • Name — идентификатор элемента (НЕ обязательно) • Counter ID — идентификатор переменной, которая будет играть роль счетчика данного цикла • Condition — аналогично элементу «оператор if» (см. «5. IF»)
7	For / forNode	Элемент, представляющий собою либо двойной оператор «for», либо оператор «foreach» по таблице. С этим элементом используется специальная связь (см. далее «12. Cyclic Link»), при помощи которой указывается, куда далее передается управление в зависимости от результата условия в блоке. Поля: • Name — идентификатор элемента (НЕ обязательно) • Counter ID by height — идентификатор переменной, которая будет играть роль счетчика данного цикла по высоте таблицы (НЕ обязательно) • Counter ID by width — идентификатор переменной, которая будет играть роль счетчика данного цикла по ширине таблицы (НЕ обязательно). Если в поле «ID of grid or list» указан элемент «список», то данное поле игнорируется. • ID of grid or list — идентификатор списка или сетки, по которому проходим в цикле *Замечание:* Если указаны все поля, то цикл аналогичен двумя операторам «for». Если оба счётчика не указаны, то цикл аналогичен оператору «foreach» по списку или двумерному массиву.

8	Output splitter / outputSplitter	Элемент, позволяющий несколько потоков управления объединить в один. Поля: • Name — идентификатор элемента (НЕ обязательно)
9	Begin node / beginNode	Элемент, который указывает, откуда должно начинаться выполнение подпрограммы. Поля: • Name — идентификатор элемента (НЕ обязательно)
10	Variable	см. описание элемента «10. Variable» из редактора «QUbiq Presentation Editor»
11	Link / link	Связь, позволяющая задать направление потока управления при выполнении подпрограммы. Задается От текущего действия (блока) К следующему. Поля: • Name — идентификатор элемента (НЕ обязательно)
12	Conditional Link / conditionLink	Связь, позволяющая задать направление потока управления в зависимости от результатов логического условия в элементе «IF» при помощи своего стереотипа. Задается От элемента-оператора условия (см. «5. IF») К некоторому следующему элементу. Поля: • Name — идентификатор элемента (НЕ обязательно) • Bool stereotype — значение стереотипа связи, зависящее от результатов логического условия соответствующего оператора. Поле может принимать те же значения, что и возвращать условный оператор (см. свойство «Return value» у элемента «4. Return (bool value)»).По умолчанию указано значение «none».
13	Cyclic link / cyclicLink	Связь, позволяющая задать направление потока управления в зависимости от завершенности цикла, с которым используется данное соединение. Задается либо ОТ элемента-оператора цикла (см. «6. While» и «7. For») К некоторому следующему элементу, либо ОТ последнего оператора цикла К элементу-оператору, в зависимости от заданного стереотипа. Поля: • Name — идентификатор элемента (НЕ обязательно) Cyclic stereotype — значение стереотипа связи, зависящее от стадии завершенности цикла. Значение может быть равным либо next (т. е. переход ОТ последнего действия в цикле К элементу-циклу), либо after (т. е. переход ОТ элемента «цикл» К следующему действию). По умолчанию задано значение «next»
		QUbiq Logic Editor
1	Logic Diagram / qUbiqLogicDiagram	Корневой элемент редактора для задания поведения некоторого обработчика. Поля: • Name — идентификатор элемента • Type of handler — тип обработчика. Поле может принимать значения: slide handler (обработчик для экранной формы), list handler (для списка), button handler (для кнопки), grid handler (для сетки)

2	Take / take	Элемент, позволяющий запоминать в переменную либо один или несколько элементов из списка, либо сами контролы. Поля: • Name — идентификатор элемента (НЕ обязательно) • To element ID — идентификатор переменной, куда запоминается элемент(-ы). При этом эта переменная должна быть соответствующего типа • From element ID — идентификатор элемента, кого мы запоминаем, или списка, откуда мы вытаскиваем элемент(-ы) • Type of memory — тип запоминания. Поле может принимать значения: only unselected data (только не выделенные элменты), only selected data (только выделенные элементы), all data (все элементы). Это поле имеет смысл только, если в поле «From element ID» указан элемент «список». По умолчанию указано значение all data.
3	Go to slide / go	Элемент, позволяющий задать переход на следующую экранную форму с текущей. Поля: • Name — идентификатор элемента (НЕ обязательно) • Go to ID slide — идентификатор экранной формы, куда необходимо перейти
4	Send message / message	Элемент, позволяющий описать отправку сообщения. Поля: • Name — идентификатор элемента (НЕ обязательно) • Type of sent message — тип сообщения. Поле принимает одно из следующих значений: request game(запрос игры), request of playing field status (запрос состояния игрового поля противника), sent a playing field status (отправка состояния своего игрового поля), request a players list(запрос списка доступных игроков) • Reciever — получатель отправляемого сообщения. Им могут быть: server (текущий сервер), other player(противник), keyboard(клавиатура). *Замечание:* в данном случае значение «keyboard» не имеет смысла, поэтому отрабатывается, как «server» • Additional text — сам текст или данные сообщения (НЕ обязательно)
5	Wait message / waitMessage	Элемент, позволяющий задать ожидание сообщения. Поля: • Name — идентификатор элемента • Type of wait message — тип ожидаемого сообщения. Поле принимает одно из следующих значений: response about playing field status (ответ на запрос состояния игрового поля противника), response to game request (ответ на запрос игры с некоторым игроком), pressing on a keyboard (ожидание нажатие клавиши) • Sender — аналогично полю «Reciever» для элемента «отправка сообщения» (см. «4. Sent message») • Additional text — дополнительный комментарий к данному элементу (НЕ обязательно)
6	Variable	см. описание «10. Variable» из редактора «QUbiq Presentation Editor»

7	Change element / change	см. все описания соответствующих элементов (№2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 13) из редактора «QUbiq Condition Editor»
8	Take elements of grid or list	
9	Return (bool value)	
10	IF	
11	While	
12	For	
13	Output splitter	
14	Begin node	
15	Link	
16	Conditional Link	
17	Cyclic link	

Табл. 4.2. Описание элементов рассматриваемого визуального языка.

Конкретные примеры, реализованные на рассмотренном визуальном языке, будут представлены ниже, в пункте «4) Апробация».

3.3) Генерация кода

После того, как приложение формально описано на предложенном в пункте «4.2. Собственное решение» визуальном языке, есть возможность по полученному набору диаграмм сгенерировать целевой текстовый код, который будет реализовывать данную программу. При этом генерация приложения происходит по логической модели, что является типичной ситуацией для среды QReal [6]. Однако в рамках данной курсовой была поддержана генерация не вся целиком, а только прототипа мобильного приложения в код платформы Ubiq Mobile (см. «2.2.1) Ubiq Mobile»). Тем самым по описанию программы сейчас генерируются:

- все экранные формы итогового приложения с корректным их заполнением;
- правила перехода между существующими экранными формами;
- заглушки для обработчиков контролов и форм;
- описание переменных со всех диаграмм.

Таким образом, при генерации кода в папке, указанной в свойстве «Path to generate» диаграммы «PresentationDiagram» языка представления «QUbiq Presentation Editor», создается подкаталог с именем разрабатываемой программы. Далее при проходе логической модели визуальных диаграмм будут генерироваться необходимые файлы с кодом, а именно несколько C# - файлов с расширением «.cs», проектный файл «.csproj» и файл конфигураций «.config». Далее этот проект необходимо собрать, в результате чего появляется собранная библиотека «.dll», реализующая наше приложение. Потом можно перенести эту сгенированную программу в соответствующую папку сервера, прописать её в файле с доступными приложениями и запустить сервер. Тогда клиенты всех мобильных устройств, которые подключены к данному серверу, будут видеть данное разработанное приложение и смогут работать с ним, как и с другими программами сервера.

Отметим, что все эти требуемые для сборки приложения файлы генерируются в соответствии с заранее заданными шаблонами. В этих шаблонах записан общий для всех генерируемых программ код и специальные пометки, которые в процессе генерации заменяются на тоже текстовые куски кода, но уже специфичные для каждого приложения.

Сама генерация проходит следующим образом. Сначала мы проходим по логической модели и вызываем метод генерации приложения для каждой диаграммы представления «PresentationDiagram», так как в рамках одного проекта может быть описано несколько программ. Далее генерация прототипа мобильного приложения идейно проходит в несколько следующих этапов:

- проходим по логической модели, ищем все объявления переменных и записываем их в итоговый файл соответствующего шаблона класса Variables в качестве его полей;
- для каждой диаграммы представления генерируем отдельный класс для описания экранных форм и правил перехода между ними;
- проходим по всем диаграммам обработчиков логической модели и генерируем соответствующий для них код в класс для экранных форм (замечание: сейчас для обработчиков генерируются только заглушки);
- генерируем файл конфигураций и проектный файл, заполняя соответствующий шаблон именами уже сгенерированных файлов.

Подробнее рассмотрим второй этап. При генерации диаграммы представления мы проходим по всем элементам «Slide» («экранная форма»), для каждого из которых генерируем отдельный метод, возвращающий требуемый «визуальный элемент» (т. е. аналог экранной форме в коде на платформе Ubiq Mobile). Каждый такой метод генерируется в соответствии с содержимым элемента экранной формы, как контейнера элементов - контролов. При этом расположение искомых контролов определяется значениями свойств «Position on slide» соответствующих элементов. Правила перехода между экранными формами генерируются тоже отдельными методами, но которые в коде являются обработчиками нажатия на кнопку («Button»). Сам этот метод в коде представляет собой просто изменение основной отображаемой экранной формы на ту, на которую необходимо перейти.

Таким образом, мы можем по формальному описанию генерировать прототипы готовых мобильных приложений, а в данном случае онлайн игр для мобильных устройств.

4) Апробация

В качестве апробации созданных визуальных языков для разработки мобильных приложений была реализована онлайн игра в крестики-нолики для одного игрока на мобильном устройстве. После формального описания данного приложения была произведена автоматическая генерация его прототипа. Ниже рассмотрим несколько основных диаграмм, разработанных для рассматриваемой игры (рис. 5.1, рис. 5.2, рис. 5.3):

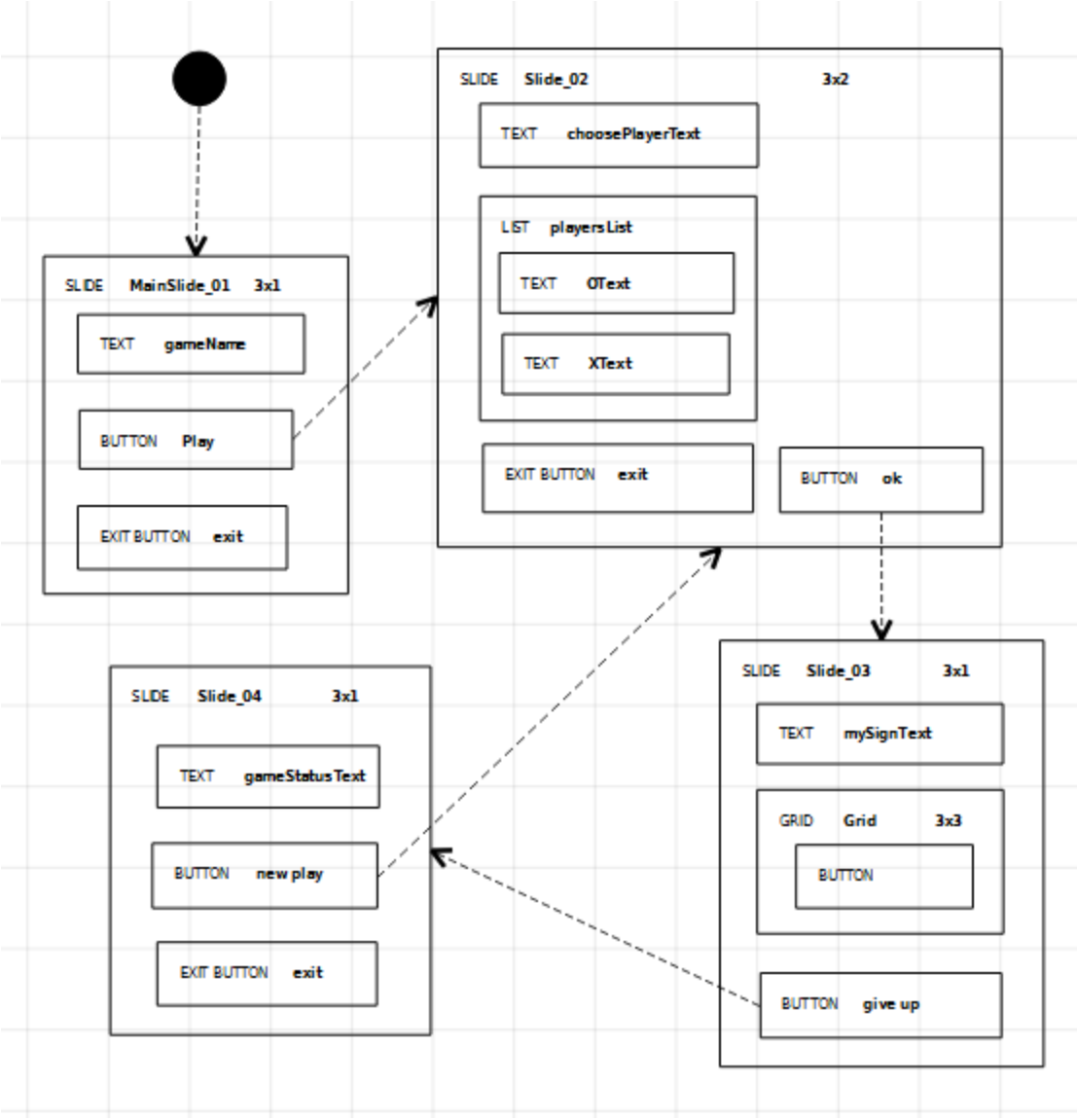

Рис. 5.1. Основная диаграмма экранных форм.

На данной диаграмме представлены четыре экранные формы и соответствующие правила перехода между ними, указанные пунктирными линиями. Первой формой, которая показывается пользователю, является элемент «MainSlide_01», на котором располагаются имя игры (берется из значения переменной «gameName»), кнопка начала игры и кнопка выхода. Переход на следующую форму происходит по нажатии на кнопку начала игры. Вторая форма состоит из вводного текста вида «Выберите, за кого будете играть», списка

выбора крестиков или ноликов, кнопки подтверждения («ОК») и кнопки выхода. При нажатии на кнопку «ОК» мы переходим на третью форму, которая как раз и содержит основное игровое поле, кнопку «сдаться» и текст с указанием, какая сторона сейчас ходит. На четвертую (последнюю) форму мы переходим либо при окончании игры (т. е. по выполнении соответствующего обработчика), либо при принятии поражения. На последней экранной форме отображается текст со состоянием игры текущего игрока (победа, проигрыш или ничья), кнопка выхода и кнопка начала новой игры, при нажатии на которой мы снова переходим на вторую форму.

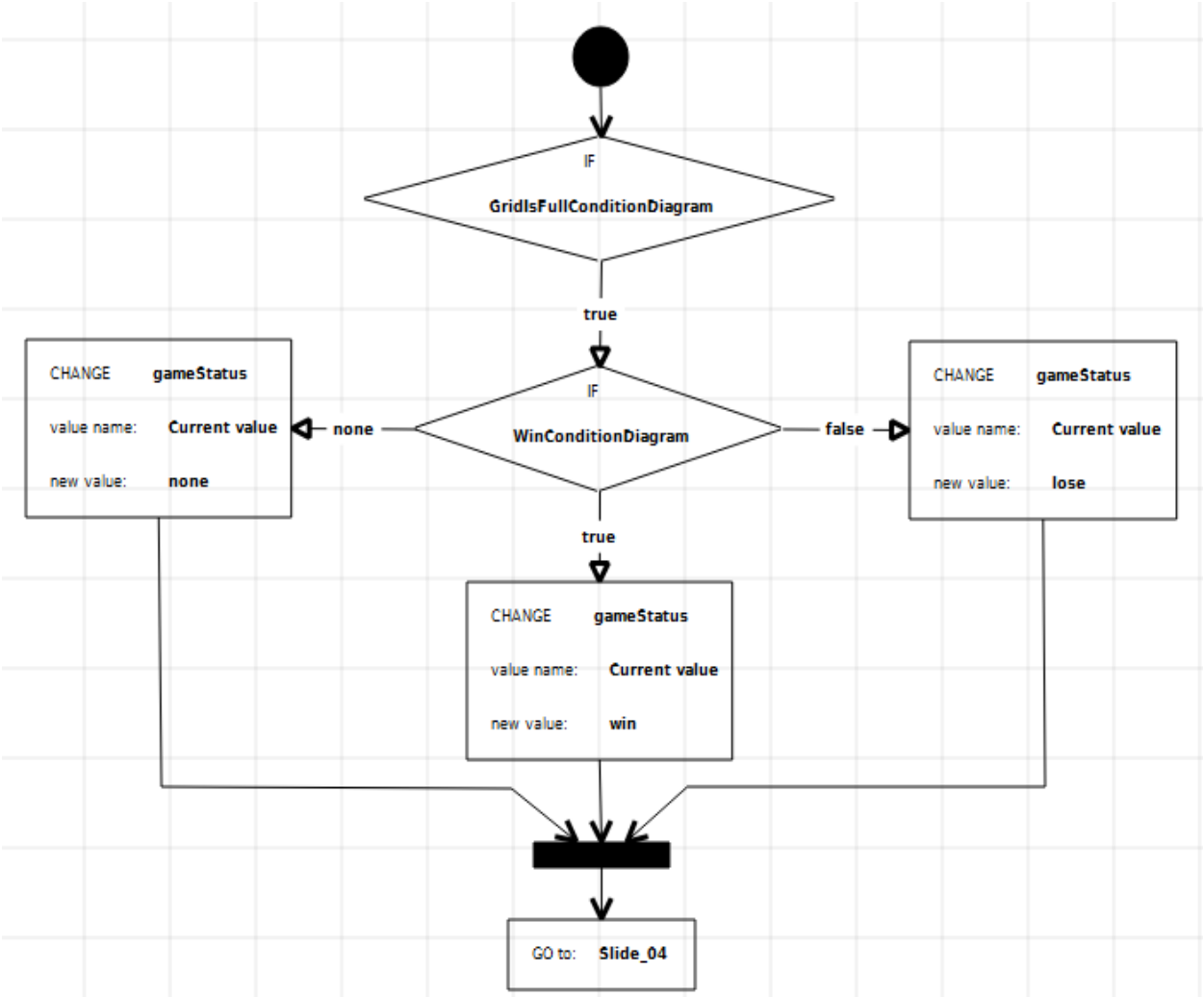

Рис. 5.2. Диаграмма обработчика третьей экранной формы, а именно основного игрового поля.

Данный обработчик, по сути, бесконечно отрабатывает описанные действия, пока мы находимся на третьей экранной форме и пока игра не дошла до завершающего состояния. Рассматриваемый обработчик сначала проверяет условие, заполнены ли все ячейки сетки (вызов диаграммы условия «GridIsFullConditionDiagram»). Если это условие будет ложным («false») или неопределенным («none»), то ничего не происходит. Если же условие окажется истинным («true»), то далее проверяется условие на победу в игре (диаграмма условия «WinConditionDiagram»), результатом чего будет «победа» (true), «проигрыш» (false) или «ничья» (none). Полученное текущее состояние игры записывается в переменную «CurrentValue», а потом происходит переход на четвертую экранную форму.

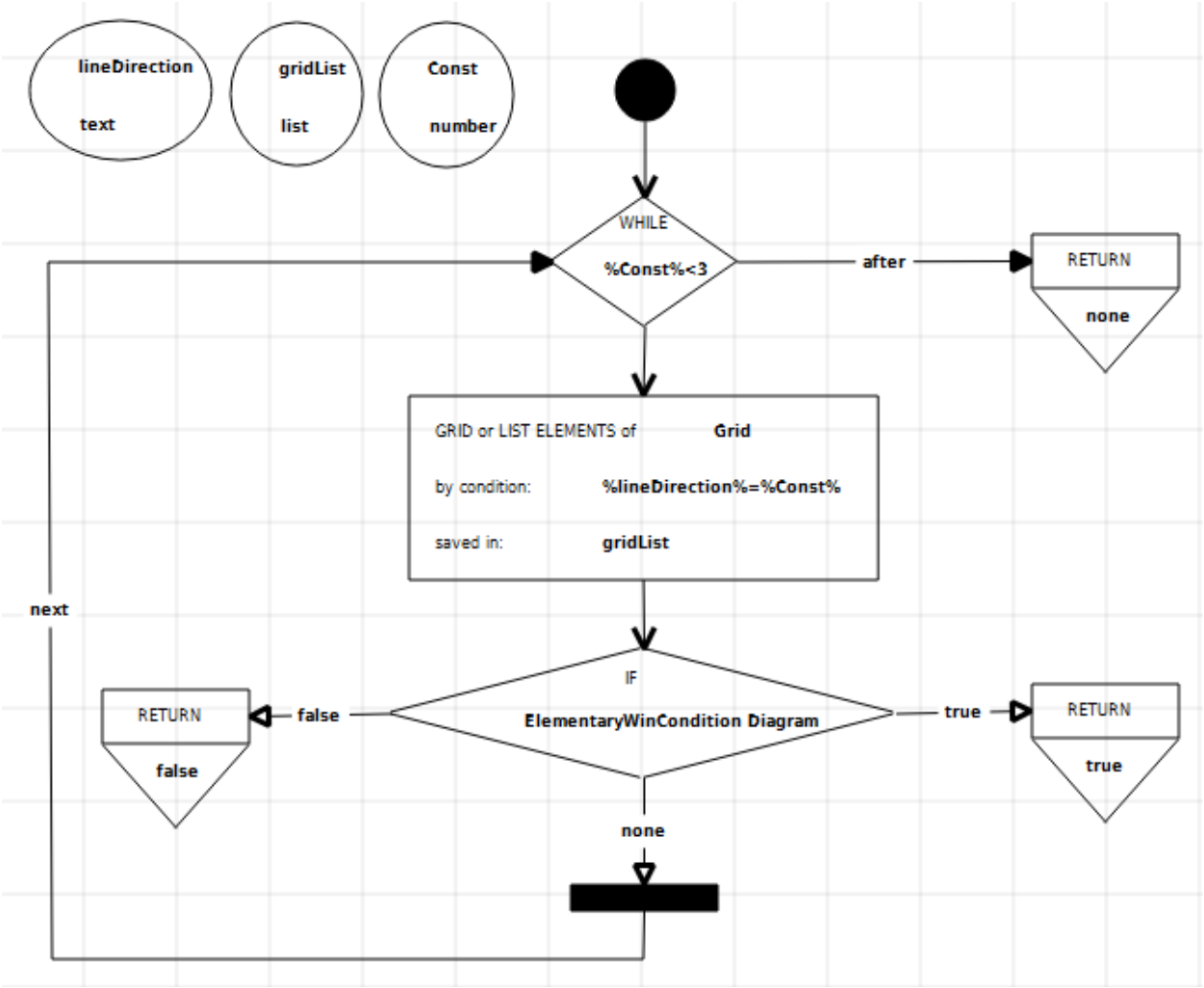

Рис. 5.3. Диаграмма логического условия, проверяющего условие победы по всем строчкам и столбцам игрового поля.

Данная диаграмма после своего одного выполнения вернёт одно из трёх значений: «победа» (true), «проигрыш» (false) и «не определено» (none).

Сначала задается цикл «while», для которого счетчиком является переменная «Const». Этот цикл отрабатывается, пока значение счетчика («%Const%») будет меньше 3. В теле цикла в переменную «gridList» сохраняются все элементы строки или столбца с порядковым номером, равным значению счетчика. Выбор между строкой и столбцом делается в зависимости от значения переменной «lineDirection» на момент начала выполнения данной диаграммы: если оно окажется равным «I», то будут выбираться строки, а если же «J», то столбцы. Далее для полученного списка элементов («gridList») проверяется элементарное логическое условие на победу (т. е. если все элементы помечены крестиком, то выиграли «крестики»; если все элементы отмечены ноликами, то выиграли «нолики»; иначе же считается неопределенным состоянием). Если результатом этого условия будет победа или проигрыш, то диаграмм вернёт соответствующее состояние игры. Иначе счетчик будет автоматически увеличен и тело цикла повторится еще раз. Если цикл был полностью завершен, а никакого значения возвращено еще не было, то возвращается неопределенный статус игры (none).

Заключение

Таким образом, в рамках данной курсовой мною было сделано следующее:

1. Рассмотрены существующие среды визуального программирования и некоторые визуальные предметно-ориентированные языки, в том числе для разработки мобильных приложений;
2. Придуман и реализован в среде QReal визуальный язык для создания мобильных приложений с нетривиальной логикой поведения, а именно для создания онлайн игр с игровым полем между двумя игроками;
3. Также реализован в среде QReal второй вариант визуального языка разработки мобильных приложений, предложенный А.Н. Тереховым;
4. Поддержана генерация кода под платформу Ubiq Mobile по диаграммам форм, то есть теперь есть возможность при помощи визуального языка описать программу и сгенерировать приложение, которое будет представлять некоторый прототип итогового приложения;
5. На разработанном визуальном языке написано мобильное приложение, реализующее онлайн игру в крестики-нолики с одним игроком.

Список литературы

1). A software engineering experiment in software component generation / R. B. Kieburtz [и др.] // Proceedings of the 18th international conference on Software engineering. — IEEE Computer Society. 1996. — С. 542—552.
2). Gray J., Karsai G. An examination of DSLs for concisely representing model traversals and transformations //System Sciences, 2003. Proceedings of the 36th Annual Hawaii International Conference on. — IEEE. 2003. —10 pp.
3). Kelly S., Tolvanen J.-P. Visual domain-specific modeling: Benefits and experiences of using metaCASE tools //International Workshop on Model Engineering, at ECOOP. — 2000.
4). Onossovski V., Terekhov A. «Ubiq Mobile – a New Universal Platform for Mobile Online Services» // Proceedings of 6th Seminar of Finish-Russian University Cooperation (FRUCT) Program. Helsinki, 2009.
5). А.Н. Терехов, Т.А. Брыксин, Ю.В. Литвинов и др., Архитектура среды визуального моделирования QReal. // Системное программирование. Вып. 4. СПб.: Изд-во СПбГУ. 2009, С. 171-196
6). Кузенкова А.С., Дерипаска А.О., Литвинов Ю.В., Поддержка метамоделирования в среде визуального программирования QReal // Материалы межвузовского конкурса-конференции студентов, аспирантов и молодых ученых Северо-Запада «Технологии Microsoft в теории и практике программирования». Спб.: Изд-во СПбГПУ, 2011. С. 100-101
7). Паронджанов В.Д. «Как улучшить работу ума».
8). Паронджанов В.Д. «Язык Дракон. Краткое описание»
9). Ubiq Mobile: Платформа для разработки мобильных сервисов, URL: ubiqmobile.com, Дата обращения: 20.05.2013
10). QReal: Добро пожаловать, URL: qreal.ru, Дата обращения: 20.05.2013
11). Визуальный язык ДРАКОН, URL: drakon.su, Дата обращения: 01.03.2013
12). DRAKON, URL: en.wikipedia.org/wiki/DRAKON, Дата обращения: 01.03.2013
13). HiAsm, URL: ru.wikipedia.org/wiki/Hiasm, Дата обращения: 02.03.2013
14). Хакер, Визуальный программинг: Конструируем приложения с помощью HiAsm, URL: www.xakep.ru/post/54779/default.asp, Дата обращения: 02.03.2013
15). HiAsm, URL: www.hiasm.com, Дата обращения: 02.03.2013
16). Scratch (programming language), URL: en.wikipedia.org/wiki/Scratch_(programming_language), Дата обращения: 03.03.2013
17). Scratch: Создавайте истории, игры и мультфильмы; Делитесь с людьми по всему миру, URL: scratch.mit.edu, Дата обращения: 03.03.2013
18). Учитесь со Scratch: Компьютерные игры изнутри, URL: setilab.ru/scratch, Дата обращения: 03.03.2013
19). Скретч, URL: letopisi.ru/index.php/%D0%A1%D0%BA%D1%80%D0%B5%D1%82%D1%87, Дата обращения: 03.03.2013
20). Scratch, URL: progopedia.com/language/scratch, Дата обращения: 03.03.2013
21). Welcome to MIT App Inventor, URL: appinventor.mit.edu, Дата обращения: 04.03.2013
22). КОМПЬЮТЕРРАLAB: Визуальное программирование для Google Android, URL: old.computerra.ru/terralab/softerra/547989, Дата обращения: 04.03.2013
23). App Inventor, URL: ru.wikipedia.org/wiki/App_Inventor, Дата обращения: 04.03.2013
24). Stencyl, URL: en.wikipedia.org/wiki/Stencyl, Дата обращения: 05.03.2013
25). Stencyl, URL: www.stencyl.com, Дата обращения: 05.03.2013
26). GameSalad, URL: en.wikipedia.org/wiki/GameSalad, Дата обращения: 06.03.2013
27). GameSalad, URL: gamesalad.com, Дата обращения: 06.03.2013
28). DragonRAD, URL: en.wikipedia.org/wiki/DragonRAD, Дата обращения: 08.03.2013
29). DragonRAD, URL: www.dragonrad.com, Дата обращения: 08.03.2013
30). Smartface, URL: en.wikipedia.org/wiki/Smartface, Дата обращения: 09.03.2013
31). Smartface App Studio, URL: developer.smartface.biz, Дата обращения: 09.03.2013
32). Smartface Platform, URL: www.mobinex.biz/smartface-platform.html, Дата обращения: 09.03.2013
33). ViziApps, URL: en.wikipedia.org/wiki/ViziApps, Дата обращения: 10.03.2013
34). ViziApps: Create Mobile Business Apps no coding, URL: www.viziapps.com, Дата обращения: 10.03.2013