Доктор технических наук
Валерий Алексеевич Жарков

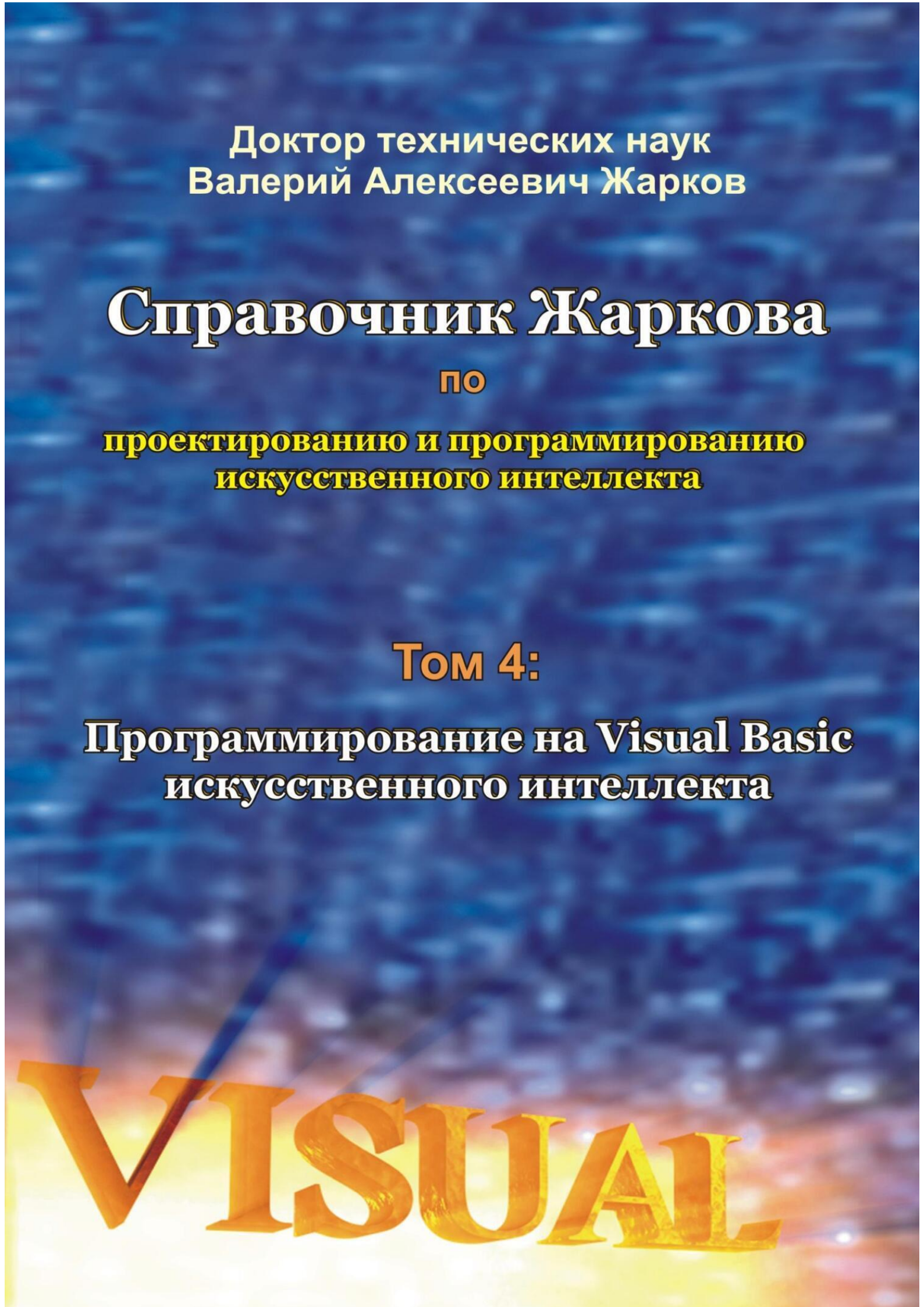
Справочник Жаркова

по

проектированию и программированию искусственного интеллекта

Том 4:

Программирование на Visual Basic искусственного интеллекта



Валерий Жарков

Справочник Жаркова по проектированию и программированию искусственного интеллекта. Том 4: Программирование на Visual Basic искусственного интеллекта

«Автор»

2022

Жарков В. А.

Справочник Жаркова по проектированию и программированию искусственного интеллекта. Том 4: Программирование на Visual Basic искусственного интеллекта / В. А. Жарков — «Автор», 2022

В серии книг “Справочник Жаркова по проектированию и программированию искусственного интеллекта” в нескольких томах собрано лучшее программирование искусственного интеллекта (ИИ). XVIII частей текстов приложений по ИИ на Visual Basic разделены на три тома. В данном томе приведены следующие части. Часть I. Краткие основы Visual Basic. Часть II. Учебная методология программирования игр и приложений с подвижными объектами. Часть III. Методология программирования искусственного интеллекта в карточных играх. Часть IV. Методология программирования искусственного интеллекта в играх по сборке картины из её частей. Часть V. Методология программирования искусственного интеллекта в играх типа змейки, которая поедает куски пищи. Часть VI. Методология программирования искусственного интеллекта в играх типа “Тетрис” по сборке сплошных полос из разнообразных фигур. Часть VII. Методология программирования искусственного интеллекта в играх в “Крестики-нолики” для Игрока с Компьютером и двух Игроков.

© Жарков В. А., 2022
© Автор, 2022

Содержание

Валерий Жарков
Справочник Жаркова по проектированию и программированию искусственного интеллекта. Том 4: Программирование на Visual Basic искусственного интеллекта

Спасибо за помощь в написании моих книг
моей жене (Жаркова Клавдия Петровна)

Введение

Это первый и единственный в мире “Справочник Жаркова по проектированию и программированию искусственного интеллекта” из нескольких томов по методологии разработки искусственного интеллекта в двухмерных и трёхмерных играх и приложениях со звуковым сопровождением для настольных компьютеров, ноутбуков и планшетов на основе популярного, совершенного и перспективного языка программирования высокого уровня Visual Basic самой мощной в мире в области программирования корпорации Microsoft (США).

Искусственный интеллект (ИИ) – это интеллектуальная компьютерная программа, способная разумно выполнять функции, задачи и действия, обычно характерные для разумных существ и свойственные человеческому интеллекту. ИИ в игре или приложении, например, в игре между Компьютером и Человеком, умеет не только проигрывать, но и выигрывать у хорошего Игрока-человека с визуализацией результатов выигрыша. Хорошо известно, что компьютер с ИИ обыгрывает в шахматы любого гроссмейстера. Компьютер с ИИ также легко обыгрывает многих хороших игроков в карты. Если программа в виде ИИ размещена, например, в роботе или другом устройстве, то после того, как ИИ решил заданную ему задачу, ИИ выдаёт команду на выполнение устройством определённого действия. При программировании ИИ важно правильно подобрать среду разработки ИИ и язык программирования.

Среда разработки (иначе, платформа, студия) Visual Studio (или коротко VS) для визуального объектно-ориентированного проектирования приложений даёт уникальную возможность быстро разрабатывать высокотехнологичные и наукоёмкие программные продукты с использованием ИИ, а также компьютерные игры с двухмерной и трёхмерной графикой также с использованием ИИ. Важно отметить, что на основе VS мы программируем не закрытые “чёрные ящики”, как это делают другие известные компьютерные фирмы, а мы создаём открытые любому пользователю (для постоянного дополнения и совершенствования) программы на базе самых мощных в мире алгоритмических языков высокого уровня Visual C#, Visual Basic и Visual C++. В данном чрезвычайно насыщенном томе (заменяющим несколько других книг) мы последовательно и всесторонне, идя от простого к сложному, излагаем методологию программирования ИИ в играх и приложениях с использований двухмерных и трёхмерных изображений.

Наша основная цель – дать читателю ту информацию, которую он больше нигде не найдёт. Поэтому мы не будем дублировать известные книги по языку программирования Visual Basic и давать подробные объяснения по теории языка VB. Если у читателя возникнуть вопросы, он легко отыщет книгу по данному языку (некоторые полезные по данной тематике книги и журналы с сайта ZharkovPress.ru приведены в нашем списке литературы) и там найдёт ответ,

так как терминология по всем тематикам у нас общая. Мы будем давать лишь краткие пояснения в виде комментариев в программах, чтобы начинающий пользователь постепенно осваивал базовые дисциплины программирования ИИ на языке VB, по возможности не используя другие книги; опытному пользователю также будут полезны эти пояснения, поскольку книги по разработке ИИ на основе новых версий языка Visual Basic в мире ещё не изданы. К достоинствам нашей книги, рассчитанной на широкий круг новичков и опытных специалистов, мы относим практическую направленность, простоту изложения (без описания сложных теорий, но давая ссылки на книги, в которых эти сложные теории можно изучить), наличие подробных методик и пошаговых инструкций, большое количество примеров и иллюстраций. Теперь читателям может не потребоваться изучать сложные теоретические книги, посещать длительные и дорогостоящие учебные курсы и покупать много отдельных книг. Автор это сделал за них. Читателю необходимо лишь открыть данную книгу в интересующем его разделе (мало кто будет изучать книгу от корки до корки, хотя это и желательно) и по аналогии с разделом (по принципу: делай, как я) самостоятельно программировать ИИ в практическом приложении или игре на основе VS. И именно при проектировании ИИ в своём конкретном и профессионально интересном приложении или игре (а не отвлечённых примеров в других книгах) читатель будет изучать базовые дисциплины по данной тематике. Создавая ИИ в своём приложении или игре по методике данной и других наших книг и журналов из списка литературы и с сайта ZharkovPress.ru (а также используя справку Help из Visual Studio, как правило, заменяющей все учебники по всем языкам), читатель сможет в одиночку работать за конструктора, технолога, математика и программиста одновременно (при разработке практических приложений) или за сценариста, режиссёра, оператора, дизайнера, художника, музыкального редактора и программиста одновременно (при разработке игр) и сэкономить недели упорного труда. Если в начальных главах серии книг инструкции по разработке ИИ в играх и приложениях на базе VS подробны (в интересах новичков), то инструкции в каждой последующей главе мы даём всё короче и короче, чтобы не повторяться и экономить место в книге.

Приводим краткое содержание всех XVIII частей из трёх томов по программированию ИИ на Visual Basic.

Приводим краткое содержание VII частей первого тома по программированию ИИ на Visual Basic.

Часть IV. Методология программирования искусственного интеллекта в играх по сборке картины из её частей

Глава 13. Методика программирования искусственного интеллекта в игре по сборке разрезанной картины заменой местами её масштабируемых частей

Часть V. Методология программирования искусственного интеллекта в играх типа змейки, которая поедает куски пищи

Глава 14. Методика программирования искусственного интеллекта в игре типа змейки, которая поедает куски пищи и за счёт этого увеличивается по длине, на основе одного файла

Глава 15. Методика программирования искусственного интеллекта в игре типа змейки, которая поедает куски пищи и за счёт этого увеличивается по длине, на основе четырёх файлов

Часть VI. Методология программирования искусственного интеллекта в играх типа “Тетрис” по сборке сплошных полос из разнообразных фигур

Глава 16. Методика программирования искусственного интеллекта в игре по сборке сплошных прямых полос из сторон фигур 12 типов

Глава 17. Методика программирования искусственного интеллекта в игре по сборке сплошных прямых полос из сторон фигур 7 типов с формами для имени игрока, результатов и справкой по игре

Часть VII. Методология программирования искусственного интеллекта в играх в “Крестики-нолики” для Игрока с Компьютером и двух Игроков

Глава 18. Методика программирования искусственного интеллекта в игре в “Крестики-нолики”

Заключение

Список литературы

Многие приложения и игры в книге основаны на программах, или разработанных корпорацией Microsoft, или опубликованных на сайте корпорации Microsof. Поэтому эти программы являются очень мощными и могут быть использованы не только при разработке ИИ в самых разнообразных играх, но и на практике для разработки различных приложений. Структура книги продумана таким образом, чтобы читатели могли создавать на профессиональном уровне (по методологиям и программам из данной и предыдущих наших книг и журналов с сайта ZharkovPress.ru) свои приложения, игры и открытые графические и вычислительные системы с применением двухмерных и трёхмерных изображений и звуковых эффектов, могли вводить разнообразные исходные данные и на выходе приложения или игры получать с использованием ИИ те результаты, которые необходимы именно им и характерны для их профессиональных или непрофессиональных интересов.

Книга предназначена для всех желающих быстро изучить основы программирования искусственного интеллекта в разнообразных двухмерных и трёхмерных компьютерных играх и приложениях на базе популярного, совершенного и перспективного (в мире программирования) языка высокого уровня **Visual Basic** последних версий для настольных компьютеров, ноутбуков и планшетов, на этих основах сразу же проектировать ИИ в сложных играх и приложениях и применять ИИ на практике или на отдыхе в разнообразных сферах профессиональной и непрофессиональной деятельности. Также адресована начинающим и опытным пользователям, программистам любой квалификации, а также учащимся и слушателям курсов, студентам, аспирантам, учителям, преподавателям и научным работникам.

В следующем томе автор продолжит описывать программирование ИИ в следующих играх и приложениях.

Вопросы, замечания и предложения по тематике данной книги можно направлять по email:

valery-zharkov@mtu-net.ru

Автор: доктор технических наук Жарков Валерий Алексеевич (город Москва)

Часть I. Краткие основы Visual Basic

Глава 1. Основные определения книги

1.1. Классификация компьютерных игр

Компьютерная игра (computer game) – это взаимодействие человека с компьютером по определённым правилам с целью обучения, тренировки или развлечения.

Правила игры задет алгоритм, который реализуется при помощи игровой программы (game program), написанной на различных языках программирования для разнообразных компьютеров и мобильных устройств, работающих под управлением различных операционных систем.

Во время игры на экране воспроизводится игровая ситуация, один или несколько игроков анализируют её и реагируют при помощи устройств ввода, для – это кнопки джойстика и кнопки клавиатуры. Компьютерные игры, хорошо спроектированные, развивают профессиональные, познавательные и художественные способности человека.

Большое количество игр для настольных компьютеров и мобильных устройств можно классифицировать по различным признакам и типам в зависимости от преследуемых целей.

С классификацией по типу операционной системы мы определились – это Microsoft Windows.

С классификацией по типу языка программирования высокого уровня мы также определились – это Microsoft Visual Basic из среды разработки Microsoft Visual Studio последних версий.

С классификацией по типу размерности изображения (двухмерные и трёхмерные) также ясно. В данной серии книг мы спрограммируем ИИ в играх и приложениях как без использования технологии DirectX, так и с использованием DirectX и XNA последних версий, которые поддерживает Microsoft.

Далее, по сюжетам и целям, различают несколько типов игр.

Логические – игры, близкие по содержанию к обычным головоломкам (которые называются также пазлами, от английского слова puzzle – головоломка) и математическим упражнениям, а также компьютерные варианты различных настольных игр (шахматы, шашки, нарды и т.д.). В головоломке или математическом упражнении мотивация, основанная на азарте, сопряжена с желанием одного игрока решить задачу быстрее другого игрока. А в игре типа шахмат мотивация основана на желании обыграть компьютер, доказать своё превосходство над машиной.

Аркадные (arcade game) – игры с путешествиями и приключениями, сюжет которых, как правило, слабый линейный, игроку не нужно особо задумываться, а нужно только, управляя компьютерным персонажем или транспортным средством, быстро передвигаться, стрелять и уничтожать движущиеся объекты. При этом игрок набирает очки, зарабатывает дополнительные “жизни”, переходит на следующий более высокий уровень данной игры. В аркаде игрок в большей степени ориентирован не на процесс игры, а на результат. К этому типу относятся также разного рода “бегалки”, “стрелялки” (shooter) и экшны (action).

Игры на быстроту реакции. В этих играх игроку нужно проявлять ловкость и быстроту реакции при нажатии кнопок , уметь нажать кнопку на опережение с учётом перемещения объектов. Эти игры во многом похожи на аркады, но, в отличие от последних, обычно не имеют

сюжета и сценария, абстрактны и не связаны с реальной действительностью. Но здесь также есть и азарт, и желание набрать больше очков, чем другой игрок.

Приключенческие или сюжетные (adventure game) – игры, в которых игрок действует по определённому сюжету и сценарию приключенческого, сказочного, фантастического или мистического содержания.

Стратегические или позиционные (extensive game) – игры, в которых игрок управляет армией, городом, страной и другими масштабными объектами.

Игры-имитации моделируют управление игроком автомобилем, вертолётом, самолётом, ракетой и другими подобными машинами.

Спортивные игры – компьютерные реализации разнообразных популярных спортивных игр (футбол, гольф и т.д.).

Традиционно азартные игры – это компьютерные варианты разнообразных карточных игр, игры в кости, рулетки, имитаторы игровых автоматов и других игр казино, существовавших задолго до изобретения компьютера.

Интерактивные (interactive game) – игры, имеющие чётко выраженную форму диалога игрока с компьютером или мобильным устройством.

Сетевые (network game) – игры, в которых могут участвовать несколько партнёров, взаимодействующих между собой через глобальную сеть Интернет или через локальную сеть LAN – Local Area Network.

Учебные (training game) – игры, способствующие развитию профессиональных возможностей человека.

По психологическому признаку игры условно делятся на ролевые и не ролевые. Ролевая игра (RPG – Role Playing Game) – это игра, в которой игрок принимает на себя роль компьютерного (конкретного видимого на экране или воображаемого) персонажа.

Известны также игры, не имеющие решения (no-solution game), и многие другие.

Современные игры сочетают свойства различных типов, имеют хорошее мультимедийное оформление, сложную мотивацию решения загадок, разветвлённую сюжетную линию, поэтому часто сложно определить, к какому же типу относится игра, и нужно выбрать наиболее подходящий тип.

Захватывающие игры для мощных мультимедийных настольных компьютеров и графических станций используют огромные возможности мультимедиа и погружают игрока в виртуальную реальность. Игры для мобильных устройств, описанные в наших предыдущих книгах, можно активно пользоваться там (например, в транспорте), где настольный компьютер недоступен.

В данной книге мы будем описывать игры для настольных компьютеров, ноутбуков и планшетов в соответствии с этой классификацией.

1.2. Требуемое программное обеспечение

Чтобы программировать искусственный интеллект в двух- и трёхмерных играх и приложениях, а также графику на основе Visual Studio, на нашем настольном компьютере (ноутбуке или планшете) должны быть установлены следующие программные продукты.

1. Среда разработки (называемая также как платформа, студия, пакет и т.д.) Visual Studio или сокращённо VS корпорации Microsoft. Системные требования к компьютеру сформулированы на сайте этой корпорации. Отметим, что имеются бесплатные версии VS, например, Express-версии и Бета-версии, которые можно загрузить с сайта Microsoft. Установка на компьютер и настройка любой версии VS подробно описаны в наших книгах с сайта ZharkovPress.ru.

VS имеет исполнительную среду CLR (Common Languege Runtime), общую для всех основных языков программирования. Поэтому впервые появилась уникальная возможность соединять (в разрабатываемом нами приложении) программы, написанные на различных языках (соответствующих спецификации CLR). Программа, написанная на любом языке или на нескольких различных языках, сначала компилируется в промежуточный язык MSIL (Microsoft Intermediate Language) и только затем преобразуется в машинный код. Такое двух шаговое выполнение программ позволяет достичь хорошей межъязыковой совместимости при высоком быстродействии.

Чтобы читателю легче было изучить и применить на практике самый популярный и перспективный (в мире программирования) язык Visual Basic, он дан в сравнении с другими языками, и базовые сведения об этих языках (Visual Basic, C#, C++, J#) сведены в таблицы (в наших книгах [Литература]). Программы, написанные на предыдущих версиях Visual Studio после конвертирования пригодны и для новой версии Visual Studio.

Графический интерфейс в операционных системах Windows, а также вся работа с графикой в Visual Studio, а следовательно, и в Visual Basic основаны на использовании интерфейса устройств графики GDI+ (Graphics Device Interface) Этот интерфейс GDI+ для двухмерной графики подробно описан в наших предыдущих книгах из списка литературы.

Кроме двухмерной графики, в книгах с сайта ZharkovPress.ru по созданию трёхмерных графических систем описано применение технологии DirectX для компьютерных игр и приложений.

2. Если планируется применять DirectX и/или XNA, то с сайта корпорации Microsof необходимо бесплатно загрузить последнюю версию технологии DirectX в виде программного продукта DirectX Software Development Kit (SDK) и XNA. Каждые несколько месяцев на сайте выставляется новая редакция DX и XNA. Более подробно о DX и XNA описано в справке от Microsoft. Отметим, что на сайте корпорации Microsoft программный продукт DirectX для разработки трёхмерных изображений находится в разделе Windows и далее в подразделе Technologies. Следовательно, DirectX (или коротко DX) авторы называют технологией, так и мы будем называть (в случае применения).

Отметим, что в данной серии книг будут разработаны игры и приложения с использованием искусственного интеллекта как без применения DirectX и XNA, что делает такие игры и приложения более универсальными, так и с применением DirectX и XNA.

Глава 2. Методика разработки приложений для выполнения расчётов с эффектами анимации

2.1. Алгоритм ввода-вывода данных

Сначала общими усилиями создадим приложение в виде традиционного калькулятора для сложения и вычитания чисел. А затем постепенно будем усложнять и увеличивать количество математических функций в приложении, и в изучении базовых дисциплин и методик расчётов, с использованием эффектов анимации, идти от простого к сложному.

Алгоритм сложения вещественных чисел в нашем первом калькуляторе стандартен:

в первое окно вводим первое слагаемое;

во второе окно второе слагаемое;

щёлкаем кнопку со знаком равенства;

в третьем окне получаем результат сложения (сумму).

Примеры в этой главе (и все последующие примеры) позволяют достичь:

1) учебную цель, чтобы научить начинающего программиста (например, инженера – практика, который никогда раньше не применял этот алгоритмический язык), или специалиста любого профиля, вводу исходных данных и выводу результатов расчёта, а также решению различных задач при помощи форм и форм (точнее, при помощи принципов визуального программирования) и объектно-ориентированного программирования (ООП) на языке высокого уровня Visual Basic;

2) практическую цель, чтобы по аналогии с нашим приложением читатели могли быстро создать персональный калькулятор (а далее, вычислительную систему и многие другие приложения) для расчёта таких математических, экономических и других функций, которые дополнят функции настольного калькулятора (и калькулятора операционной системы, например, Windows) и других известных систем компьютерной математики.

2.2. Проект приложения

Некоторые этапы разработки данного приложения-калькулятора в дальнейшем (в последующих главах) при создании других приложений будут повторяться. Поэтому в этой главе, в интересах читателей, мы дадим подробные инструкции выполнения всех основных этапов разработки, чтобы в последующих главах эти же этапы описывать кратко со ссылкой на эту главу (и не повторяться).

Создаём проект в пакете (или комплексе) Visual Basic среды разработки Visual Studio (или предыдущих версий среды) по следующей типичной схеме.

1. Запускаем среду (называемую также: среда разработки, студия, платформа, пакет и т.п.) Visual Studio и щёлкаем значок New Project (или File, New, Project).

2. В панели New Project (рис. 2.1) в окне Project Types выбираем тип проекта Visual Basic, Windows, Windows и проверяем, чтобы в окне Templates был выделен (как правило, по умолчанию) шаблон Windows Forms Application; в окне Name записываем имя проекта, например, Calculator. В расположенном ниже окне Location мы видим каталог (папку Projects с текстом пути к этой папке), в котором будет создан данный проект. Мы можем щёлкнуть по стрелке в окне Location (или кнопку Browse) и в появившемся списке выбрать или записать другой путь и другую папку, которую здесь же можем создать.

3. Если создаваемое нами решение (Solution) будет состоять из одного проекта, и мы не планируем разрабатывать другие проекты в рамках этого решения (как в данном случае), то

можно удалить (если он установлен) флажок Create Directory for Solution (рис. 2.1) с целью упрощения построения программы.

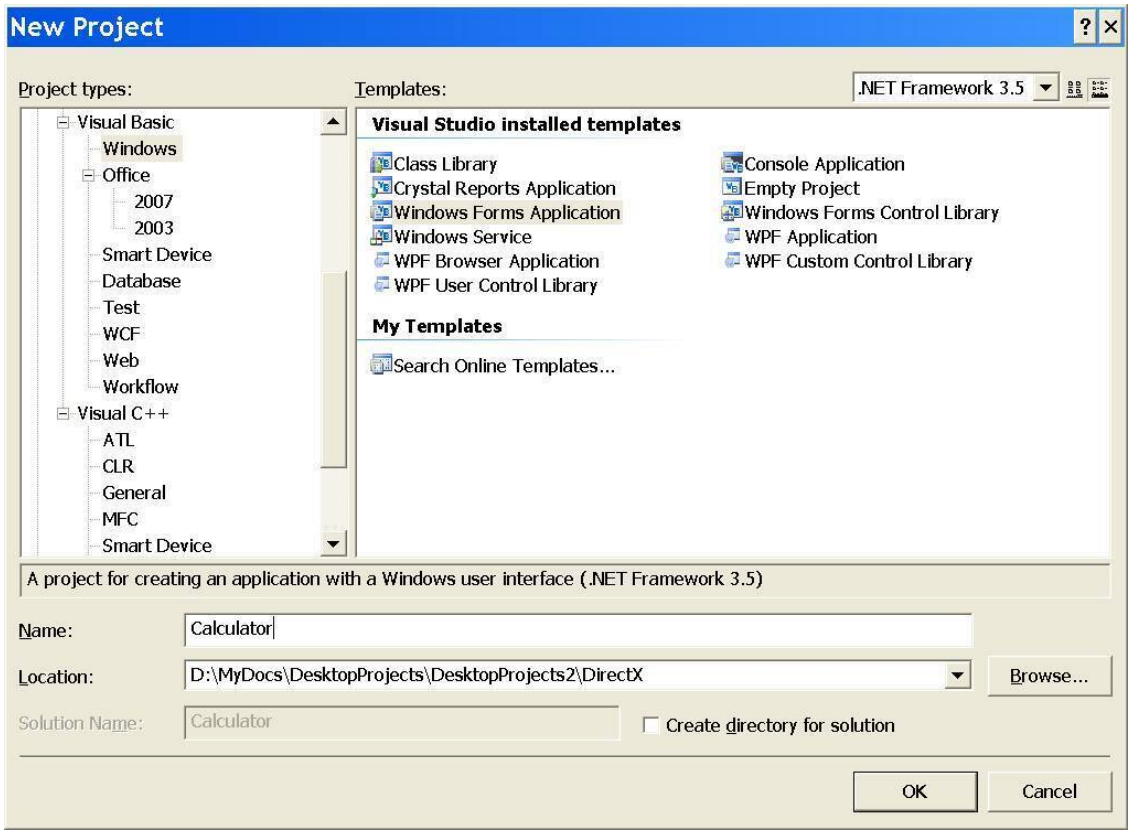


Рис. 2.1. В окне Project Types выделяем тип Visual Basic, Windows.

4. В панели New Projects (рис. 2.1) щёлкаем OK. Visual Basic создаёт проект приложения и выводит форму в виде Form1 в режиме проектирования (иначе, дизайна или редактирования) с вкладкой Form1.vb[Design], при помощи которой далее можно будет открывать эту форму.

5. Несмотря на то, что мы не написали ещё ни одной строчки программного кода, приложение уже должно работать в любой версии Visual Studio. Для проверки работоспособности приобретенной нами Visual Studio выполняем построение программы:

если мы создаём решение (Solution) из нескольких проектов, то в меню Build выбираем Build Solution или щёлкаем значок с изображением трёх вертикальных стрелок, или нажимаем клавиши Ctrl+Shift+B; напомним, что на главное меню значки переносятся с панели Tools, Customize, а удаляются после нажатия клавиши Alt, захвата и смещения значка (как в текстовом процессоре Microsoft Word);

если мы создаём решение (Selection) из одного проекта (как в нашем случае), то можно выбрать Build, Build Selection или щёлкнуть значок с изображением двух вертикальных стрелок; в меню Build в наименовании команды Build Selection вместо слова Selection будет имя нашего проекта, в данном случае Calculator.

Мы должны увидеть в выходном окне Output на вкладке Output (рис. 2.2) сообщение компилятора VB об успешном построении без предупреждений и ошибок:

```
Build complete – 0 errors, 0 warnings
=== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ===
```

6. Если ошибок нет, то запускаем программу на выполнение: в меню Debug выбираем Start Without Debugging (или нажимаем клавиши Ctrl+F5) или Start (клавиша F5), или щёлкаем по значку с изображением восклицательного знака.

Поверх формы Form1 в режиме проектирования должна появиться форма Form1 в режиме выполнения, которую можно захватить мышью и переместить в другое место экрана. На этой форме щёлкаем значок “x” (Close – Закрыть) или делаем правый щелчок и в контекстном меню выбираем Закрыть.

Основная проверка работоспособности Visual Studio закончена, и мы можем приступить к дальнейшей работе (согласно сформулированному выше алгоритму).

2.3. Методика проектирования формы

Для предварительного (пока без внесения элементов управления с панели инструментов Toolbox, рис. 2.2) визуального графического проектирования формы выполняем следующие типичные шаги нашей инструкции.

1. Панель Properties (Свойства) со свойствами формы Form1 (рис. 2.3) мы можем вызвать (если она не появилась автоматически) при помощи команды View, Properties Window и разместить в виде отдельной панели в любом месте экрана, а можем сделать в виде вкладки более общей панели. Если в панели Properties нет заголовка Form1, то просто щёлкаем внутри формы, или там же делаем правый щелчок и в контекстном меню выбираем Properties. В панели Properties далее мы всегда будем подразумевать одноименную вкладку Properties (если не оговаривается особо, например, вкладка событий Events). После изменения (заданного по умолчанию) свойства мы должны дать программе команду внести это изменение в форму или элемент управления; для этого щёлкаем по выделенному имени свойства или нажимаем клавишу Enter. Если не видна форма (форма) Form1 в режиме проектирования, то на рабочем столе щёлкаем вкладку Form1.vb[Design] или в панели Solution Explorer (Исследователь-Проводник Решения), вкладка на рис. 2.3, дважды щёлкаем по имени файла Form1.vb.

2. В панели Properties с заголовком Form1 (рис. 2.3) щёлкаем (выделяем) слово Font и появившуюся кнопку с тремя точками. Мы увидим панель Font (рис. 2.4).

3. В панели Font (Шрифт) устанавливаем, например, шрифт Times New Roman и размер (Size) 14 для текста на форме и для текста на элементах управления, которые мы будем переносить на форму с панели Toolbox. В панели Font щёлкаем OK.

4. Чтобы изменить заголовок формы, в панели Properties в свойстве Text вместо слова Form1 записываем (или вставляем из буфера обмена: правый щелчок, Paste), например, Calculator; щёлкаем по слову Text (или нажимаем клавишу Enter), рис. 2.3.

5. При помощи свойства BackColor (рис. 2.3) мы можем установить (вместо установленного по умолчанию цвета Control) из списка другой цвет клиентской области Form1. Перечень этих цветов будет показан далее. Напомним, что в клиентскую область не входит верхняя полоска с текстом и граница формы.

6. При помощи свойства BackgroundImage (Фон), рис. 2.3, в качестве фона мы можем установить имеющийся у нас рисунок (форматов (.bmp), (.jpg), (.gif) и др.) или даже несколько рисунков, которые с заданным нами интервалом времени будут поочерёдно сменять друг друга в режиме анимации; затем на этом изменяющемся фоне можно размещать элементы управления (например, TextBox и Button) и компоненты (например, Timer), как будет показано далее.

7. Напомним, что на инструментальной панели Toolbox (рис. 2.2) имеются элементы управления Windows Forms и специализированные компоненты.

Для создания пользовательского интерфейса нашего приложения сначала на форме можно разместить элемент управления в виде рамки группы GroupBox, чтобы затем внутри этой рамки располагать другие элементы. Для этого на панели инструментов Toolbox (рис. 2.2)

щёлкаем элемент GroupBox (рамка группы). Выполняем щелчок на форме. Чтобы изменить название GroupBox с номером элемента, в панели Properties в свойстве Text вместо имеющегося там текста записываем (или вставляем из буфера обмена: правый щелчок, Paste), например, “Сложение чисел (Addition of numbers)”; щёлкаем по выделенному слову Text (или нажимаем клавишу Enter).

Затем в панели Properties можно установить другие свойства.

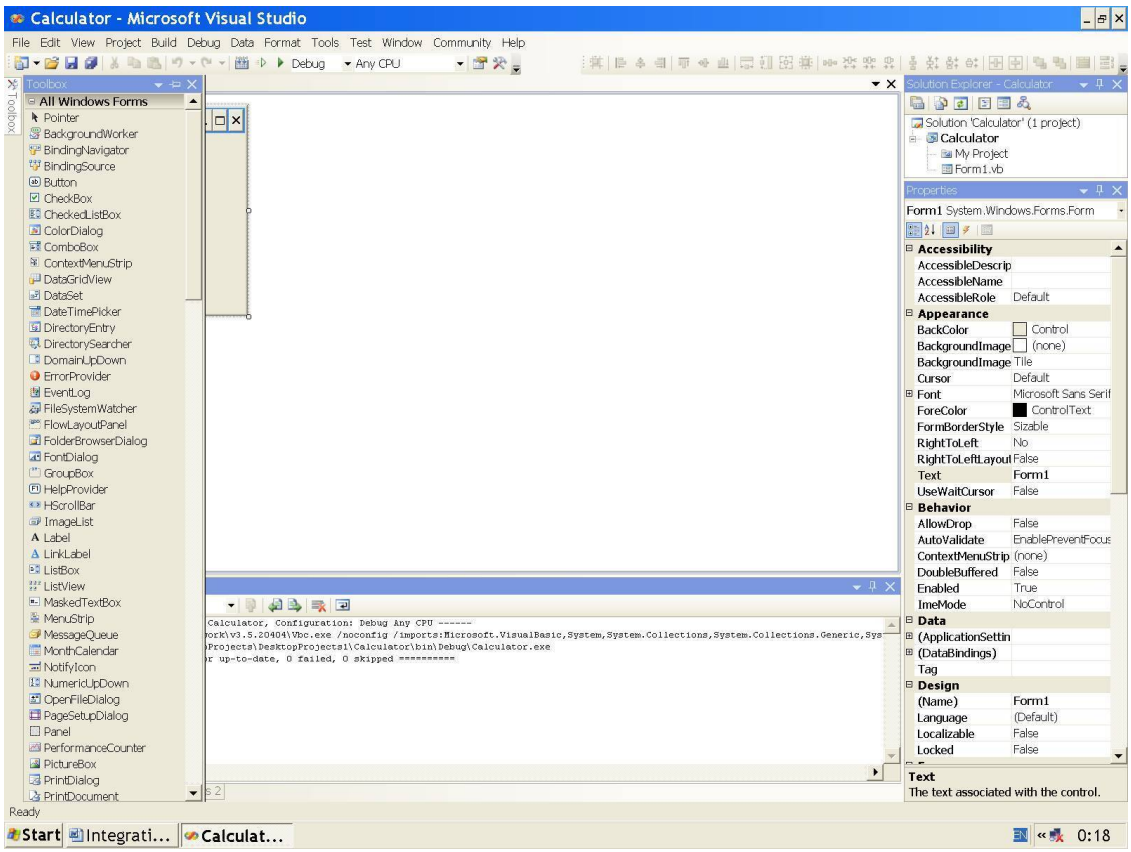


Рис. 2.2. Панель инструментов Toolbox.

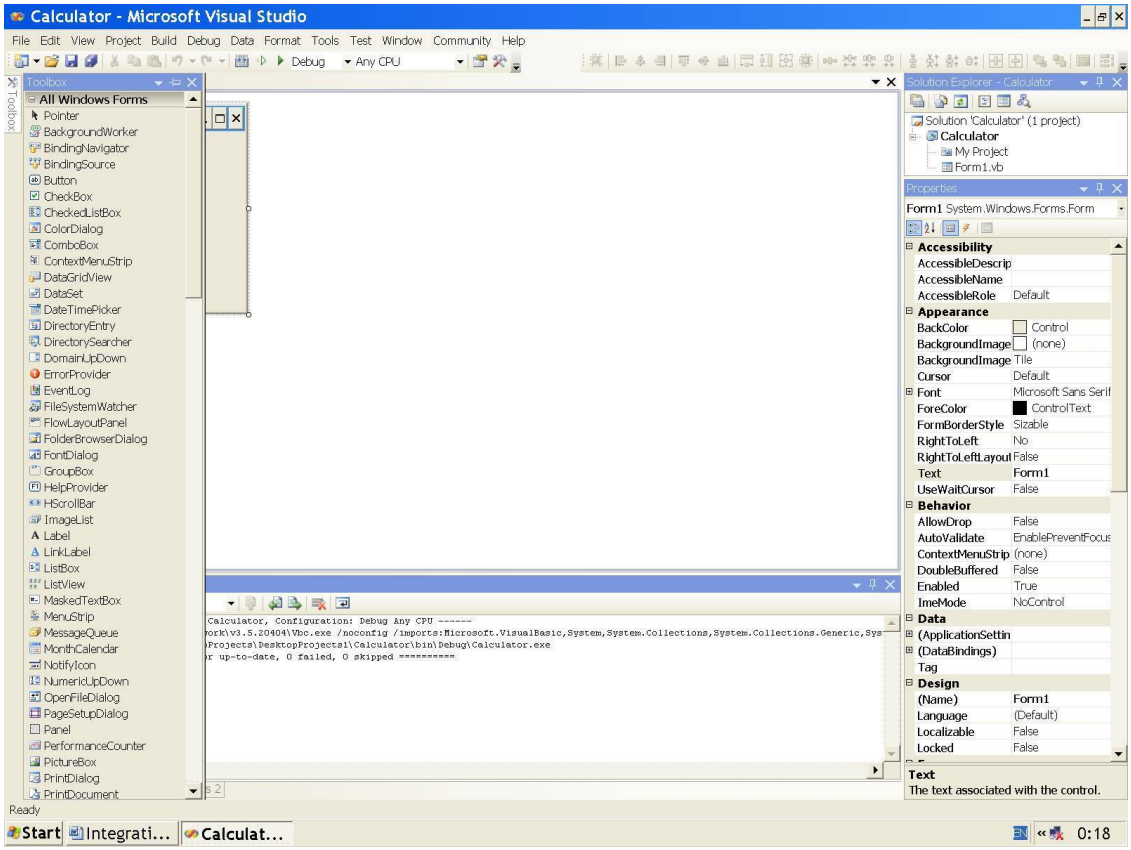


Рис. 2.3. Панель Properties.

8. Аналогично размещаем первое окно текста TextBox (для ввода первого слагаемого); в панели Properties появляется новый заголовок для этого элемента.

9. Размещаем второе окно текста TextBox для второго слагаемого и третье окно для суммы.

10. Чтобы пользователю было понятно, что означает каждое окно, около каждого окна вводим надпись в виде элемента управления Label. Например, первую надпись Label с номером элемента мы можем изменить или сразу с клавиатуры, или в панели Properties с заголовком для этого элемента в свойстве Text вместо имеющегося там текста записываем (или вставляем из буфера обмена: правый щелчок, Paste), например, “число (number)”.

11. Аналогично записываем текст выше второго и третьего окон, а между первым и вторым окнами – знак суммы “+”.

12. Чтобы получить результат сложения после щелчка кнопки со знаком равенства “=”, вводим эту кнопку на форму по обычной схеме: на панели Toolbox выбираем Button; щёлкаем на форме; надпись внутри этого элемента мы можем изменить или сразу с клавиатуры, или в панели Properties в свойстве Text вместо имеющегося там текста записываем (или вставляем из буфера обмена: правый щелчок, Paste), например, знак равенства “=”; щёлкаем по выделенному слову Text (или нажимаем клавишу Enter). В классе Form1 на вкладке Form1.Designer.vb (рис. 2.5) появились эти элементы управления. Форма приняла окончательный вид (рис. 2.6).

На приведённых рисунках видны отличия новой платформы Visual Studio от предыдущих версий этой же платформы, которые подробно описаны в наших предыдущих книгах [Литература]. Важно отметить, что все программы, которые далее мы будем разрабатывать, применимы для всех версий платформы Visual Studio.

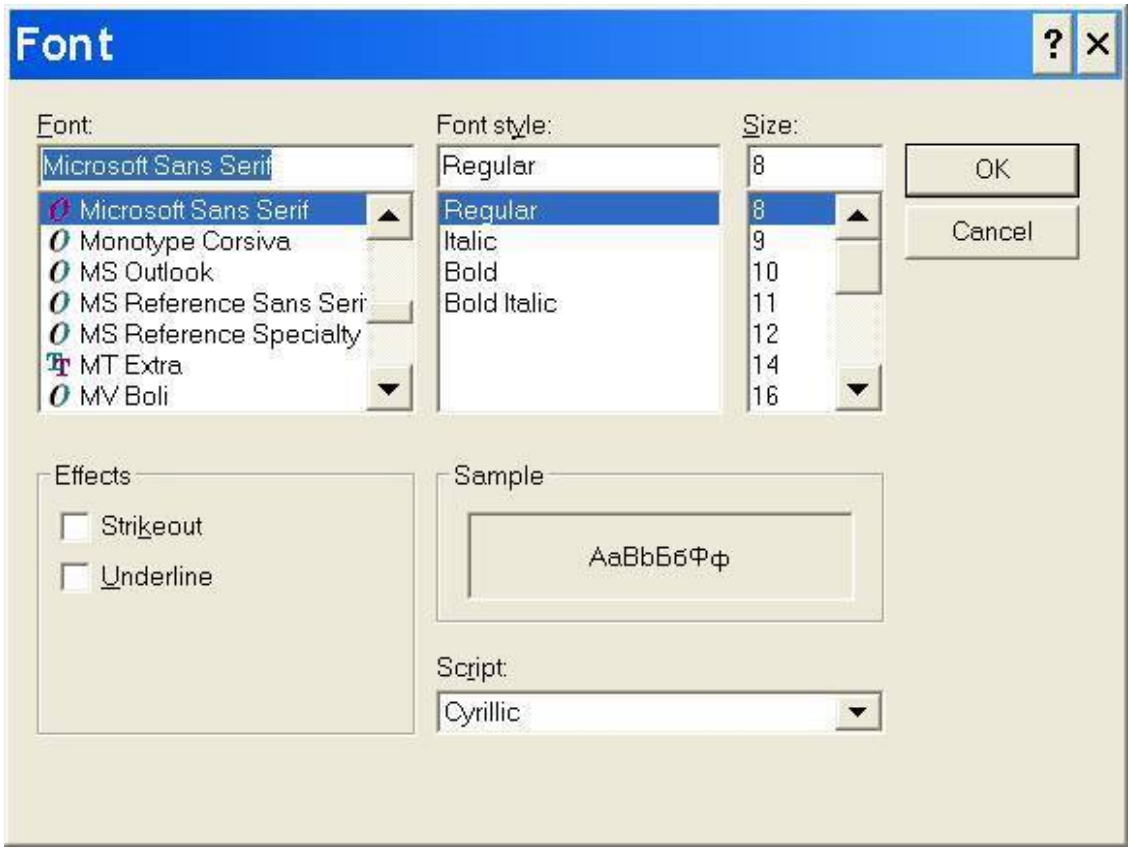


Рис. 2.4. В панели Font устанавливаем шрифт.

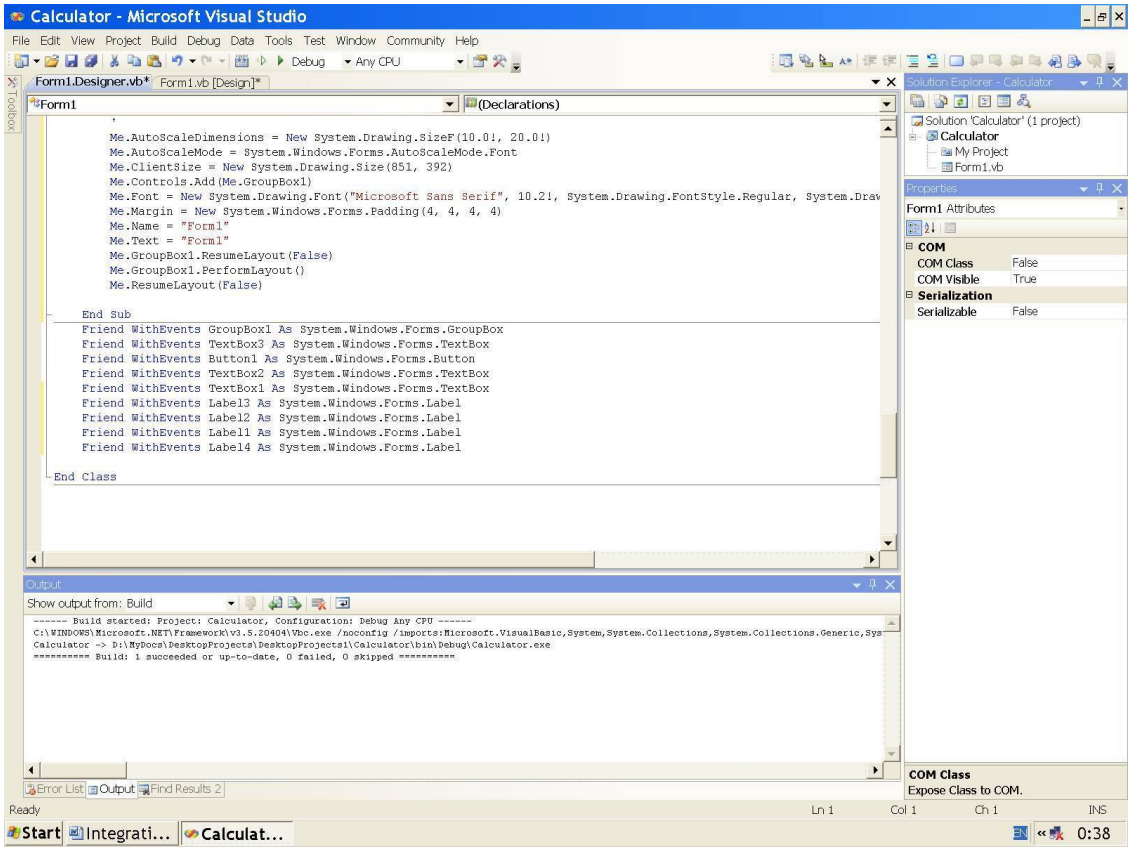


Рис. 2.5. Вкладка Form1.Designer.vb.

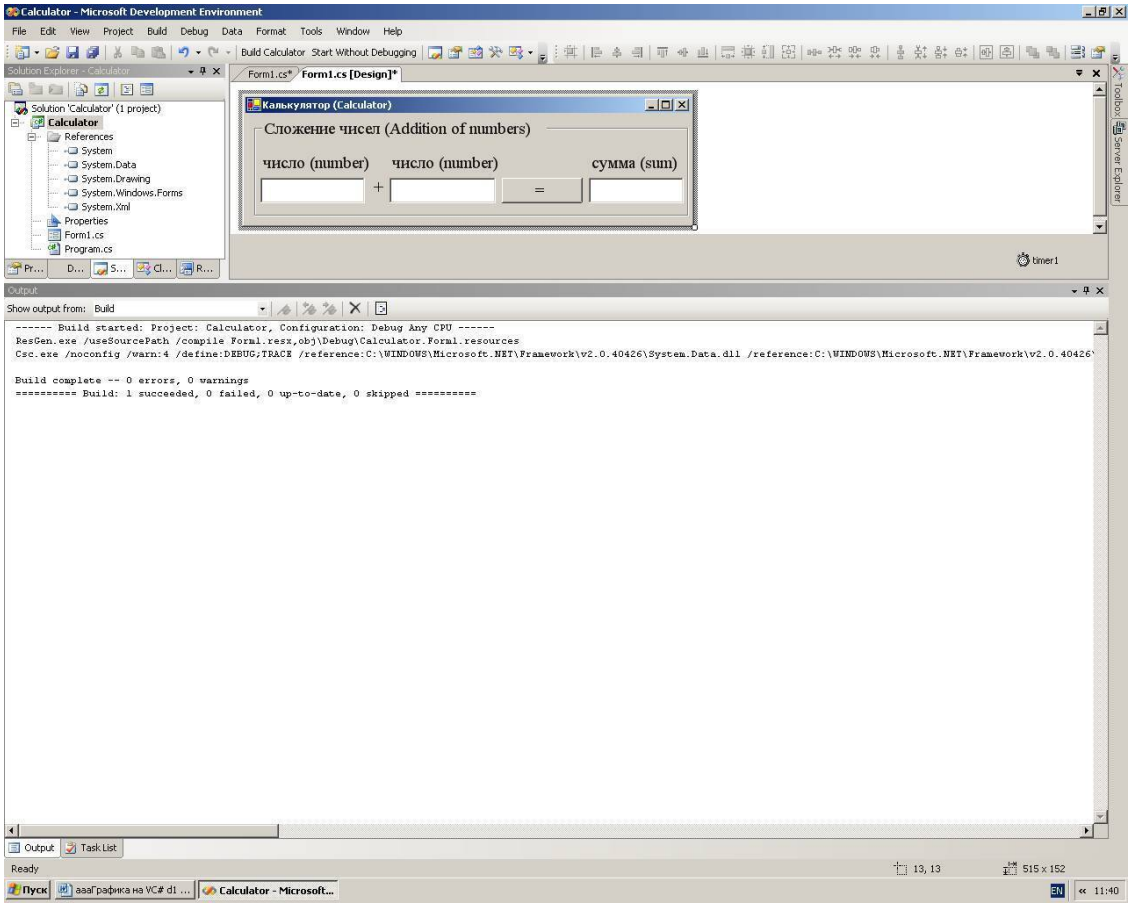


Рис. 2.6. Форма после графического (визуального) проектирования.

2.4. Код программы

Теперь в файл Form1.vb нам необходимо написать нашу часть кода для сложения двух чисел на форме. Для связывания с кодом элементов управления и компонентов используются методы, которые называются обработчиками событий и вызываются после двойного щелчка по имени соответствующего метода на вкладке Events (со значком в виде молнии) на панели Properties (рис. 2.3). Например, обработчик события в виде щелчка кнопки “=” (рис. 2.6) вызывается после двойного щелчка по имени метода Click на вкладке Events панели Properties. Но так как щелчок кнопки является наиболее распространённым событием, то он задан как событие по умолчанию и поэтому может быть также вызван двойным щелчком кнопки в режиме проектирования.

Выполняем это. Появляется файл Form1.vb с автоматически сгенерированным кодом и следующим шаблоном метода для обработки события в виде щелчка кнопки:

```
Private Sub Button1_Click(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles Button1.Click
End Sub
```

Так как в проекте по умолчанию уже подключено корневое пространство имён (Imports System), то имя System (и другие подключённые имена) далее в нашем коде мы можем не записывать. В данный шаблон можно записать много вариантов кода для выполнения простых арифметических операций. Чтобы лучше понять синтаксис визуального (с использова-

нием элементов управления, например, TextBox) программирования, приведём четыре варианта кода для сложения двух чисел. Первый вариант кода – с тремя переменными:

Листинг 2.1. Наш основной код с тремя переменными.

```
Dim a, b, c As Double
a = Convert.ToDouble(TextBox1.Text)
b = Convert.ToDouble(TextBox2.Text)
c = a + b
TextBox3.Text = c.ToString
```

Второй вариант – с двумя переменными:

```
Dim a, b As Double
a = Convert.ToDouble(TextBox1.Text)
b = Convert.ToDouble(TextBox2.Text)
TextBox3.Text = a + b.ToString
```

Третий вариант – с одной переменной:

```
Dim a As Double
a = Convert.ToDouble(TextBox1.Text)
TextBox3.Text = (a + _
Convert.ToDouble(TextBox2.Text)).ToString
```

Четвёртый вариант – без использования переменных:

```
TextBox3.Text = (Convert.ToDouble(TextBox1.Text) + _
Convert.ToDouble(TextBox2.Text)).ToString
```

По этой методике можно записывать код для выполнения различных математических операций. Подробные объяснения этой программы (какие пространства имён, классы, методы и свойства мы использовали) даны в наших книгах [Литература].

После построения программы (щёлкаем Build, Build Selection или значок с изображением трёх вертикальных стрелок) мы увидим в выходном окне Output сообщение компилятора VB или об успешном построении, или об ошибке (если при вводе нашего кода был введен не тот символ) с указанием типа ошибки и номера строки кода с ошибкой. Ошибку стандартно исправляем, снова строим программу, и так до тех пор, пока не получим сообщение компилятора без ошибок и предупреждений.

Если ошибок нет, то в меню Debug выбираем Start Without Debugging (или щёлкаем по значку с изображением восклицательного знака). На рабочем столе поверх формы в режиме проектирования (рис. 2.6) появляется форма в режиме выполнения (рис. 2.7), которую можно захватить мышью за верхнюю полоску и передвинуть в удобное место.

2.5. Выполнение расчётов

Выполняем типичный расчёт: 1,1 + 2,2 = 3,3 и результат показываем на рис. 2.7.

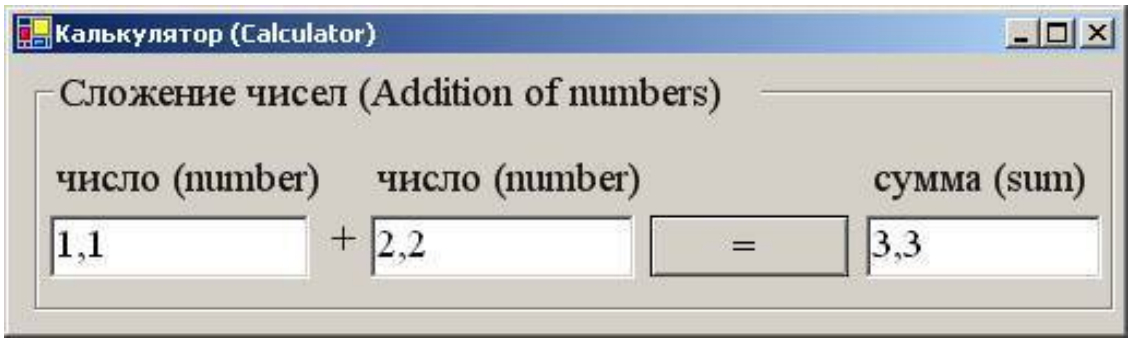


Рис. 2.7. Пример сложения двух чисел.

На рис. 2.7 видно, что и в первые два окна мы записываем десятичную дробь с запятой, и результат получаем в виде десятичной дроби с запятой. Следовательно, этот вариант кода на листинге 2.1 соответствует российским (и международным) стандартам, когда десятичная дробь записывается с запятой. Поэтому в дальнейшем мы будем применять именно этот вариант кода.

Аналогично выполняется суммирование различных чисел: целых и дробных, положительных и отрицательных. Таким образом, мы выполнили первый традиционный расчёт сложения двух чисел и теперь можем разрабатывать методики для решения более сложных задач при помощи Visual Basic (в последующих главах).

2.6. Техническая характеристика калькулятора

Исследуем возможности созданного нами калькулятора с целью применения его на практике. Попытаемся ввести в первое окно максимально большое число, состоящее, например, из двоек (цифр 2). Оказывается, в окно можно вводить большое количества цифр (сколько поместится в окне, каким бы большим мы его не делали), но учитываться в расчёте будет только ограниченное количество этих цифр. Для примера вводим двадцать двоек. Во второе окно также записываем цифры, например, двадцать троек (цифр 3). После щелчка кнопки “=” результат виден на рис. 2.8. В числе с плавающей точкой (точнее, запятой), например, 5,555555555555557E+19 (рис. 2.8) цифры перед символом E называются мантиссой, а после E – порядком. Следовательно, в нашем калькуляторе максимальное количество разрядов мантиссы, дающих правильное значение числа, – пятнадцать (последняя пятнадцатая цифра 6 на рис. 2.8 округлена и определяет погрешность вычислений). Если после каждого щелчка кнопки “=” постепенно увеличивать количество цифр в первом или во втором окне, то увидим, что тридцать вторая (и далее) цифра уже не увеличивает порядок суммарного числа в третьем окне. Следовательно, в нашем калькуляторе максимальный порядок числа равен 31.

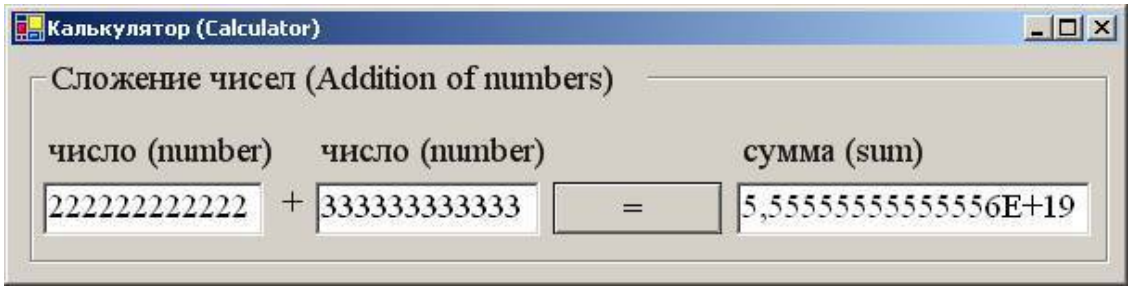


Рис. 2.8. Результат сложения двух больших чисел.

Логичным завершением исследования возможностей нашего калькулятора явится его следующая краткая техническая характеристика:

1. Система счисления вещественных чисел при вводе и выводе – десятичная.
2. Максимальное количество разрядов мантиссы числа – пятнадцать (15).
3. Максимальный порядок числа – тридцать один (31).
4. Диапазон вычислений числа “x” по модулю |x|

1*10E-031 <= |x| <= 9.99999999999999*10E+031.

5. Форма представления запятой (точки):

в диапазоне

1 <= |x| <= 999999999999999

– естественная;

в диапазонах
1*10E-031 <= |x| < 1
и 999999999999999 < |x| <= 9.99999999999999*10E+031
– плавающая.

Как видно из этой технической характеристики, созданный нами калькулятор в чем-то превосходит настольные калькуляторы и Windows-калькуляторы, а в чем-то уступает. Но главное достоинство состоит в том, что наш калькулятор является открытой вычислительной системой и, если в этом есть необходимость, аналогично (как для сложения и вычитания) по разработанной выше методике мы можем вводить в наше приложение-калькулятор выполнение других арифметических и математических операций с различным количеством методов, и постепенно создавать все более сложную нашу персональную (или корпоративную) вычислительную систему для решения именно наших конкретных задач. Начнём проектировать анимацию.

2.7. Общая методика создания анимации

Разработаем общую методику создания анимации и апробируем её на примере создания мигающего заголовка формы, точнее, создания чередующегося заголовка, когда одно название заголовка будем сменяться другим названием с заданной нами частотой (или интервалом времени). По этой методике анимационный заголовок можно встроить в любое приложение.

Для создания любой анимации необходимо ввести компонент Timer по обычной схеме: на панели инструментов Toolbox щёлкаем строку Timer (рис. 2.3); щёлкаем на форме. Ниже Form1 появляется значок с надписью Timer1 (рис. 2.9), который можно захватить мышью и перенести в другое место. Напомним, что в отличие от элементов управления компоненты располагаются вне формы и поэтому на форме в режиме выполнения не видны. В панели Properties с заголовком timer1 в свойстве Enabled вместо False выбираем True (рис. 2.10), а в свойстве Interval вместо заданных по умолчанию 100 миллисекунд задаём, например, значение 500 миллисекунд (напомним, что 1000 миллисекунд равны 1 секунде). Естественно, эти установки можно выполнить не только в панели Properties, но и в программе, например, при помощи такого кода.

Листинг 2.2. Метод для включения таймера и задания интервала времени.

```
Private Sub InitializeTimer()
'Включаем таймер Timer:
Timer1.Enabled = True
'Генерируем событие Tick через каждый интервал Interval:
Timer1.Interval = 500
End Sub
```

Теперь в режиме выполнения проекта с интервалом в эти 500 миллисекунд (или 0,5 секунды) будет генерироваться запрограммированное нами событие Tick и выполняться при помощи метода Timer1_Tick (см. ниже листинг 2.3), а именно, в данной главе будет мигать заголовок формы.

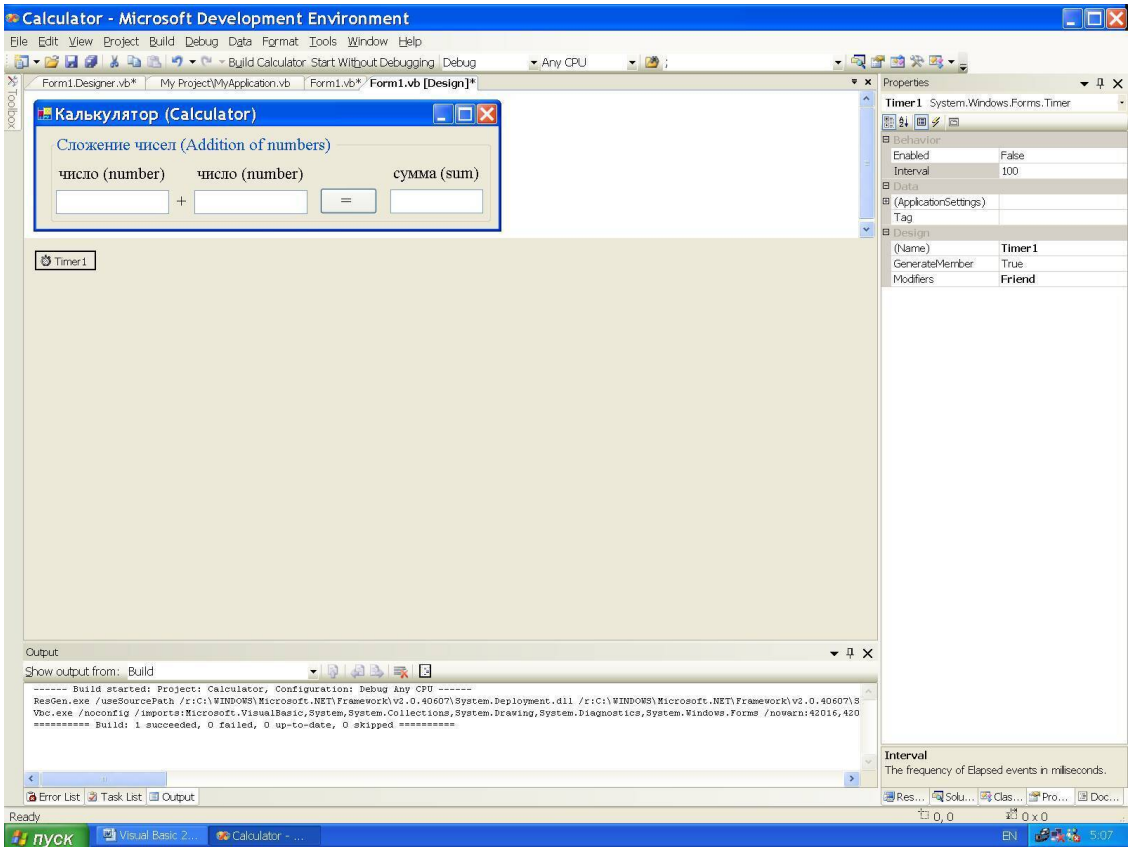


Рис. 2.9. Значок для компонента Timer.

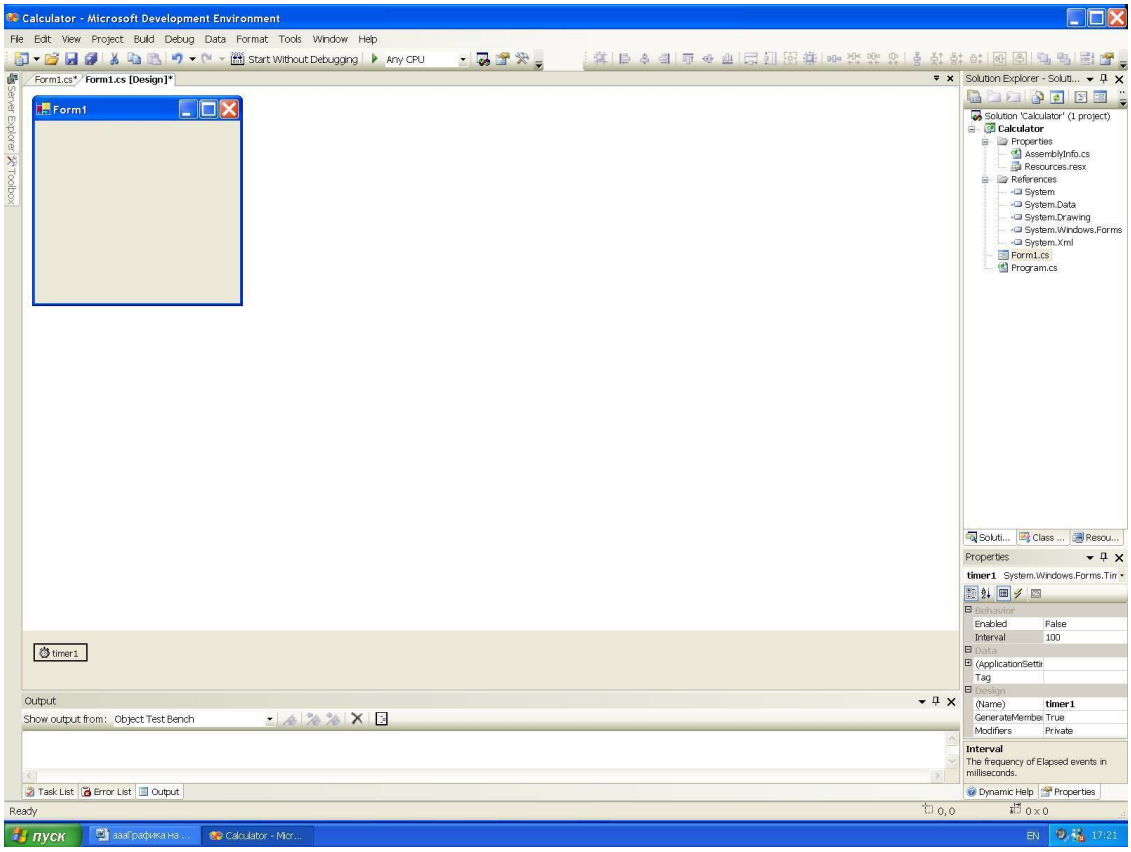


Рис. 2.10. Панель Properties.

Следовательно, мы закончили визуальную разработку анимационного эффекта, и нам необходимо написать код программы. Для этого дважды щёлкаем значок для компонента Timer (рис. 2.9). Появляется файл Form1.vb с автоматически сгенерированным шаблоном метода, который после записи нашего кода принимает вид листинга 2.3.

Листинг 2.3. Метод для создания мигающего заголовка.

```
Private Sub Timer1_Tick(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles Timer1.Tick
'Объявляем статическую переменную,
'по умолчанию равную False:
Static myText As Boolean
'Задаём чередование двух заголовков формы:
If myText Then
Me.Text = "Калькулятор (Calculator)"
Else
Me.Text = "Калькулятор с анимацией " & _
"(Calculator with animation)"
End If
'Изменяем значение myText на противоположное:
myText = Not myText
End Sub
```

Подробные пояснения к этому коду, а также другие варианты подобного кода даны в наших книгах [Литература].

Строим программу и запускаем на выполнение обычным образом: Build, Build Selection; Debug, Start Without Debugging. В ответ Visual Basic выполняет программу и выводит форму в режиме выполнения. На этой форме с заданной нами частотой в 500 миллисекунд (или 0,5 секунды) заголовок “Калькулятор (Calculator)” сменяется на “Калькулятор с анимацией (Calculator with animation)” (рис. 2.11), и таким образом создаётся эффект анимации.

Если на листинге 2.3 вместо слова "Калькулятор (Calculator)" записать оператор "" (т.е. удалить слово "Калькулятор (Calculator)"), то будет появляться и исчезать только второй заголовок формы "Калькулятор с анимацией (Calculator with animation)", и этот заголовок будет только мигать (без замены текста) с заданной частотой. Далее на этом калькуляторе можно выполнять описанные выше расчёты (рис. 2.11).

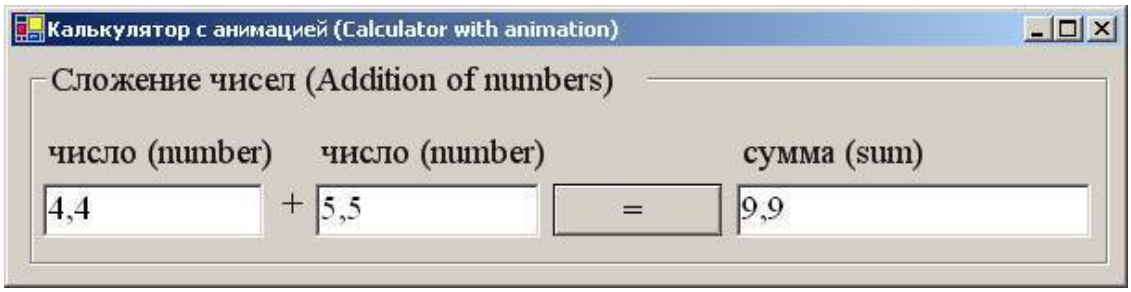


Рис. 2.11.

Аналогично создаётся анимация по другим вариантам, приведённым в наших книгах [Литература].

Следовательно, мы закончили разработку методики создания анимации на примере анимационного заголовка формы.

Подчеркнем, что мы разработали именно **общую методику создания анимации**, так как если на листинге 2.3 вместо ключевого слова Me записать значение свойства Name для какого-нибудь элемента управления (Label1, Button1 и т.д.), то мы получим эффект анимации для этого элемента управления.

Важно отметить, что на любой форме мы можем разместить несколько компонентов Timer (Таймер), каждый из которых будет создавать свой эффект анимации по разработанной в данной книге методологии.

2.8. Методика приостановки и возобновления анимации и мультипликации

В любом работающем приложении целесообразно предусмотреть возможность приостановки анимации и мультипликации (остановки изменения во времени какого-либо изображения), например, когда цель анимации достигнута, и она больше не нужна, а также предусмотреть возможность повторного запуска анимации, остановки, запуска и т.д. Можно разработать много вариантов прекращения анимации без прекращения работы всего приложения. Но все основные варианты основаны на том, что в методе для обработки какого-либо события в данном приложении вместо заданного выше значения True свойства Enabled мы должны записать значение False, например, при помощи следующей одной строки кода (которую мы уже применили в предыдущем листинге).

Листинг 2.4. Строка кода, останавливающая анимацию.

```
Timer1.Enabled = False
```

Недостаток записи только этой одной строки кода заключается в том, что после остановки анимации мы не сможем запустить её вновь. Чтобы возобновить анимацию, мы должны в обработчик события записать другую строку кода:

Листинг 2.5. Строка кода, возобновляющая анимацию.

```
Timer1.Enabled = True
```

Теперь объединим эти две последние строки кода в обработчике события с целью приостановки и возобновления анимации после каждого щелчка, например, кнопки. Для этого в режиме проектирования Form1 стандартно (как описано выше) вводим новую кнопку, в свойстве Text записываем &Stop/Start Animation и дважды щёлкаем по этой кнопке. Появляется файл Form1.vb с автоматически сгенерированным шаблоном метода, выше которого объявляем булеву переменную, а в шаблон записываем код, как показано на следующем листинге.

Листинг 2.6. Код для приостановки и возобновления анимации. Вариант 1.

```
'Объявляем булеву переменную OffOn и задаём ей значение False:
Dim OffOn As Boolean '= False по умолчанию.
Private Sub Button2_Click(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles Button2.Click
'Задаём чередование остановки и возобновления анимации
'после каждого щелчка кнопки Button:
If (OffOn = False) Then
'Останавливаем анимацию:
Timer1.Enabled = False
'Изменяем значение OffOn на противоположное:
OffOn = True
Else
'Возобновляем анимацию:
Timer1.Enabled = True
'Изменяем значение OffOn на противоположное:
```

```
OffOn = False
End If
End Sub
```

Этот листинг можно записать короче:

Листинг 2.7. Код для приостановки и возобновления анимации. Вариант 2.

```
'Объявляем булеву переменную OffOn и задаём ей значение True:
Dim OffOn As Boolean = True
Private Sub Button2_Click(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles Button2.Click
OffOn = Not OffOn
Timer1.Enabled = OffOn
End Sub
```

Для проверки реализации алгоритма запускаем программу, например, так: Ctrl+F5. В ответ Visual Basic выполняет программу и выводит форму в режиме выполнения.

На этой форме с заданной нами частотой в 500 миллисекунд (или 0,5 секунды) заголовок “Калькулятор (Calculator)” сменяется на “Калькулятор с анимацией (Calculator with animation)” (рис. 2.11), и таким образом создаётся эффект анимации.

Анимация прекращается и возобновляется поочерёдно после каждого щелчка кнопки (рис. 2.12). Так как в свойстве Text мы записали &Stop/Start Animation с символом &, то первая буква S подчёркнута, и, следовательно, эту кнопку можно нажать не только мышью, но и комбинацией клавиш Alt+s.

Аналогично можно разработать другие варианты анимации с одним или несколькими компонентами Timer (Таймер) на любой форме, а также другие варианты приостановки и возобновления анимации и мультипликации, как будет показано далее.

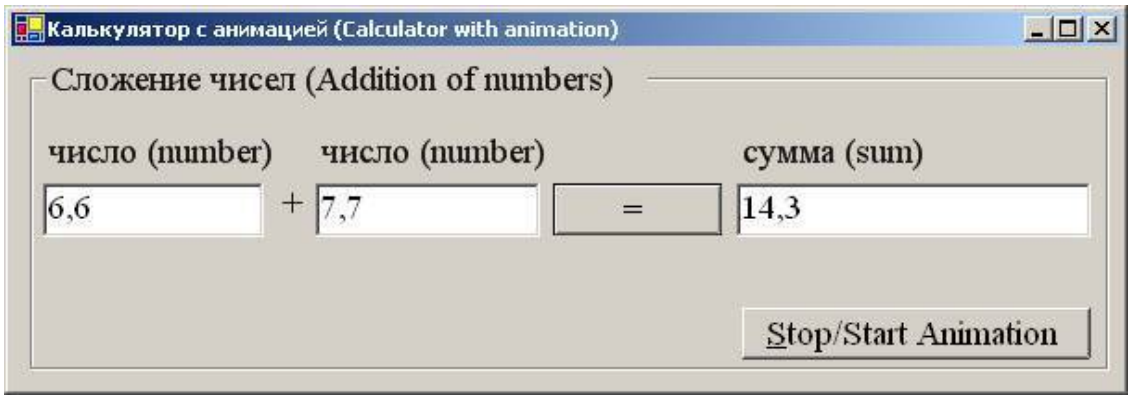


Рис. 2.12.

2.9. Методика подачи звукового сигнала

Целесообразно, чтобы в работающем приложении эффекты анимации и мультипликации сопровождались звуковыми эффектами, и самым простым из них является подача звукового сигнала. Подача звукового сигнала основана на том, что в процедуру по обработке какого-либо события в данном приложении мы записываем стандартную функцию Beep().

Если мы запишем эту функцию Beep() в процедуру для обработки события Tick таймера, то звуковой сигнал будет периодически создаваться согласно генерируемому событию Tick с заданным нами интервалом времени Interval. Если мы хотим ограничить число звуковых сиг-

налов величиной N, то выше процедуры объявляем и задаём (инициализируем, приравнивая, например, 10) эту переменную N:

```
Dim N As Integer = 10
```

а в самой процедуре организовываем цикл по статической переменной i:

```
Static i As Integer
i = i + 1
If i <= N Then
Beep()
End If
```

В дальнейшем мы разработаем программы для подачи звукового сигнала в различные моменты анимации, например, в момент каждого удара вечно прыгающего мяча о преграду (внутри которой прыгает мяч).

Также в дальнейшем кратко (а в наших предыдущих книгах [Литература] подробно) мы разработаем методику дополнения любого приложения говорящими мультипликационными персонажами, которыми можно управлять при помощи щелчков клавиш и кнопок и голосовых команд в микрофон.

В заключении этой главы отметим, что данная методология (парадигма) проектирования классического калькулятора для сложения двух чисел позволяет нам не только самостоятельно и быстро изучить (понять и осознать) некоторые основы Visual Basic с учётом анимации, но и одновременно (параллельно с освоением) создать открытое (для дополнения) приложение, которое мы уже можем применять в нашей индивидуальной практической и повседневной деятельности.

В следующей главе мы опишем более сложную методику создания приложения-калькулятора не на одной, а на двух (и более) формах с другими анимационными эффектами и разработаем методику передачи данных с одной формы на другую.

Глава 3. Методика разработки приложений на нескольких формах и передачи данных с одной формы на другую

3.1. Алгоритм приложения и проектирование первой формы

Будем усложнять методические примеры. Поэтому, если в предыдущей главе мы разработали методику ввода исходных данных в одну форму и вывода результатов проектирования на эту же форму, в этой главе рассмотрим пример (который может найти широкое применение на практике) и разработаем методику ввода исходных данных в одну форму, а вывода результатов проектирования на другую форму. Эта же методика может быть применена и при создании вычислительной системы для вывода результатов проектирования на большое число форм в соответствии с потребностями пользователя. Так как на практике (например, на производстве) важным является решение различных расчётных задач, то продолжим разработку методического примера расчёта, например, умножения двух чисел.

Алгоритм этого примера сформулируем так: на первой (главной) форме: в первое окно вводим первый сомножитель; во второе окно вводим второй сомножитель; щёлкаем по кнопке со знаком равенства; на появившейся второй форме (с пустыми тремя окнами) щёлкаем кнопку OK, после чего во всех трёх окнах мы увидим числа. Проверяем (контролируем) правильность вывода программой двух сомножителей в первые два окна второй формы (эти значения мы просто передадим с первой формы и подробно объясним, как это делается); анализируем результат умножения, который мы увидим в третьем окне.

Кратко, чтобы не повторяться (более подробно приведено выше и в [Литература]), опишем создание проекта для нового приложения-диалога. В VS щёлкаем значок New Project (или выбираем File, New, Project). В панели New Projects в окне Project Types выбираем тип проекта Visual Basic, Windows, в окне Templates проверяем, что выделено (по умолчанию) Windows Forms Application, в окне Name записываем имя проекта, например, Calculator2_2 (первая цифра 2 означает второй вариант калькулятора, а вторая цифра 2 – на двух формах). Щёлкаем OK. В ответ Visual Basic создаёт проект нашего приложения и выводит рабочий стол с формой Form1(аналогично форме в предыдущей главе). По разработанной выше методике осуществляем визуальное проектирование формы (рис. 3.1) и вводим элементы управления (рамку группы GroupBox, окна TextBox, кнопки Button, тексты Label) и компонент таймер (свойства таймера Timer: Enabled – True; значение Interval, например, оставляем по умолчанию, равным 100).

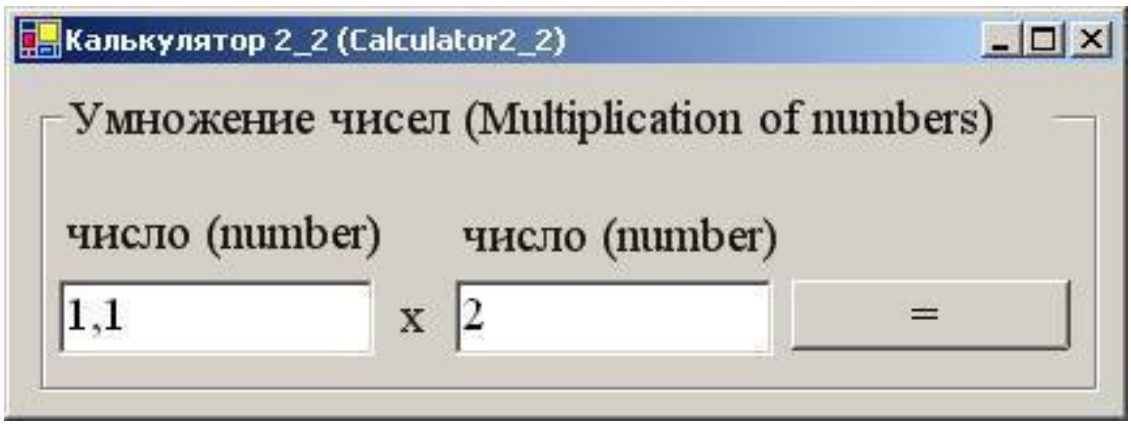


Рис. 3.1. Form1 в режиме выполнения.

3.2. Проектирование следующей формы

Для ввода в проект новой формы, в меню Project выбираем Add Windows Form, в панели Add New Item щёлкаем кнопку Add (или Open). В ответ Visual Basic выводит рабочий стол с новой формой Form2 и добавляет в панель Solution Explorer новый пункт Form2.vb. Аналогично, как первую, проектируем вторую форму (рис. 3.2). По этой схеме можно добавлять и большее количество форм, сколько необходимо для каждого конкретного приложения.

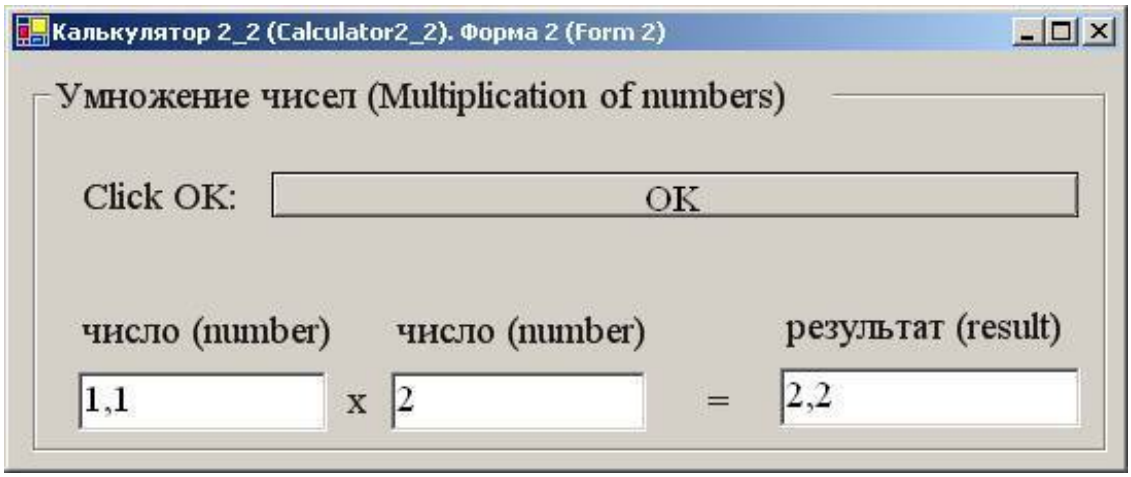


Рис. 3.2. Form2 в режиме выполнения.

3.3. Код программы

Дважды щёлкаем кнопку “=” на форме Form1 в режиме проектирования. Появившийся шаблон (после записи нашего кода) принимает вид следующей процедуры.

Листинг 3.1. Процедура Button1_Click с нашим кодом для первой формы.

```
Private Sub Button1_Click(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles Button1.Click
Dim A, B As Double
A = Convert.ToDouble(TextBox1.Text)
B = Convert.ToDouble(TextBox2.Text)
Dim myForm2 As New Form2
myForm2.C = A
myForm2.D = B
myForm2.Show()
End Sub
```

Дважды щёлкаем кнопку OK на Form2. Перед появившимся шаблоном объявляем две открытые глобальные переменные C и D, а внутри этого шаблона записываем наш код, после чего шаблон принимает вид следующей процедуры.

Листинг 3.2. Строка и процедура Button1_Click с нашим кодом для Form2.

```
Public C, D As Double
Private Sub Button1_Click(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles Button1.Click
Dim F, G As Double
```

```
F = C : G = D
TextBox1.Text = F.ToString
TextBox2.Text = G.ToString
TextBox3.Text = (F * G).ToString
End Sub
```

Таких локальных переменных, как A и B, и, соответственно, глобальных переменных C и D, в общем случае, мы записываем попарно столько, сколько на первой форме имеется окон TextBox, из которых мы будем передавать значения на другую форму. Переменные F, G можно не вводить (мы их ввели для наглядности) и заменить их на C, D. Отметим, что мы разработали несколько вариантов кода для передачи данных с одной формы на другую, но в этой книге приводим только один вариант (листинги 3.1 и 3.2), как наиболее простой. Другие варианты кода даны и подробно объяснены в наших книгах [Литература].

3.4. Методика разработки анимации в виде бегущей строки

На основании разработанной в предыдущей главе методики создания анимационного заголовка формы в данной главе мы разработаем методику создания бегущей строки любого текста, как в заголовке формы, так и внутри какого-либо элемента управления. Эту методику опишем на примерах двух вариантов подвижного заголовка, а именно: бегущий слева – направо заголовок первой формы; бегущий справа – налево заголовок следующей формы. Алгоритм бегущего слева – направо заголовка первой форме формулируем так: начиная с первой буквы, поэтапно появляются буквы заголовка (по одной букве) с заданным нами в панели Properties интервалом времени Interval; после появления всех букв заголовка он исчезает, и цикл поэтапного (побуквенного) вывода заголовка повторяется. Для программной реализации этого алгоритма дважды щёлкаем значок для компонента Timer ниже первой формы в режиме проектирования. Появляется файл Form1.vb с шаблоном метода Timer1_Tick для обработки события Tick, периодически (с заданным интервалом) возбуждаемого объектом (таймером) Timer. Этот автоматически генерируемый шаблон мы уже приводили выше. Здесь мы будем давать лишь наш код, который следует записать в этот шаблон.

Листинг 3.3. Код для бегущего слева – направо заголовка.

```
'Записываем текст заголовка:
Dim myString As String = "Калькулятор2_2 (Calculator2_2) "
'Объявляем статическую переменную, по умолчанию равную нулю:
Static i As Integer
'Справа – налево появляются буквы заголовка:
Me.Text = myString.Substring(0, i)
i = i + 1
'Организовываем цикл вывода заголовка:
If i = myString.Length Then i = 1
```

Алгоритм бегущего справа – налево заголовка следующей формы формулируем иначе (чем предыдущий): появляются все буквы заголовка; начиная с последней буквы, поэтапно исчезают буквы заголовка (по одной букве) с заданным нами в панели Properties интервалом времени Interval; после исчезновения последней буквы заголовка снова появляются все буквы заголовка и цикл поэтапного (побуквенного) удаления заголовка повторяется. Дважды щёлкаем значок для компонента Timer ниже формы в режиме проектирования. Появляется файл Form1.vb с шаблоном, в который записываем код:

Листинг 3.4. Код для бегущего справа – налево заголовка.

```
'Записываем текст заголовка:
Dim myString As String = _
```

```
"Калькулятор2_2 (Calculator2_2). Форма2 (Form2) "
'Статическая переменная, равная числу знаков заголовка:
Static i As Integer = myString.Length()
'Слева – направо удаляются буквы заголовка:
Me.Text = myString.Substring(0, i)
i = i – 1
'Организовываем цикл удаления букв заголовка:
If i = -1 Then i = myString.Length()
```

Аналогично можно запрограммировать бегущую строку внутри какого-либо элемента управления (или нескольких элементов управления), если на листингах 3.3 и 3.4 в строке (Me.Text = myString.Substring(0, i) после оператора (Me.) мы допишем имя этого элемента управления (свойство Name), например, (Button1.) для кнопки.

3.5. Выполнение расчётов

Проверяем в действии созданное нами приложение (проект) в виде программы-калькулятора, например, для вычисления произведения двух чисел:

1. Запускаем программу: Build, Build Selection; Debug, Start Without Debugging.

В ответ Visual Basic выполняет программу и выводит первую форму с пустыми окнами и мигающим курсором в первом окне. Мы видим также бегущий слева – направо заголовок формы.

2. В первое окно вводим первый сомножитель (рис. 3.1).

3. Щёлкаем во втором окне, вводим второй сомножитель и щёлкаем кнопку “=”.

Появляется вторая форма (рис. 3.2) с пустыми окнами. Мы видим также бегущий справа – налево заголовок формы.

4. На второй форме щёлкаем кнопку OK.

В ответ Visual Basic на второй форме показывает (рис. 3.2):

в первом окне – значение первого сомножителя;

во втором окне – значение второго сомножителя;

в третьем окне – результат умножения двух чисел.

После окончания расчётов щёлкаем значок “x” (Close). В ответ Visual Basic закрывает вторую форму, но оставляет открытой первую форму. Мы можем ввести другие значения в окна первой форме и аналогично получить результат умножения других чисел.

Однако после окончания расчётов мы можем и не закрывать вторую форму и далее выполнять расчёты следующим образом.

1. Щёлкаем в окнах первой формы (активизируем её), вводим два (или одно) других числа (например, результат предыдущего расчёта) и щёлкаем кнопку “=”.

Появляется второй вид второй формы с пустыми окнами.

2. Щёлкаем OK и на этой форме получаем результат умножения уже других чисел.

Аналогично можно получить любое количество экземпляров второй формы с результатами вычислений. Эти формы мы можем перемещать (чтобы они не закрывали друг друга) и анализировать.

После окончания расчётов последовательно щёлкаем значок “x” (Close) на каждой форме, и формы также последовательно (по одной) закрываются.

Таким образом, мы получили решение задач согласно разработанным выше алгоритмам с учётом анимации.

На базе этого методического примера (данной главы) мы можем вводить в наше приложение-калькулятор выполнение других арифметических и математических операций с двумя,

тремя и большим количеством чисел, и с большим количеством форм, а также применять разработанные здесь эффекты анимации.

В заключении этой главы ещё раз отметим, что по сравнению с известными настольными и калькуляторами в операционной системе Windows, разработанное нами приложение-калькулятор имеет следующие преимущества: каждое число и результат расчёта расположены в своих окнах (а не в одном окне, как в стандартном калькуляторе); количество цифр в числе можно задать большим, чем в стандартном настольном калькуляторе; наш калькулятор является открытой вычислительной системой, в которую можно ввести выполнение таких математических операций, какие в стандартном калькуляторе отсутствуют; в формы можно ввести (по методикам из данной книги в последующих главах) рисунки, поясняющий текст и другие элементы управления. Кроме того, наш калькулятор имеет эффекты анимации, которые позволяют выделить заголовки и обратить внимание пользователя на важную информацию в этих заголовках.

В других наших книгах (из списка литературы) мы разработали методологию создания персональной (собственной, личной) или корпоративной вычислительной системы с эффектами анимации для выполнения более сложных расчётов с использованием многих исходных данных, которые пользователь введёт в форму.

А в данной книге, следуя её названию, а также следуя приведённым в предыдущих главах основам, приступим к разработке двухмерных и трёхмерных игр и приложений на платформах Visual Studio для настольных компьютеров, ноутбуков и планшетов как с DirectX, так и без.

Часть II. Учебная методология программирования игр и приложений с подвижными объектами

Глава 4. Методика анимации и управления подвижными объектами

4.1. Методика добавления объекта в проект

Разработаем общую обучающую методику создания типичной и широко распространённой игры, когда в качестве летающих игровых объектов используются продукты питания, следуя статье с сайта microsoft.com: Rob Miles. Games Programming with Cheese: Part One. Так как эта статья написана по программированию игры на Visual C# для смартфона и, к тому же, при помощи устаревшей версии Visual Studio, то автор данной книги переработал статью для программирования игры на настольном компьютере и, к тому же, на Visual Basic при помощи новейшей версии Visual Studio. Общие требования к программному обеспечению для разработки этой игры приведены выше. Методично и последовательно начнём решать типичные задачи (с подробными объяснениями) по созданию данной базовой учебной игры и всех подобных игр типа аркады (arcade). Первым летающим объектом, используемом в игре, является, например, какой-либо продукт питания, например, маленький кусочек сыра (cheese). Так как на экране должно размещаться большое количество игровых объектов, то размер изображения сыра также должен быть небольшим, например, 25 x 32 пикселей. Если необходимо уменьшить объём файла любого изображения, то можно воспользоваться каким-либо графическим редактором, например, Paint, который поставляется с любой операционной системой Windows.

Игру с метающими объектами, например, типа продуктов питания мы будем разрабатывать постепенно, сначала создавая простые проекты, а затем дополняя и усложняя их.

Создаём базовый учебный проект по обычной схеме: в VS в панели New Project в окне Project types выбираем тип проекта Visual Basic, Windows, в окне Templates выделяем шаблон Windows Forms Application, в окне Name записываем (или оставляем по умолчанию) имя проекта, например, Cheese1 и щёлкаем OK. Важно отметить, что, так как, в отличие от приведённой выше статьи, имя этого проекта мы будем определять далее в коде программы, то в окне Name можно записать любое имя. Создаётся проект, появляется форма Form1 (рис. 4.1) в режиме проектирования. Проектируем (или оставляем по умолчанию) форму, как подробно описано в параграфе “Методика проектирования формы”. Например, если мы желаем изменить фон формы с серого на белый, то в панели Properties (для Form1) в свойстве BackColor устанавливаем значение Window.

Добавляем в проект (из отмеченной выше статьи или из Интернета) файл изображения сыра cheese.jpg по стандартной схеме, а именно: в меню Project выбираем Add Existing Item, в этой панели в окне “Files of type” выбираем “All Files”, в центральном окне находим и выделяем имя файла и щёлкаем кнопку Add (или дважды щёлкаем по имени файла).

В панели Solution Explorer мы увидим этот файл (рис. 4.2).

Теперь этот же файл cheese.jpg встраиваем в проект в виде ресурса по разработанной выше схеме, а именно: в панели Solution Explorer выделяем появившееся там имя файла, а в панели Properties (для данного файла) в свойстве Build Action (Действие при построении) вместо заданного по умолчанию значения Content (Содержание) или None выбираем значение

Embedded Resource (Встроенный ресурс). Для написания программы, в самом верху файла Form1.vb записываем пространство имён System.Reflection для управления классом Assembly:

```
Imports System.Reflection 'Для класса Assembly.
```

В панели Properties (для Form1) на вкладке Events дважды щёлкаем по имени события Paint. Появившийся шаблон метода Form1_Paint после записи нашего кода принимает следующий вид.

Другие варианты вывода изображения, например, на элемент управления PictureBox и после щелчка по какому-либо элементу управления уже приводились в предыдущих книгах.

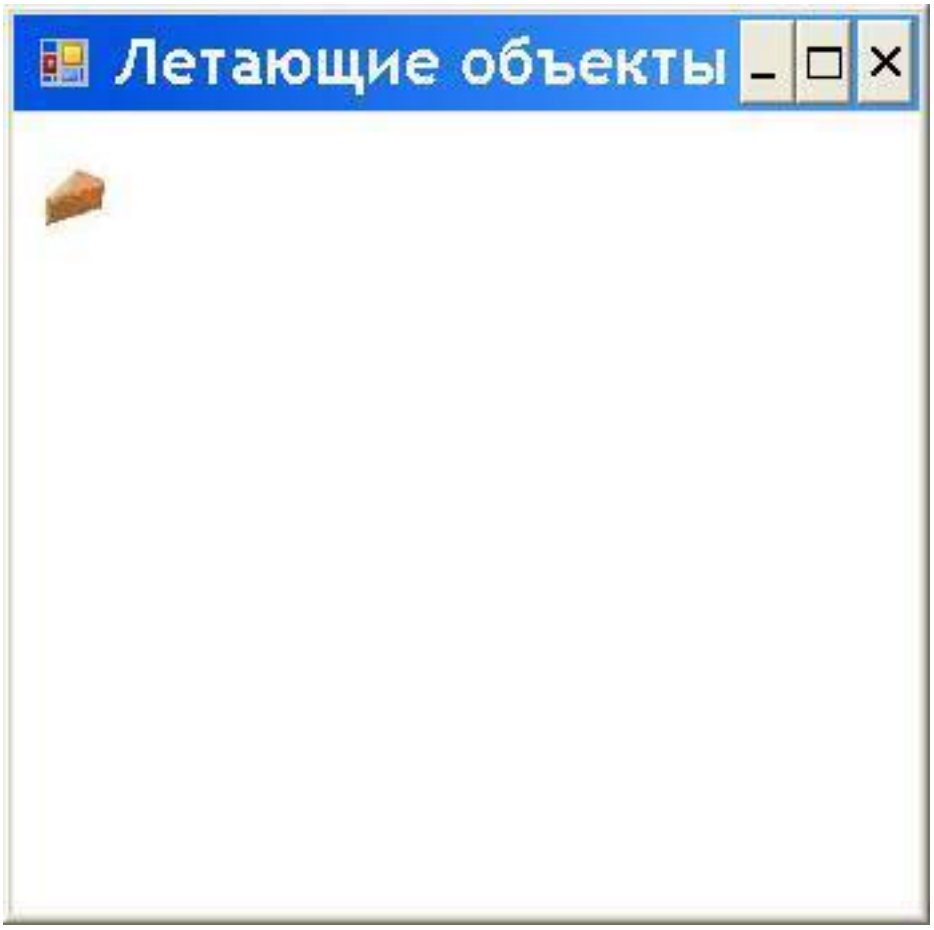
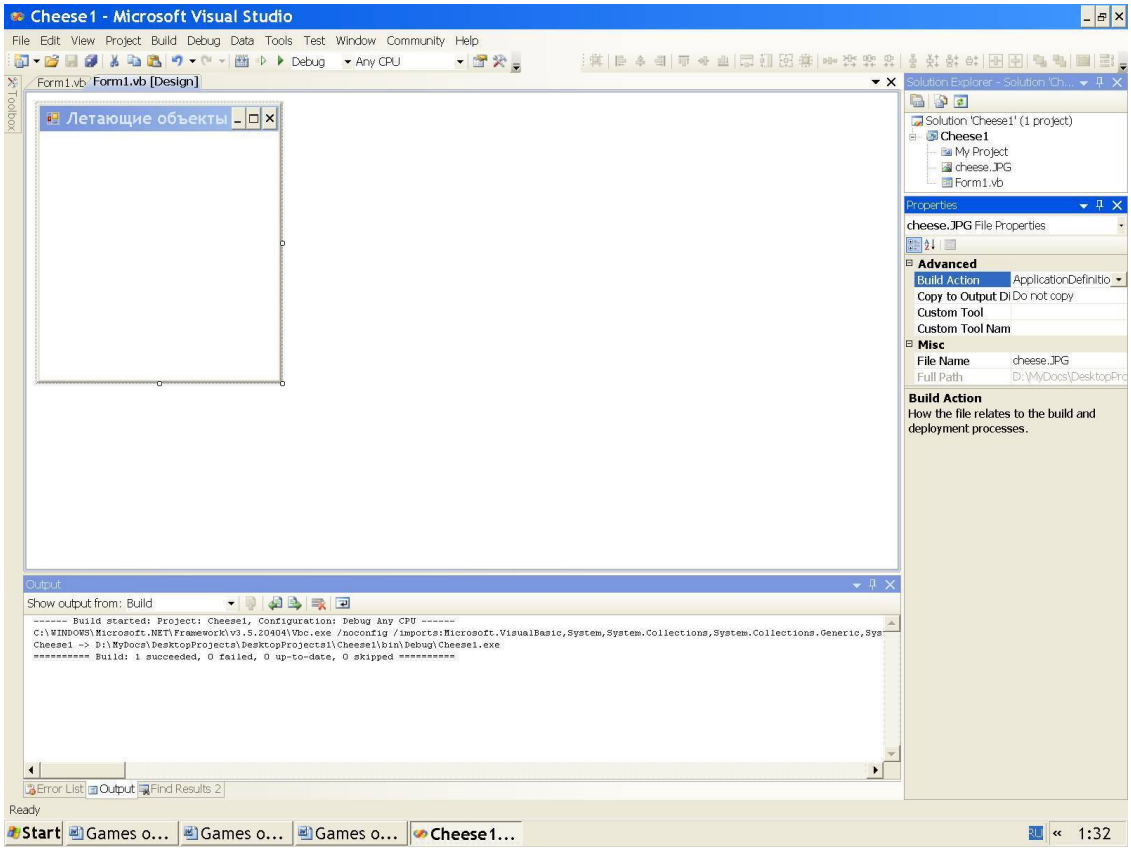


Рис. 4.1. Форма.



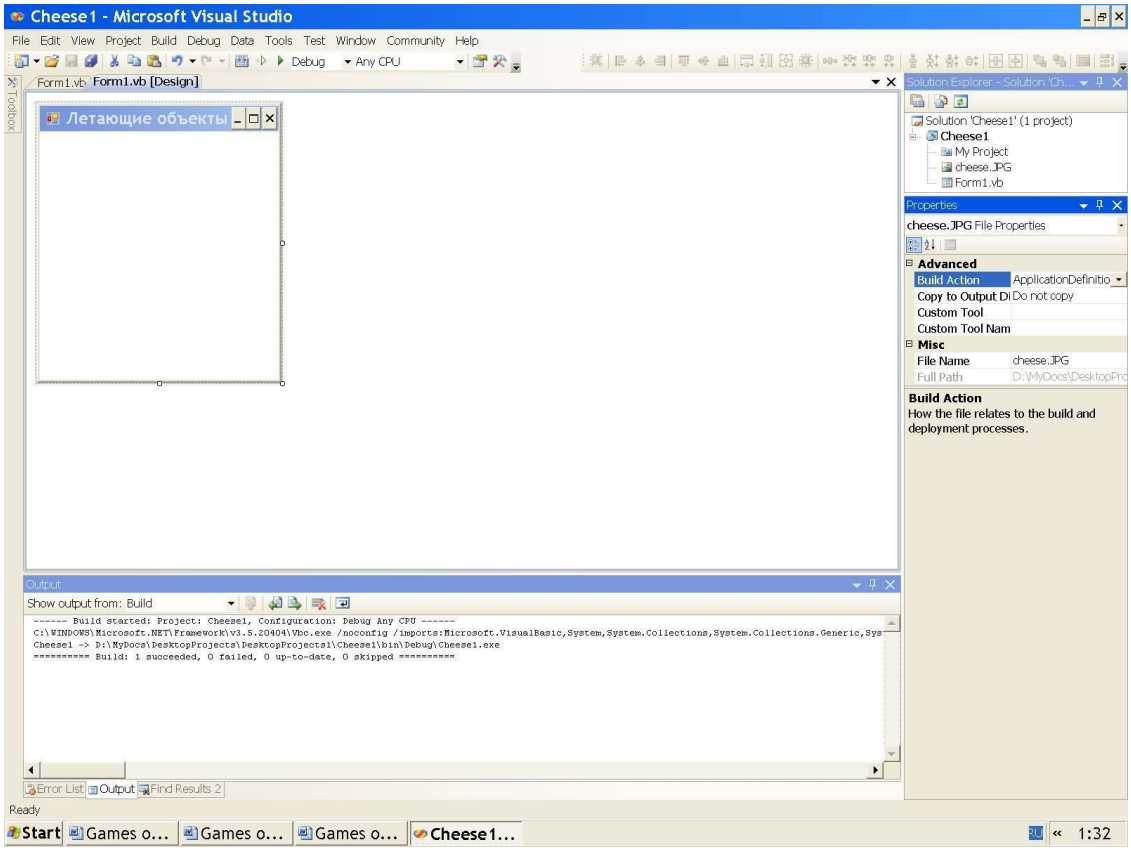


Рис. 4.2. Файл рисунка в SE и Properties.

Листинг 4.1. Метод для построения изображения.

```
'Объявляем объект класса System.Drawing.Image для продукта:
Dim cheeseImage As Image
'Загружаем в проект файлы изображений и звуков по такой схеме:
'Создаём объект myAssembly класса Assembly и присваиваем ему
'ссылку на исполняемую сборку нашего приложения:
Dim myAssembly As Assembly = Assembly.GetExecutingAssembly()
'Создаём объект myAssemblyName
'класса System.Reflection.AssemblyName и присваиваем ему
'имя сборки, которое состоит из имени проекта,
'Version, Culture, PublicKeyToken:
Dim myAssemblyName As AssemblyName = myAssembly.GetName()
'Из имени сборки при помощи свойства Name
'выделяем имя проекта типа string:
Dim myName_of_project As String = myAssemblyName.Name
Private Sub Form1_Paint(ByVal sender As System.Object, _
ByVal e As System.Windows.Forms.PaintEventArgs) _
Handles MyBase.Paint
'Загружаем в объект класса System.Drawing.Image
'добавленный в проект файл изображения заданного формата
'при помощи потока встроенного ресурса (ResourceStream):
cheeseImage = _
New Bitmap(myAssembly.GetManifestResourceStream( _
myName_of_project + "." + "cheese.JPG"))
'Рисуем изображение на форме Form1:
e.Graphics.DrawImage(cheeseImage, 10, 20)
End Sub
```

Строим и запускаем программу на выполнение обычным образом:

Build, Build Selection;

Debug, Start Without Debugging.

В ответ VS выводит панель Deploy (с именем проекта), на которой выбираем устройство (Device) типа Windows Mobile 6 Classic (или Professional) Emulator и щёлкаем кнопку Deploy.

Появляется форма Form1 с изображением типа встроенного нами рисунка сыра cheese.jpg (рис. 4.1).

Верхний левый угол изображения по отношению к верхнему левому углу экрана (где находится начало координат) расположен в соответствии с заданными нами координатами в строке кода (e.Graphics.DrawImage(myBitmap, 10, 20)).

4.2. Методика анимации объекта

Программа может рисовать теперь сыр на экране. Затем она должна перемещать сыр, неоднократно рисуя и перерисовывая изображение сыра в различных позициях. Если программа делает это достаточно быстро, создаётся иллюзия движения (анимация).

Следующий пример кода создаёт метод updatePositions, который перемещает сыр. На данной стадии проектирования сыр будет только двигаться вправо и вниз (по осям координат “x” и “y”). Таким образом, добавляем в данный (или новый) проект такой код.

Листинг 4.2. Изменение координат продукта.

```
'Текущая абсцисса объекта:
Dim cx As Integer = 50
```

```
'Текущая ордината объекта:
Dim cy As Integer = 100
Private Sub updatePositions()
cx = cx + 1
cy = cy + 1
End Sub
```

Видно, что программа использует переменные cx и cy, чтобы задавать местоположение сыра. Сейчас их значения становятся больше на единицу каждый раз, когда вызывается обновление экрана, что заставляет сыр двигаться направо и вниз.

В процессе игры, для вызова метода updatePositions через одинаковые промежутки времени, целесообразно использовать таймер. С панели инструментов Toolbox размещаем на форме компонент Timer (Таймер). В панели Properties (для данного компонента Timer) в свойстве Enabled оставляем логическое значение False, а свойству Interval задаём значение 40 (миллисекунд, что соответствует 25 кадрам в секунду по стандарту телевещания России; 1000 миллисекунд равно 1 секунде).

Важно отметить, что добавление в проект компонента Timer (Таймер) означает, что наша игра должна отключить таймер, когда игра находится в фоновом режиме, и включить таймер при активации игры. Именно поэтому в панели Properties (для данного компонента Timer) в свойстве Enabled мы оставили логическое значение False.

Кроме того, таймер не должен быть включённым, пока программа не загрузит изображение. Поэтому в приведённом выше методе Form1_Paint в самом низу следует дописать:

```
'Включаем таймер:
Timer1.Enabled = True
```

Окончательно, код в теле метода Form1_Paint должен иметь такой вид.

Листинг 4.3. Метод для рисования изображения.

```
Private Sub Form1_Paint(ByVal sender As System.Object, _
ByVal e As System.Windows.Forms.PaintEventArgs) _
Handles MyBase.Paint
'Загружаем в объект класса System.Drawing.Image
'добавленный в проект файл изображения заданного формата
'при помощи потока встроенного ресурса (ResourceStream):
cheeseImage = _
New Bitmap(myAssembly.GetManifestResourceStream( _
myName_of_project + "." + "cheese.JPG"))
'Рисуем изображение на форме Form1:
e.Graphics.DrawImage(cheeseImage, cx, cy)
'Включаем таймер:
Timer1.Enabled = True
End Sub
```

Теперь всякий раз, когда вызывается метод Form1_Paint, программа рисует сыр на экране с соответствующими координатами cx и cy.

Дважды щёлкаем по значку для компонента Timer (ниже формы в режиме проектирования). Появляется шаблон метода timer1_Tick, который после записи нашего метода updatePositions и библиотечного метода Invalidate (или Refresh) для перерисовки изображения на экране принимает следующий вид.

Листинг 4.4. Метод для смены кадров на экране и перемещения фигуры.

```
Private Sub Timer1_Tick(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles Timer1.Tick
'Вызываем метод:
```

```
updatePositions()
'Перерисовываем экран:
Invalidate()
End Sub
```

Строим и запускаем программу на выполнение обычным образом: Build, Build Selection; Debug, Start Without Debugging. В ответ VS выводит панель Deploy (с именем проекта), на которой выбираем устройство (Device) типа Windows Mobile 6 Classic (или Professional) Emulator и щёлкаем кнопку Deploy. Появляется форма в режиме выполнения, на форме Form1 которого изображение типа встроенного нами рисунка сыра cheese.jpg из верхнего левого угла перемещается по диагонали сверху вниз (в нижний правый угол) и скрывается (рис. 4.3).

Изображение объекта мерцает, что в дальнейшем будет исправлено применением двойной буферизации.

Таким образом, мы разработали методику анимации, по которой можно перемещать любые объекты на экране .

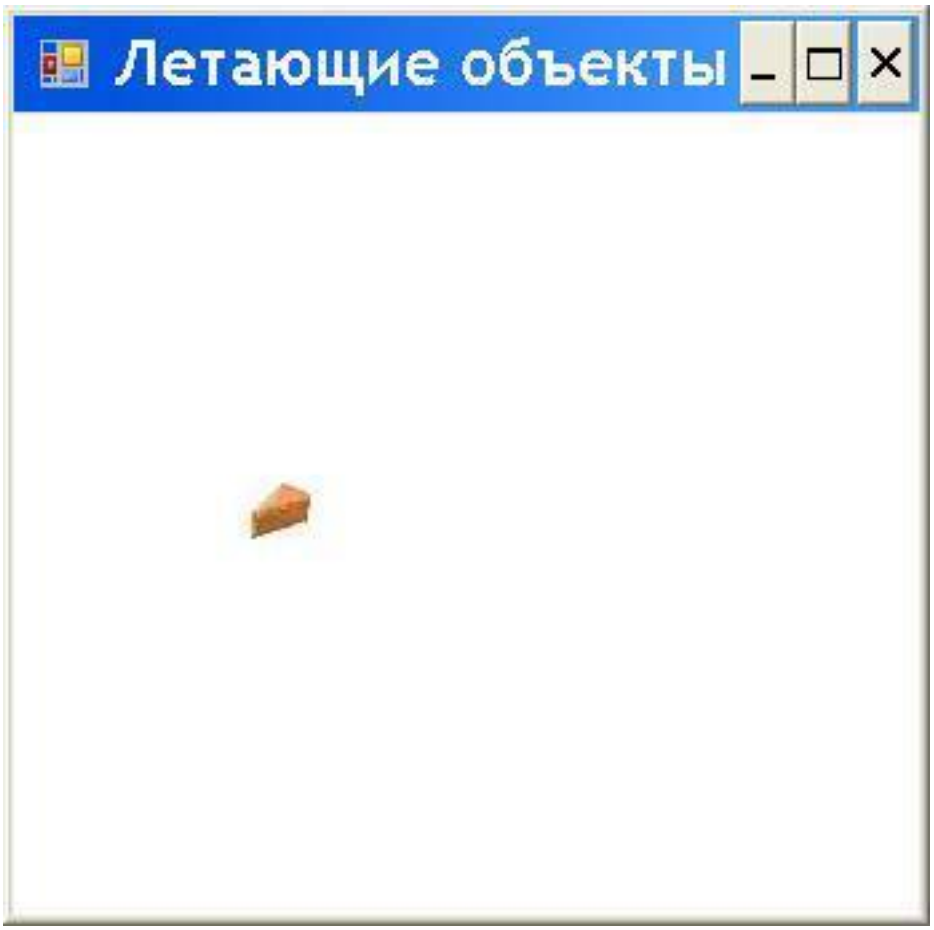


Рис. 4.3. Объект перемещается по диагонали сверху вниз.

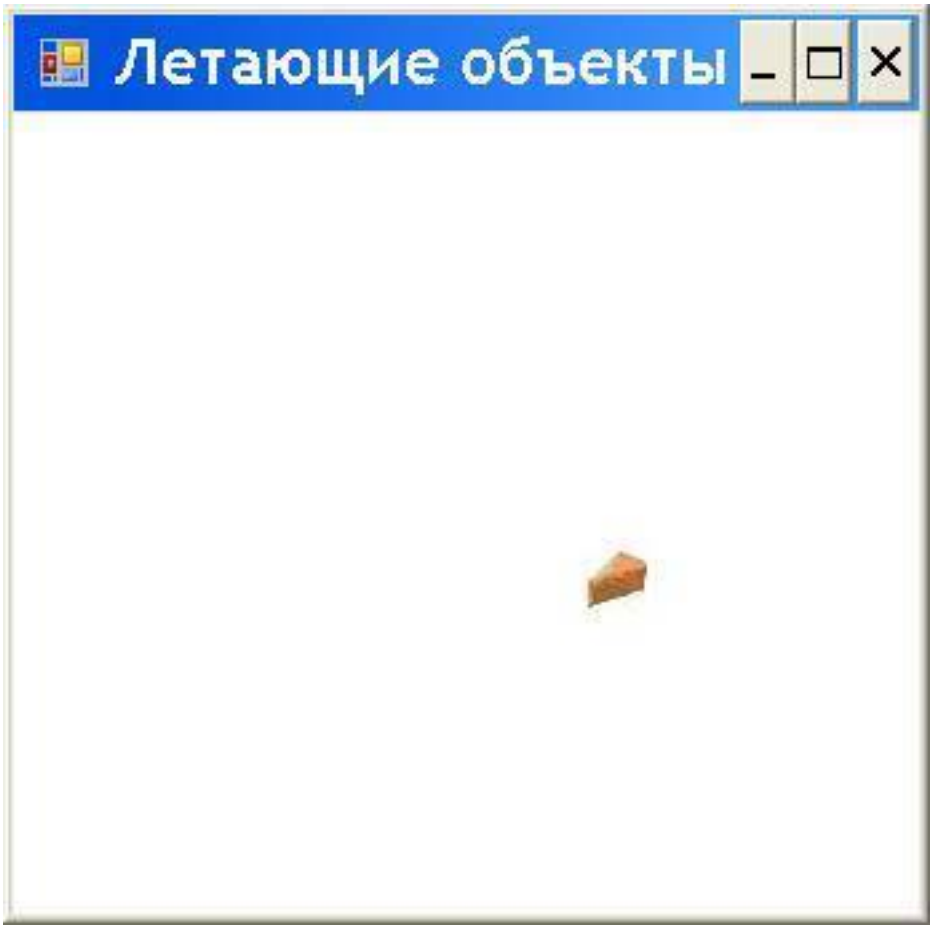


Рис. 4.4. Отскок объекта.

4.3. Методика проектирования отскока объекта от границы

Разработаем методику решения задачи по отскоку заданного нами объекта от заданных нами границ, например, от границ экрана. В качестве предмета и замкнутого пространства могут быть, например:

резиновый мяч, металлический или пластмассовый шар, который с большой силой бросил человек в каком-то помещении; предмет летает внутри помещения и отскакивает от пола, потолка и стен этого помещения;

пуля, выпущенная из огнестрельного оружия, например, стальная дробь, выпущенная из охотничьего ружья в комнате и в полете отскакивающая от пола, потолка и стен этой комнаты.

На практике подобные очень сложные задачи решаются после ввода в постановку задачи большого числа допущений.

Мы также введём большое число допущений, после чего задачу формулируем так:

решаем плоскую задачу, т.е. предмет изображаем в виде его проекции на плоскость “x, y”; в качестве примера предмета выбираем кусочек сыра cheese.jpg, проекция которого на плоскость имеет вид прямоугольника;

на этой плоскости “x, y” замкнутое пространство изображаем в виде задаваемой нами замкнутой линии; в качестве примера замкнутой линии выбираем прямоугольник границы экрана;

предмет перемещается в этой плоскости “x, y” до столкновения с границей (линией), а после удара о границу должен отскочить от границы под определённым углом и перемещаться до следующего столкновения с границей, и так далее перемещаться и отражаться от линии;

принимаем обычное допущение, что до столкновения с границей предмет перемещается (летит) по прямой линии;

на основании допущения о том, что угол падения равен углу отражения, принимаем, что после столкновения с линией прямоугольника предмет отскакивает от этой линии под тем же углом; величину угла падения и угла отражения предмета от прямой линии принимаем равной 45 градусам;

перемещение предмета осуществляется поэтапно за интервал времени, который мы установим с помощью компонента Timer (Таймер);

интервал времени устанавливаем по значению свойства Interval компонента Timer; таким образом скорость перемещения объекта можно изменять за счёт изменения свойства Interval;

анимация является бесконечным (если в него не вмешиваться) нециклическим процессом; анимацию можно остановить на любом этапе и запустить вновь.

Для решения этой задачи программа должна отслеживать текущую позицию (в виде координат) объекта, и затем, когда значение одной из двух координат объекта станет равным значению одной из двух координат границы, изменить координаты объекта в противоположном от границы направлении.

Таким образом, в данном проекте приведённый выше метод updatePositions заменяем на следующий.

Листинг 4.5. Отскок объекта от границ.

```
'Перемещение по оси "x" вправо (Right):
Dim goingRight As Boolean = True
'Перемещение по оси "y" вниз (Down):
Dim goingDown As Boolean = True
Private Sub updatePositions()
If (goingRight) Then
cx = cx + 1
Else
cx = cx – 1
End If
If ((cx + cheeseImage.Width) >= _
Me.ClientSize.Width) Then
goingRight = False
End If
If (cx <= 0) Then
goingRight = True
End If
If (goingDown) Then
cy = cy + 1
Else
cy = cy – 1
End If
If ((cy + cheeseImage.Height) >= _
Me.ClientSize.Height) Then
goingDown = False
End If
If (cy <= 0) Then
```

```
goingDown = True
End If
End Sub
```

В этом коде видно, что координаты объекта “x, y” изменяются на величину +1, когда объект перемещается в положительном направлении осей “x, y” (вправо и вниз), и изменяются на величину -1, когда объект перемещается в отрицательном направлении осей “x, y” (влево и вверх).

Код использует свойства ширины и высоты объекта (cheeseImage.Width и cheeseImage.Height) и экрана Me.ClientSize.Width и Me.ClientSize.Height). Вследствие этого программа будет нормально работать для любых размеров объекта и формы или экрана.

В режиме выполнения (Build, Build Selection; Debug, Start Without Debugging) мы видим, что на форме Form1 изображение типа встроенного нами рисунка сыра cheese.jpg перемещается по диагоналям в различных направлениях, отскакивая от границ экрана (рис. 4.4).

Методика приостановки и возобновления анимации уже была приведена выше.

4.4. Методика управления скоростью перемещения объекта и добавления звукового сигнала

Предыдущая программа довольно медленно перемещает объект по экрану. Если ширина экрана, например, 100 пикселей, то с частотой 25 кадров в секунду объект пересекает этот экран по горизонтальной прямой приблизительно за 4 секунды. Для управления скоростью перемещения объекта вместо предыдущего кода, в котором изображение перемещается на 1 пиксель через каждый Interval времени срабатывания таймера, можно изменить количество пикселей xSpeed, на которое объект перемещается через каждый Interval времени срабатывания таймера, как показано в следующем коде:

```
If (goingRight) Then
cx += xSpeed
Else
cx -= xSpeed
End If
```

Изменяя значение xSpeed, можно увеличить или уменьшить горизонтальную составляющую (по оси “x”) скорости объекта.

Следующий аналогичный код для координаты “y” позволяет изменять вертикальную составляющую скорости объекта:

```
If (goingDown) Then
cy += ySpeed
Else
cy -= ySpeed
End If
```

Увеличивать или уменьшать скорость перемещения объекта можно при помощи переменной change в следующем методе:

```
Private Sub changeSpeed(ByVal change As Integer)
xSpeed += change
ySpeed += change
End Sub
```

В этом коде целочисленная переменная change задана в виде параметра метода changeSpeed. Положительное значение переменной change увеличивает перемещение изображения через каждый Interval времени срабатывания таймера и, тем самым, увеличивает скорость, отрицательное – уменьшает.

Если мы хотим подавать звуковой сигнал в различные моменты анимации, например, в момент каждого удара объекта о границу (внутри которой перемещается объект), то поступаем следующим образом. Согласно разработанной в наших предыдущих книгах методике использования в приложении типа Visual C# метода (функции) из любого другого языка, на первом этапе необходимо создать ссылку на тот язык, например, на Visual Basic. Для этого в меню Project выбираем команду Add Reference, в панели Add Reference на вкладке (.NET) выбираем ссылку Microsoft.VisualBasic и щёлкаем кнопку OK. А в соответствующий метод, например, updatePositions записываем строку:

Microsoft.VisualBasic.Interaction.Beep();

в тех местах, где нам нужен этот сигнал.

Так как в данной книге мы разрабатываем приложения типа Visual Basic, то нам не нужно добавлять ссылку Microsoft.VisualBasic, а нужно просто записать строку (вызвать метод):

Beep()

в тех местах, где нам нужен этот сигнал.

Таким образом, в данном проекте приведённый выше метод updatePositions заменяем на следующий.

Листинг 4.6. Отскок объекта от границ.

```
'Текущее приращение перемещения по оси "x":
Dim xSpeed As Integer = 1
'Текущее приращение перемещения по оси "y":
Dim ySpeed As Integer = 1
'Метод для увеличения скорости перемещения:
Private Sub changeSpeed(ByVal change As Integer)
xSpeed += change
ySpeed += change
End Sub
'Метод для изменения координат объекта:
Private Sub updatePositions()
If (goingRight) Then
cx += xSpeed
Else
cx -= xSpeed
End If
If ((cx + cheeseImage.Width) >= _
Me.ClientSize.Width) Then
goingRight = False
'В момент столкновения подаем звуковой сигнал Beep:
Beep()
End If
If (cx <= 0) Then
goingRight = True
'В момент столкновения подаем звуковой сигнал Beep:
Beep()
End If
If (goingDown) Then
cy += ySpeed
Else
cy -= ySpeed
End If
```

```
If ((cy + cheeseImage.Height) >= _
Me.ClientSize.Height) Then
goingDown = False
'В момент столкновения подаем звуковой сигнал Beep:
Beep()
End If
If (cy <= 0) Then
goingDown = True
'В момент столкновения подаем звуковой сигнал Beep:
Beep()
End If
End Sub
```

Для управления скоростью перемещения объекта воспользуемся каким-либо элементом управления или компонентом, например, наиболее распространённым элементом Button (Кнопка). С панели инструментов Toolbox размещаем на форме две кнопки Button и в панели Properties в свойстве Text для левой кнопки записываем “Быстрее”, а для правой кнопки – “Медленнее”. Отметим, что для этих целей вместо кнопок Button (чтобы не загромождать форму) можно использовать и клавиши клавиатуры по описанной далее методике.

В режиме редактирования дважды щёлкаем по левой кнопке “Быстрее”.

Появившийся шаблон метода после записи одной строки (changeSpeed(1)) принимает следующий вид.

Листинг 4.7. Метод для изменения скорости объекта.

```
Private Sub Button1_Click(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles Button1.Click
changeSpeed(1)
End Sub
```

Аналогично дважды щёлкаем по правой кнопке “Медленнее”. Появившийся шаблон метода после записи одной строки (changeSpeed(-1)) принимает следующий вид.

Листинг 4.8. Метод для изменения скорости объекта.

```
Private Sub Button2_Click(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles Button2.Click
changeSpeed(-1)
End Sub
```

В режиме выполнения (Build, Build Selection; Debug, Start Without Debugging) мы видим, что на форме Form1 изображение типа встроенного нами рисунка сыра cheese.jpg перемещается в различных направлениях (рис. 4.5 и 4.6), отскакивая от границ экрана, а после выбора кнопок “Быстрее” или “Медленнее” этот объект перемещается соответственно быстрее или медленнее.

Причём, при каждом соприкосновении объекта с границей экрана мы слышим звуковой сигнал Beep.

Рис. 4.5. Перемещение объекта.

Рис. 4.6. Перемещение объекта.

4.5. Методика добавления нового объекта в игру

Теперь, когда программа может отображать кусочек сыра cheese.jpg в динамике, добавляем второй объект игры, который, как ракетка в теннисе отбивает мяч, будет отбивать этот кусочек сыра. В качестве такого большего по размерам объекта выбираем батон белого хлеба (с которым обычно едят сыр).

Добавляем в проект (из отмеченной выше статьи или из Интернета) файл изображения батона хлеба bread.jpg по стандартной схеме, а именно: в меню Project выбираем Add Existing Item, в этой панели в окне “Files of type” выбираем “All Files”, в центральном окне находим и выделяем имя файла и щёлкаем кнопку Add (или дважды щёлкаем по имени файла). В панели Solution Explorer мы увидим этот файл.

Теперь этот же файл bread.jpg встраиваем в проект в виде ресурса по разработанной выше схеме, а именно: в панели Solution Explorer выделяем появившееся там имя файла, а в панели Properties (для данного файла) в свойстве Build Action (Действие при построении) вместо заданного по умолчанию значения Content (Содержание) или None выбираем значение Embedded Resource (Встроенный ресурс).

Объявляем и инициализируем объект breadImage (класса Image) для загрузки в него изображения хлеба и две текущие координаты bx и by верхнего левого угла прямоугольника, описанного вокруг хлеба, в системе координат с началом в верхнем левом углу экрана. А приведённый выше код в теле метода Form1_Paint заменяем на тот, который дан на следующем листинге.

Листинг 4.9. Метод для рисования изображения.

```
'Объявляем объект класса System.Drawing.Image для предмета:
Dim breadImage As Image ' = Nothing по умолчанию.
'Текущая абсцисса предмета:
Dim bx As Integer = 0
'Текущая ордината предмета:
Dim by As Integer = 0
Private Sub Form1_Paint(ByVal sender As System.Object, _
ByVal e As System.Windows.Forms.PaintEventArgs) _
Handles MyBase.Paint
'Загружаем в объект класса System.Drawing.Image
'добавленный в проект файл изображения заданного формата
'при помощи потока встроенного ресурса (ResourceStream):
cheeseImage = _
New Bitmap(myAssembly.GetManifestResourceStream( _
myName_of_project + "." + "cheese.JPG"))
breadImage = _
New Bitmap(myAssembly.GetManifestResourceStream( _
myName_of_project + "." + "bread.JPG"))
'Рисуем изображение на форме Form1:
e.Graphics.DrawImage(cheeseImage, cx, cy)
e.Graphics.DrawImage(breadImage, bx, by)
'Включаем таймер:
Timer1.Enabled = True
End Sub
```

В режиме выполнения (Build, Build Selection; Debug, Start Without Debugging) мы видим, что на форме Form1 к перемещающемуся изображению сыра cheese.jpg добавилось изображение хлеба bread.jpg (в верхнем левом углу экрана), рис. 4.7. Однако изображения и сыра, и хлеба мерцают, что необходимо исправить методом двойной буферизации (в следующем параграфе).

4.6. Методика устранения мерцания изображения при помощи двойной буферизации

Идея устранения мерцания изображения методом двойной буферизации заключается в том, что сначала изображение проектируют не на экране, как до применения двойной буферизации, а в специальном буфере в памяти компьютера, а когда изображение полностью спроектировано в буфере памяти, оно копируется на экран . Так как процесс копирования готового изображения из буфера на экран происходит быстрее, чем процесс прорисовки изображения сразу на экране без использования промежуточного буфера, то мерцание изображения исчезает.

Чтобы устранить мерцание изображения при помощи двойной буферизации, приведённый выше код в теле метода Form1_Paint заменяем на тот, который дан на следующем листинге (с подробными комментариями).

Рис. 4.7. Подвижный сыр и неподвижный хлеб.

Рис. 4.8. Сыр закрывает хлеб.

Листинг 4.10. Метод для рисования изображения.

```
'Буфер в виде объекта класса Bitmap:
Dim backBuffer As Bitmap = Nothing
Private Sub Form1_Paint(ByVal sender As System.Object, _
ByVal e As System.Windows.Forms.PaintEventArgs) _
Handles MyBase.Paint
'Загружаем в объект класса System.Drawing.Image
```

```
'добавленный в проект файл изображения заданного формата
'при помощи потока встроенного ресурса (ResourceStream):
cheeseImage = _
New Bitmap(myAssembly.GetManifestResourceStream( _
myName_of_project + "." + "cheese.JPG"))
breadImage = _
New Bitmap(myAssembly.GetManifestResourceStream( _
myName_of_project + "." + "bread.JPG"))
'Если необходимо, создаём новый буфер:
If (backBuffer Is Nothing) Then
backBuffer = New Bitmap(Me.ClientSize.Width, _
Me.ClientSize.Height)
End If
Using g As Graphics = Graphics.FromImage(backBuffer)
'Очищаем форму:
g.Clear(Color.White)
'Рисуем изображение в буфере backBuffer:
g.DrawImage(breadImage, bx, by)
g.DrawImage(cheeseImage, cx, cy)
End Using
'Рисуем изображение на форме Form1:
e.Graphics.DrawImage(backBuffer, 0, 0)
'Включаем таймер:
Timer1.Enabled = True
End Sub
```

Если мы сейчас запустим программу на выполнение, то увидим, что мерцание уменьшилось, но не исчезло совсем. Это объясняется тем, что при выполнении метода Form1_Paint операционная система Windows сначала заполняет экран цветом фона (background color), в нашем примере белым фоном (white), и только после этого поверх фона прорисовывает встроенные в программу изображения. Поэтому необходимо сделать так, чтобы операционная система Windows не изменяла фон. Для этого воспользуемся неоднократно применяемым и в наших предыдущих книгах, и в данной книге шаблоном метода OnPaintBackground, в тело которого мы ничего не будем записывать, как показано на следующем листинге.

Листинг 4.11. Метод OnPaintBackground.

```
Protected Overrides Sub OnPaintBackground( _
ByVal e As System.Windows.Forms.PaintEventArgs)
'Запрещаем перерисовывать фон (Background).
End Sub 'Конец метода OnPaintBackground.
```

Этот метод OnPaintBackground следует записать непосредственно за методом Form1_Paint, естественно, в теле класса Form1.

Теперь в режиме выполнения (Build, Build Selection; Debug, Start Without Debugging) подвижный сыр и неподвижный хлеб уже не мерцают, и мы решили данную задачу.

Однако при перемещении сыр может перекрыть батон хлеба (рис. 4.8), хотя по правилам игры пользователь должен управлять перемещением хлеба, не давая сыру упасть вниз, а маленький кусочек сыра при столкновении должен отскочить от большого батона хлеба в противоположном направлении. Поэтому методично и последовательно перейдём к решению этих задач.

4.7. Методика управления направлением перемещения объекта при помощи элементов управления и мыши

Теперь программа должна перемещать батон хлеба таким образом, чтобы игрок мог отбивать хлебом сыр, как ракетка отбивает мяч в теннисе. Для перемещения объекта вверх (Up), вниз (Down), влево (Left) и вправо (Right) пользователь может использовать разнообразные элементы управления и компоненты с панели инструментов Toolbox, мышь, клавиатуру, джойстик и другие устройства. Для примера, размещаем на форме четыре кнопки Button с соответствующими заголовками в свойстве Text для перемещения хлеба Вверх, Вниз, Влево и Вправо (рис. 4.9). Перед размещением кнопок, для формы Form1 в панели Properties увеличиваем её размеры Size, например, до 384; 473.

По второму варианту, свяжем верхний левый угол прямоугольника, описанного вокруг хлеба, с указателем мыши, чтобы в режиме выполнения хлеб следовал за управляемым нами указателем мыши. В режиме проектирования дважды щёлкаем по каждой новой кнопке, а в панели Properties на вкладке Events дважды щёлкаем по имени события MouseMove. Появившиеся шаблоны методов для обработки этих событий после записи нашего кода принимают такой вид.

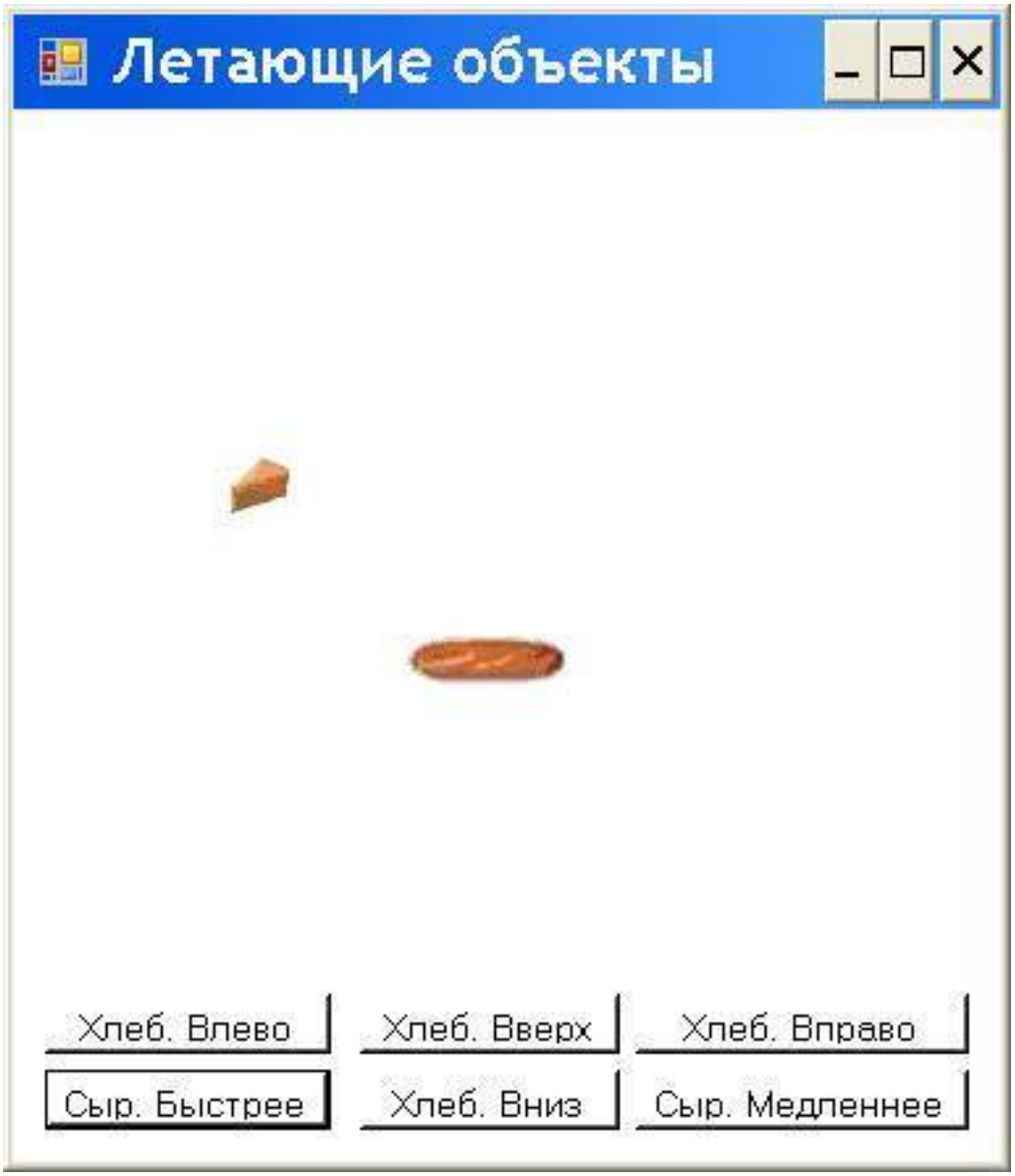


Рис. 4.9. Подвижный сыр и управляемый нами хлеб.

Рис. 4.10. Сыр закрывает хлеб.

Листинг 4.12. Методы для обработки событий.

```
Private Sub Button3_Click(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles Button3.Click
'Перемещаем объект вверх:
by -= ySpeed
End Sub
Private Sub Button4_Click(ByVal sender As System.Object, _
```

```
ByVal e As System.EventArgs) Handles Button4.Click
'Перемещаем объект вниз:
by += ySpeed
End Sub
Private Sub Button5_Click(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles Button5.Click
'Перемещаем объект влево:
bx -= xSpeed
End Sub
Private Sub Button6_Click(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles Button6.Click
'Перемещаем объект вправо:
bx += xSpeed
End Sub

Private Sub Form1_MouseMove(ByVal sender As System.Object, _
ByVal e As System.Windows.Forms.MouseEventArgs) _
Handles MyBase.MouseMove
'Определяем координаты указателя мыши на форме:
Dim mouseX As Integer = e.X
Dim mouseY As Integer = e.Y
'Задаём координаты хлеба, равные координатам мыши:
bx = mouseX
by = mouseY
End Sub
```

Для удобства, задаём другие значения начальным координатам хлеба (например, Dim bx As Integer = 150 : Dim by As Integer = 200) в нижней части формы.

После запуска программы (Build, Build Selection; Debug, Start Without Debugging) сыр самостоятельно (без нашего участия) перемещается по экрану, отталкиваясь от границы со звуковым сигналом.

А после нажатий мышью четырёх кнопок Вверх, Вниз, Влево и Вправо мы можем перемещать батон хлеба в соответствующих четырёх направлениях по всему экрану (рис. 4.9).

На рис. 4.9 видно, что по умолчанию на форме выделена первая кнопка Button (на этой кнопке размещён фокус (Focus) программы), и поэтому данную кнопку мы можем нажать не только мышью, но и клавишей Enter, после чего скорость перемещения сыра возрастёт пропорционально количеству нажатий. Если мы желаем, чтобы по умолчанию была выделена другая, например, вторая кнопка Button, то в каком-либо методе, например, в методе Form1_Paint следует записать известный код (button2.Focus();). В режиме выполнения мы можем перемещать фокус, например, клавишами со стрелками или Tab на любую кнопку на форме, после чего воздействовать на эту кнопку клавишей Enter.

По второму варианту, мы связали верхний левый угол прямоугольника, описанного вокруг хлеба, с указателем мыши, и теперь в режиме выполнения хлеб следует за управляемым нами указателем мыши.

Следовательно, мы решили задачу по управлению объектом (в виде батона хлеба) при помощи элементов управления, например, кнопок Button и мыши.

Аналогично, как для события MouseMove, для управления игрой можно использовать методы-обработчики других событий мыши, которые имеются в панели Properties на вкладке Events для формы Form1 (рис. 4.11).

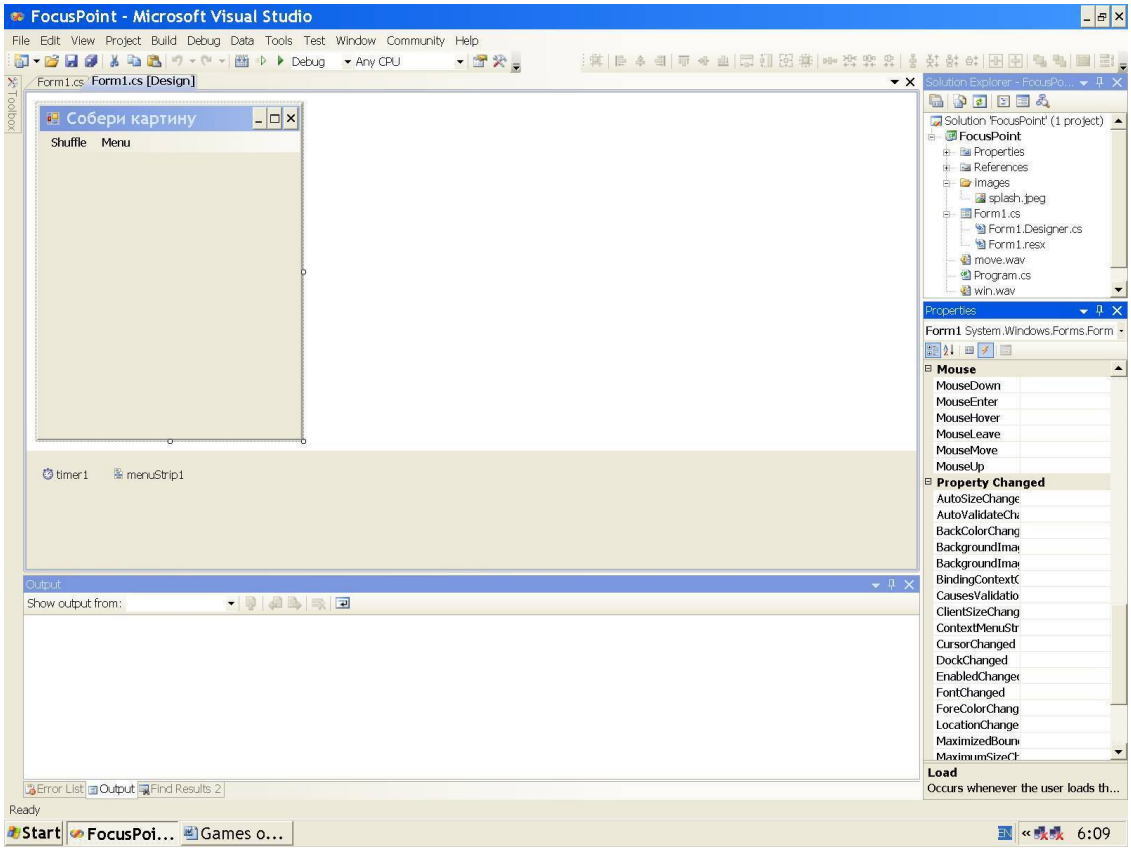


Рис. 4.11. События для управления игрой.

Отметим, что для управления игрой вместо кнопок Button (чтобы не загромождать форму) и мыши можно использовать и клавиши клавиатуры по описанной далее методике.

Однако при перемещении сыр может перекрыть батон хлеба (рис. 4.10), хотя по правилам игры, напомним, пользователь должен управлять перемещением хлеба, не давая сыру упасть вниз, а маленький кусочек сыра при столкновении должен отскочить от большого батона хлеба в противоположном направлении. Поэтому перейдём к решению задачи о столкновении летающих объектов (в следующей главе).

Таким образом, в этой главе мы разработали такие общие методики:

добавления объекта в проект;

анимации объекта;

проектирования отскока объекта от заданной нами границы, например, экрана;

управления скоростью перемещения объекта;

добавления звукового сигнала в ключевые для игры моменты, например, в момент столкновения объекта с границей;

добавления нового объекта в игру (с использованием общего для всех объектов кода);

устранения мерцания изображения при помощи двойной буферизации;

управления направлением перемещения объекта с помощью клавиш.

Эти методики можно использовать при разработке самых разнообразных игр.

Глава 5. Методика обнаружения столкновений, программирования уничтожений летающих объектов и подсчёта очков

5.1. Определение прямоугольников, описанных вокруг объектов

Продолжаем разработку методики создания типичной и широко распространённой игры, когда в качестве летающих игровых объектов используются продукты питания, следуя следующей статье с сайта microsoft.com: Rob Miles. Games Programming with Cheese: Part Two. Так как в данной статье программы написаны на языке Visual C# и, кроме того, для устаревшей версии Visual Studio, то автору данной книги пришлось переписать все программы на язык Visual Basic и, кроме того, для современной версии Visual Studio. Общие требования к программному обеспечению для разработки этой игры приведены выше.

Также продолжаем методично и последовательно решать типичные задачи по созданию данной и всех подобных игр.

Программы игр могут обнаружить столкновения между объектами при помощи прямоугольников, описанных вокруг заданных объектов. Естественно, это является существенным допущением, т.к. подавляющее большинство объектов имеют форму, отличную от прямоугольника. Однако данное допущение применяется во многих играх, и пользователь в азарте игры не замечает этой погрешности.

Прямоугольник, описанный вокруг изображения батона хлеба bread.jpg, показан на рис. 5.1.

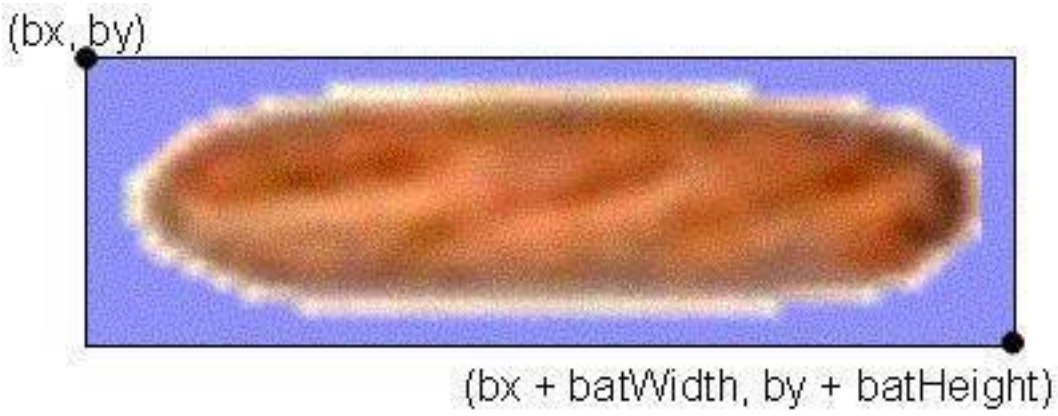


Рис. 5.1. Прямоугольник, описанный вокруг хлеба.

Ширина полей между объектом и описанным вокруг объекта прямоугольником должна быть сведена к минимуму, чтобы объект обязательно касался прямоугольника в как можно большем количестве точек и отрезков линий. Если начало прямоугольной системы координат “x, y” находится в верхнем левом углу экрана , то координаты верхней левой точки (bx, by) и нижней правой точки (bx + batWidth, by + batHeight) однозначно определяют данный прямоугольник на экране.

В среде выполнения .NET Framework (для настольных компьютеров) известна структура Rectangle (из пространства имён System.Drawing), у которой метод-конструктор Rectangle Constructor имеет несколько перегрузок. Наиболее применяемая перегрузка метода-конструктора Rectangle Constructor (которую далее и мы будем часто применять) с параметрами (Int32,

Int32, Int32, Int32) структуры Rectangle на главных (в мире программирования) языках приведена в табл. 5.1.

Таблица 5.1.

Метод-конструктор Rectangle Constructor (Int32, Int32, Int32, Int32) структуры Rectangle.

Visual Basic (Declaration)

```
Public Sub New ( _
x As Integer, _
y As Integer, _
width As Integer, _
height As Integer _
)
```

Visual Basic (Usage)

```
Dim x As Integer
Dim y As Integer
Dim width As Integer
Dim height As Integer
Dim instance As New Rectangle(x, y, width, height)
```

C#

```
public Rectangle (
int x,
int y,
int width,
int height
)
```

C++

```
public:
Rectangle (
int x,
int y,
int width,
int height
)
```

J#

```
public Rectangle (
int x,
int y,
int width,
int height
)
```

JScript

```
public function Rectangle (
x : int,
y : int,
width : int,
height : int
)
```

В этом определении метода-конструктора Rectangle Constructor параметры переводятся так:

x – координата “x” верхнего левого угла прямоугольника;

y – координата “y” верхнего левого угла прямоугольника;

width – ширина (по оси “x”) прямоугольника;

height – высота (по оси “y”) прямоугольника.

Далее в нашей программе мы сначала объявим прямоугольники, описанные вокруг объектов, как новые переменные, например, так:

```
'Прямоугольник, описанный вокруг первого объекта:
Dim cheeseRectangle As Rectangle
'Прямоугольник, описанный вокруг второго объекта:
Dim breadRectangle As Rectangle
```

а затем в каком-либо методе создадим (при помощи ключевого слова new) и инициализируем эти объекты-прямоугольники, например, так:

```
cheeseRectangle = New Rectangle(cx, cy, _
cheeseImage.Width, cheeseImage.Height)
breadRectangle = New Rectangle(bx, by, _
breadImage.Width, breadImage.Height)
```

5.2. Обнаружение столкновения прямоугольников, описанных вокруг подвижных объектов

В этой структуре Rectangle (из пространства имён System.Drawing) имеются методы, которые могут обнаруживать пересечения различных перемещающихся прямоугольников. Эти методы определяют, находится ли точка одного прямоугольника внутри другого прямоугольника, и если находится, то программа определяет эту ситуацию и как столкновение этих двух прямоугольников, и как столкновение двух объектов, расположенных внутри этих прямоугольников.

Когда далее при написании программы мы поставим оператор-точку “.” после какого-либо объекта структуры Rectangle, то увидим подсказку с двумя основными методами Intersect и IntersectsWith (рис. 5.2) для обнаружения пересечения двух прямоугольников.

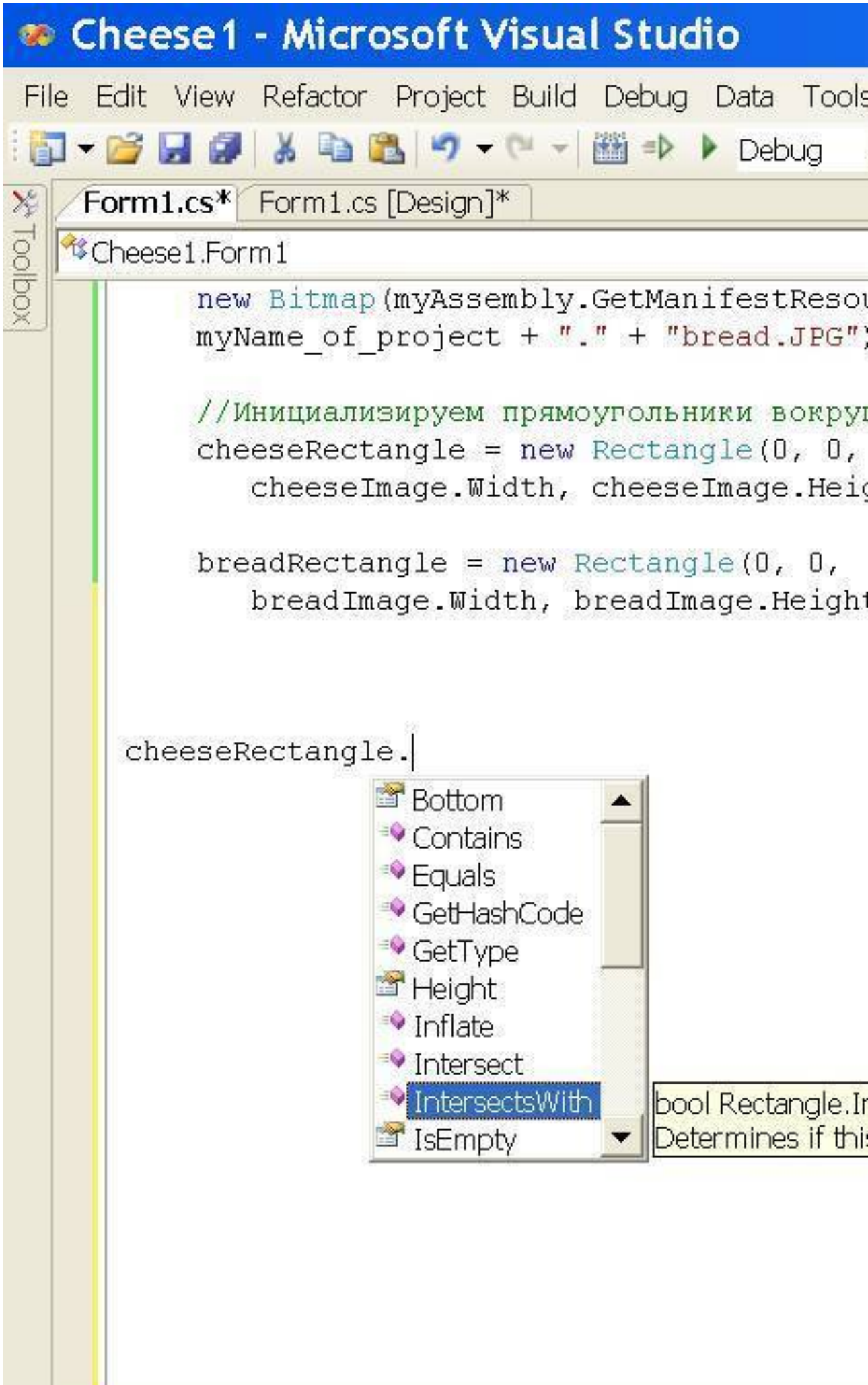

Cheese1 - Microsoft Visual Studio
File Edit View Refactor Project Build Debug Data Tools
Debug
Form1.cs* Form1.cs [Design]*
Toolbox
Cheese1.Form1
new Bitmap(myAssembly.GetManifestResou
myName_of_project + "." + "bread.JPG"
//Инициализируем прямоугольники вокру
cheeseRectangle = new Rectangle(0, 0,
cheeseImage.Width, cheeseImage.Heig
breadRectangle = new Rectangle(0, 0,
breadImage.Width, breadImage.Height
cheeseRectangle.
Bottom
Contains
Equals
GetHashCode
GetType
Height
Inflate
Intersect
IntersectsWith
IsEmpty
bool Rectangle.I
Determines if thi

Рис. 5.2. Подсказка с методами Intersect и IntersectsWith.

Определение для наиболее применяемого метода IntersectsWith (который далее и мы будем часто применять) с параметром (Rectangle *rect*) структуры Rectangle на главных (в мире программирования) языках приведено в табл. 5.2.

Таблица 5.2.

Определение метода **Rectangle.IntersectsWith** структуры Rectangle.

Visual Basic (Declaration)

```
Public Function IntersectsWith ( _
rect As Rectangle _
) As Boolean
```

Visual Basic (Usage)

```
Dim instance As Rectangle
Dim rect As Rectangle
Dim returnValue As Boolean
returnValue = instance.IntersectsWith(rect)
```

C#

```
public bool IntersectsWith (
Rectangle rect
)
```

C++

```
public:
bool IntersectsWith (
Rectangle rect
)
```

J#

```
public boolean IntersectsWith (
Rectangle rect
)
```

JScript

```
public function IntersectsWith (
rect : Rectangle
) : Boolean
```

Этот метод IntersectsWith обнаруживает пересечение заданного нами первого прямоугольника со вторым прямоугольником, объявленного здесь как параметр *rect*.

Если метод определит, что ни одна точка одного прямоугольника не находится внутри другого прямоугольника, то метод возвращает логическое значение False.

А если метод определит, что хотя бы одна точка одного прямоугольника находится внутри другого прямоугольника, то метод IntersectsWith возвращает логическое значение True, и это значение применяется для изменения направления движения какого-либо прямоугольника на противоположное (чтобы уйти от дальнейшего пересечения), например, в таком коде:

```
'Проверяем столкновение объектов:
If (cheeseRectangle.IntersectsWith(breadRectangle)) Then
'Изменяем направление движения на противоположное:
goingDown = Not goingDown
'В момент столкновения подаем звуковой сигнал Beep:
Beep()
End If
```

5.3. Код и выполнение программы

Теперь в проекте, который мы начали разрабатывать в предыдущей главе (и продолжаем в данной главе) объявляем два прямоугольника, а приведённый выше код в теле метода Form1_Paint заменяем на тот, который дан на следующем листинге (с подробными комментариями).

Листинг 5.1. Метод для рисования изображения.

```
'Прямоугольник, описанный вокруг первого объекта:
Dim cheeseRectangle As Rectangle
'Прямоугольник, описанный вокруг второго объекта:
Dim breadRectangle As Rectangle
Private Sub Form1_Paint(ByVal sender As System.Object, _
ByVal e As System.Windows.Forms.PaintEventArgs) _
Handles MyBase.Paint
'Загружаем в объекты класса System.Drawing.Image
'добавленные в проект файлы изображения заданного формата
'при помощи потока встроенного ресурса (ResourceStream):
cheeseImage = _
New Bitmap(myAssembly.GetManifestResourceStream( _
myName_of_project + "." + "cheese.JPG"))
breadImage = _
New Bitmap(myAssembly.GetManifestResourceStream( _
myName_of_project + "." + "bread.JPG"))
'Инициализируем прямоугольники, описанные вокруг объектов:
cheeseRectangle = New Rectangle(cx, cy, _
cheeseImage.Width, cheeseImage.Height)
breadRectangle = New Rectangle(bx, by, _
breadImage.Width, breadImage.Height)
'Если необходимо, создаём новый буфер:
If (backBuffer Is Nothing) Then
backBuffer = New Bitmap(Me.ClientSize.Width, _
Me.ClientSize.Height)
End If
'Создаём объект класса Graphics из буфера:
Using g As Graphics = Graphics.FromImage(backBuffer)
'Очищаем форму:
g.Clear(Color.White)
'Рисуем изображение в буфере backBuffer:
g.DrawImage(cheeseImage, cx, cy)
g.DrawImage(breadImage, bx, by)
End Using
'Рисуем изображение на форме Form1:
e.Graphics.DrawImage(backBuffer, 0, 0)
'Включаем таймер:
Timer1.Enabled = True
End Sub
```

А вместо приведённого выше метода updatePositions для изменения координат записываем следующий метод, дополненный кодом для обнаружения столкновения объектов.

Листинг 5.2. Метод для изменения координат и обнаружения столкновения объектов.

```
Sub updatePositions()
If (goingRight) Then
cx += xSpeed
Else
cx -= xSpeed
End If
If ((cx + cheeseImage.Width) >= Me.ClientSize.Width) Then
goingRight = False
'В момент столкновения подаем звуковой сигнал Beep:
Beep()
End If
If (cx <= 0) Then
goingRight = True
'В момент столкновения подаем звуковой сигнал Beep:
Beep()
End If
If (goingDown) Then
cy += ySpeed
Else
cy -= ySpeed
End If
If ((cy + cheeseImage.Height) >= Me.ClientSize.Height) Then
goingDown = False
'В момент столкновения подаем звуковой сигнал Beep:
Beep()
End If
If (cy <= 0) Then
goingDown = True
'В момент столкновения подаем звуковой сигнал Beep:
Beep()
End If
'Задаём прямоугольникам координаты объектов:
cheeseRectangle.X = cx
cheeseRectangle.Y = cy
breadRectangle.X = bx
breadRectangle.Y = by
'Проверяем столкновение объектов:
If (cheeseRectangle.IntersectsWith(breadRectangle)) Then
'Изменяем направление движения на противоположное:
goingDown = Not goingDown
'В момент столкновения подаем звуковой сигнал Beep:
Beep()
End If
End Sub
```

В режиме выполнения (Build, Build Selection; Debug, Start Without Debugging) при помощи кнопок и мыши мы можем перемещать хлеб и этим хлебом, как ракеткой, отбивать

сыр или вверх, или вниз (рис. 5.3). Напомним, что, так как угол падения сыра на хлеб равен 45 градусам, то и угол отражения сыра от хлеба (и от границ экрана) также равен 45 градусам.

5.4. Основные схемы столкновений и их реализация

Приведённый на предыдущем листинге код обнаруживает столкновение только тогда, когда сыр падает на хлеб сверху вниз и соприкасается с верхней плоскостью хлеба. Если же сыр соприкасается с хлебом сбоку (слева или справа), то отскока сыра от хлеба не происходит. Поэтому устраним этот недостаток, чтобы игра была более реалистичной.

Если мы оперируем с окружностями, описанными вокруг объектов, то возможны три основные схемы столкновений, показанные на рис. 5.4. В схемах 1 и 3 маленький круг ударяется о большой круг под углом 45 градусов и отражается под этим же углом и по этой же линии. В схеме 2 маленький круг ударяется о большой круг под углом 90 градусов и также вертикально отражается вверх.

Если же мы оперируем с прямоугольниками, описанными вокруг объектов, то возможны четыре основные схемы столкновений, показанные на рис. 5.5.

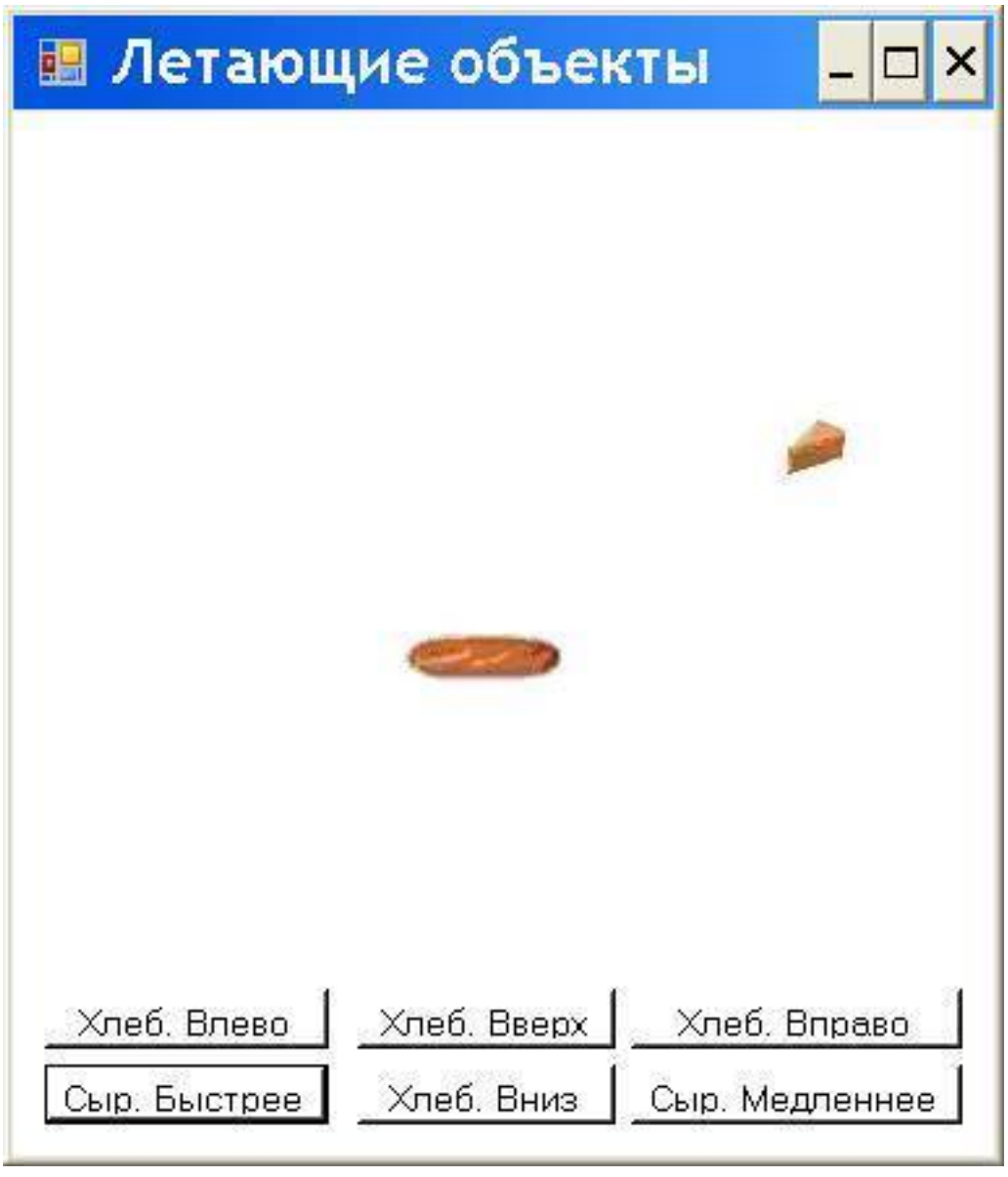


Рис. 5.3. Сыр отскочил от хлеба.

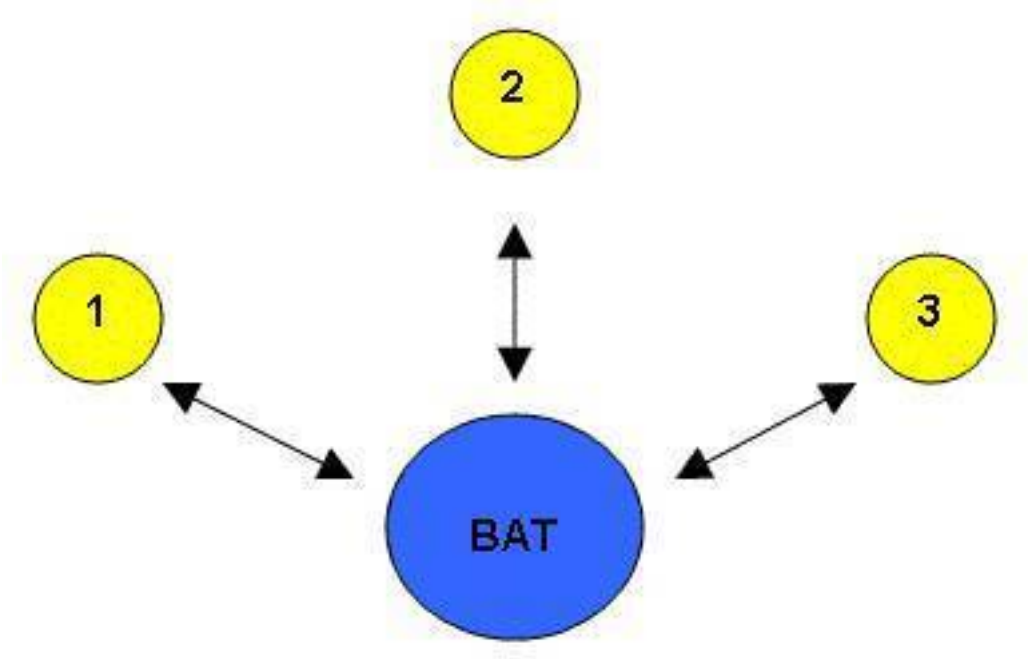
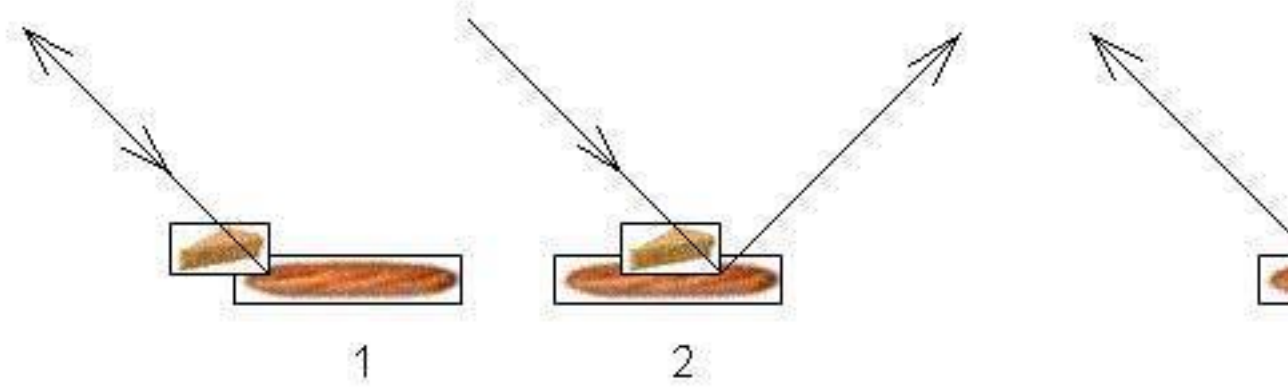


Рис. 5.4. Три схемы столкновений.



5.5. Четыре схемы столкновений.

В схемах 1 и 4 маленький прямоугольник ударяется о большой прямоугольник сбоку под углом 45 градусов и отражается под этим же углом и по этой же линии. В схемах 2 и 3 маленький прямоугольник падает на большой прямоугольник под углом 45 градусов, но отражается не по линии падения, а по линии отражения, перпендикулярной линии падения.

Для реализации более правильных схем столкновений, показанных на рис. 5.5, в нашем проекте вместо приведённого выше метода updatePositions для изменения координат записываем следующий метод, дополненный новым кодом для обнаружения столкновения объектов.

Листинг 5.3. Метод для изменения координат и обнаружения столкновения объектов.

```
Sub updatePositions()
If (goingRight) Then
cx += xSpeed
Else
cx -= xSpeed
```

```
End If
If ((cx + cheeseImage.Width) >= Me.ClientSize.Width) Then
goingRight = False
'В момент столкновения подаем звуковой сигнал Beep:
Beep()
End If
If (cx <= 0) Then
goingRight = True
'В момент столкновения подаем звуковой сигнал Beep:
Beep()
End If
If (goingDown) Then
cy += ySpeed
Else
cy -= ySpeed
End If
If ((cy + cheeseImage.Height) >= Me.ClientSize.Height) Then
goingDown = False
'В момент столкновения подаем звуковой сигнал Beep:
Beep()
End If
If (cy <= 0) Then
goingDown = True
'В момент столкновения подаем звуковой сигнал Beep:
Beep()
End If
'Проверяем столкновение объектов:
If (goingDown) Then
'Если сыр движется вниз и имеется столкновение:
If (cheeseRectangle.IntersectsWith(breadRectangle)) Then
'В момент столкновения подаем звуковой сигнал Beep:
Beep()
'мы имеем столкновение:
Dim rightIn As Boolean = breadRectangle.Contains( _
cheeseRectangle.Right, _
cheeseRectangle.Bottom)
Dim leftIn As Boolean = breadRectangle.Contains( _
cheeseRectangle.Left, _
cheeseRectangle.Bottom)
'виды столкновений:
If (rightIn And leftIn) Then
'отскок вверх:
goingDown = False
Else
'отскок вверх:
goingDown = False
'отскоки по горизонтали:
If (rightIn) Then
goingRight = False
```

```
End If
If (leftIn) Then
goingRight = True
End If
End If
End If
End If
End Sub
```

В режиме выполнения (Build, Build Selection; Debug, Start Without Debugging) при помощи кнопок Button и мыши мы можем перемещать хлеб и этим хлебом, как ракеткой, отбивать сыр вверх не только верхней стороной прямоугольника (описанного вокруг объекта), как было в предыдущем коде, но теперь и боковыми сторонами этого прямоугольника. Однако мы можем отбивать, только если сыр перемещается сверху вниз.

5.5. Добавление новых объектов

Продолжаем усложнять игру за счёт добавления в неё новых объектов в виде продуктов питания, например, помидоров (tomatoes) в виде файла tomato.gif, рис. 5.6.

Рис. 5.6.
Помидор.

В начале игры несколько i-х помидоров в виде массива tomatoes[i] должны появиться в верхней части экрана в качестве мишеней (рис. 5.7), которые должны исчезать после попадания в них летающего сыра (рис. 5.8).

Попадание сыра в помидор определяется уже применяемым выше методом IntersectWith.

Исчезновение помидоров выполняется при помощи свойства visible, которому присваивается логическое значение False (в коде: tomatoes(i).visible = False).

Управляя при помощи кнопок Button и мыши перемещением батона хлеба, игрок может отражать сыр вверх таким образом, чтобы уничтожить как можно больше помидоров за меньшее время, набирая при этом очки.

Добавляем в наш проект (из отмеченной выше статьи или из Интернета) файл изображения помидора tomato.gif по стандартной схеме, а именно: в меню Project выбираем Add Existing Item, в этой панели в окне “Files of type” выбираем “All Files”, в центральном окне находим и выделяем имя файла и щёлкаем кнопку Add (или дважды щёлкаем по имени файла). В панели Solution Explorer мы увидим этот файл.

Теперь этот же файл tomato.gif встраиваем в проект в виде ресурса по разработанной выше схеме, а именно: в панели Solution Explorer выделяем появившееся там имя файла, а в панели Properties (для данного файла) в свойстве Build Action (Действие при постро-

нии) вместо заданного по умолчанию выбираем значение Embedded Resource (Встроенный ресурс).

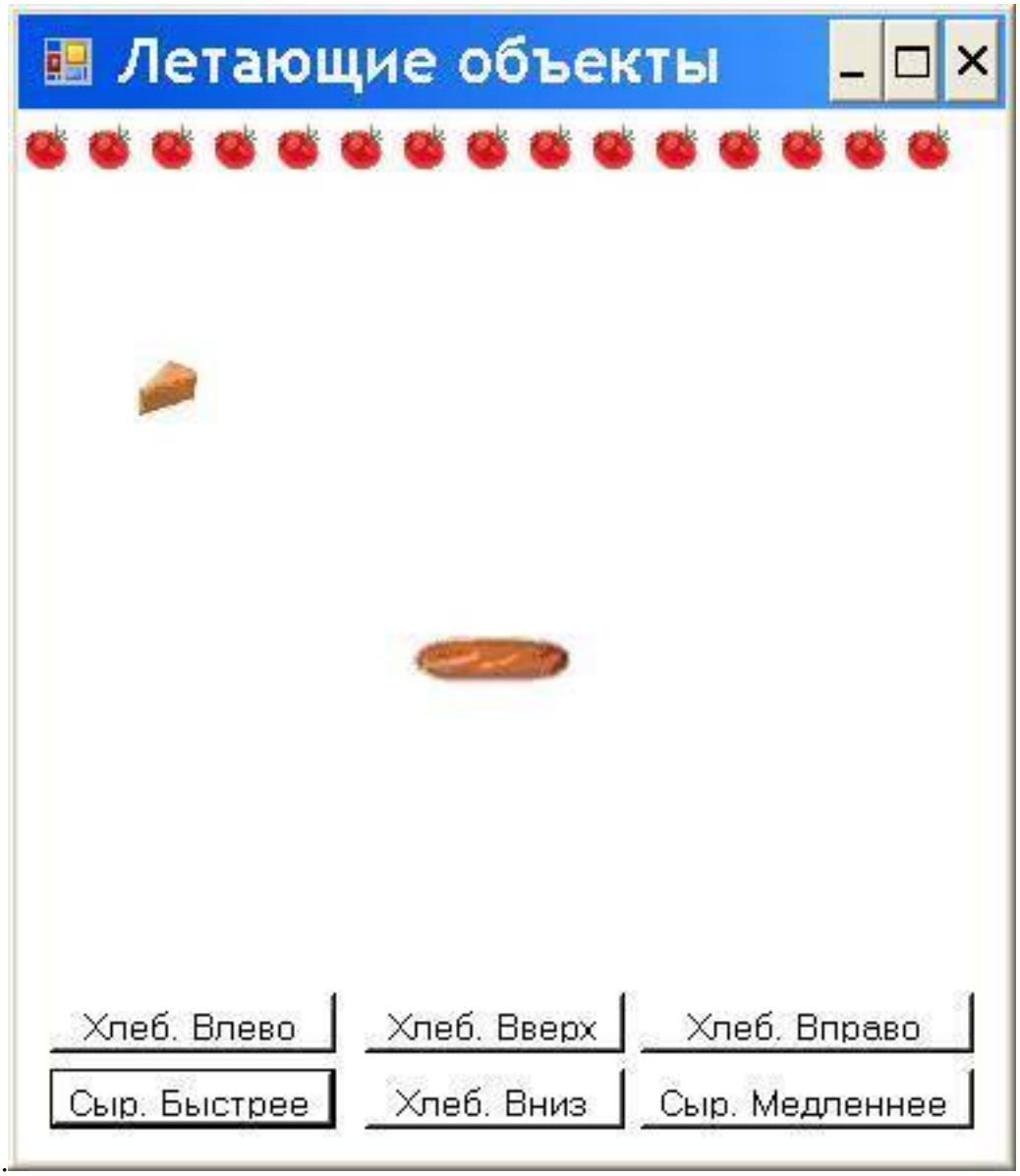


Рис. 5.7. Помидоры – мишени.

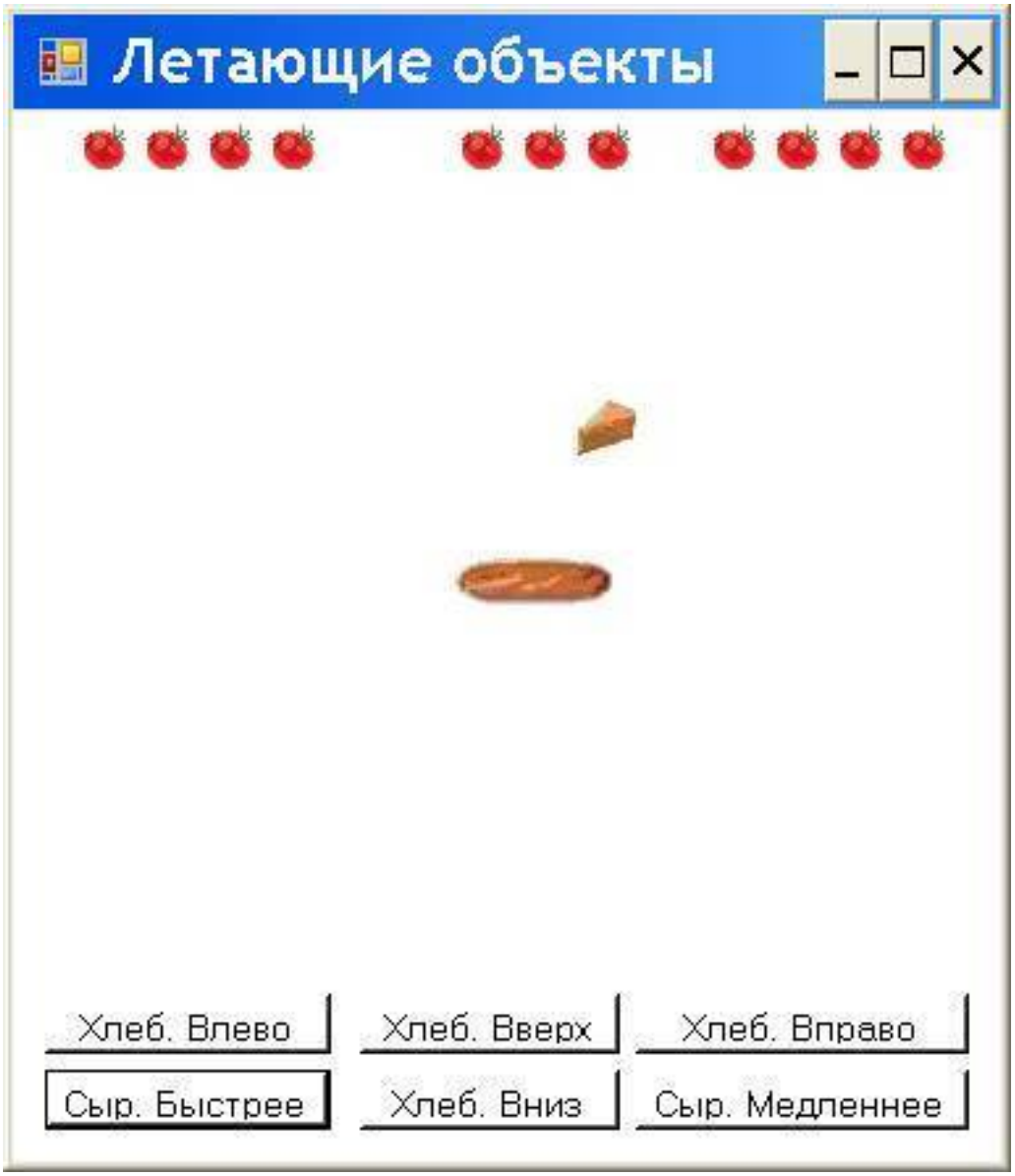


Рис. 5.8. Помидоры исчезают после попадания в них сыра.

Для программной реализации рисования и уничтожения помидоров после попадания в них сыра, в классе Form1 нашего проекта записываем следующий код.

Листинг 5.4. Переменные и методы для помидоров (tomatoes).

```
'Объявляем объект класса System.Drawing.Image для продукта:
Dim tomatoImage As Image
'Position and state of tomato
Structure tomato
Public rectangle As Rectangle
Public visible As Boolean
End Structure
' Spacing between tomatoes. Set once for the game
Dim tomatoSpacing As Integer = 4
' Height at which the tomatoes are drawn. Will change
' as the game progresses. Starts at the top.
Dim tomatoDrawHeight As Integer = 4
```

```
' The number of tomatoes on the screen. Set at the start
' of the game by initialiseTomatoes.
Dim noOfTomatoes As Integer
' Positions of the tomato targets.
Dim tomatoes() As tomato
' called once to set up all the tomatoes.
Sub initialiseTomatoes()
noOfTomatoes = (Me.ClientSize.Width – tomatoSpacing) / _
(tomatoImage.Width + tomatoSpacing)
' create an array to hold the tomato positions
ReDim tomatoes(noOfTomatoes)
' x coordinate of each potato
Dim tomatoX As Integer = tomatoSpacing / 2
Dim i As Integer
For i = 0 To tomatoes.Length – 1
tomatoes(i).rectangle = _
New Rectangle(tomatoX, tomatoDrawHeight, _
tomatoImage.Width, tomatoImage.Height)
tomatoX = tomatoX + tomatoImage.Width + tomatoSpacing
Next
End Sub
' Called to place a row of tomatoes.
Sub placeTomatoes()
Dim i As Integer
For i = 0 To tomatoes.Length – 1
tomatoes(i).rectangle.Y = tomatoDrawHeight
tomatoes(i).visible = True
Next
End Sub
```

Приведённый выше код в теле метода Form1_Paint заменяем на тот, который дан на следующем листинге.

Листинг 5.5. Метод для рисования изображения.

```
Private Sub Form1_Paint(ByVal sender As System.Object, _
ByVal e As System.Windows.Forms.PaintEventArgs) _
Handles MyBase.Paint
'Если необходимо, создаём новый буфер:
If (backBuffer Is Nothing) Then
backBuffer = New Bitmap(Me.ClientSize.Width, _
Me.ClientSize.Height)
End If
'Создаём объект класса Graphics из буфера:
Using g As Graphics = Graphics.FromImage(backBuffer)
'Очищаем форму:
g.Clear(Color.White)
'Рисуем изображение в буфере backBuffer:
g.DrawImage(cheeseImage, cx, cy)
g.DrawImage(breadImage, bx, by)
Dim i As Integer
For i = 0 To tomatoes.Length – 1
```

```
If (tomatoes(i).visible) Then
g.DrawImage(tomatoImage, _
tomatoes(i).rectangle.X, _
tomatoes(i).rectangle.Y)
End If
Next
End Using
'Рисуем изображение на форме Form1:
e.Graphics.DrawImage(backBuffer, 0, 0)
End Sub
```

Добавление новых объектов в игру соответственно усложняет код. В панели Properties (для Form1) на вкладке Events дважды щёлкаем по имени события Load. Появившийся шаблон метода Form1_Load после записи нашего кода принимает следующий вид.

Листинг 5.6. Метод для рисования изображения.

```
Private Sub Form1_Load(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles MyBase.Load
'Загружаем в объекты класса System.Drawing.Image
'добавленные в проект файлы изображения заданного формата
'при помощи потока встроенного ресурса (ResourceStream):
cheeseImage = _
New Bitmap(myAssembly.GetManifestResourceStream( _
myName_of_project + "." + "cheese.JPG"))
breadImage = _
New Bitmap(myAssembly.GetManifestResourceStream( _
myName_of_project + "." + "bread.JPG"))
'Инициализируем прямоугольники, описанные вокруг объектов:
cheeseRectangle = New Rectangle(cx, cy, _
cheeseImage.Width, cheeseImage.Height)
breadRectangle = New Rectangle(bx, by, _
breadImage.Width, breadImage.Height)
'Загружаем помидор:
tomatoImage = _
New Bitmap(myAssembly.GetManifestResourceStream( _
myName_of_project + "." + "tomato.gif"))
'Инициализируем массив помидоров и прямоугольников:
initialiseTomatoes()
'Размещаем помидоры в верхней части экрана:
placeTomatoes()
'Включаем таймер:
Timer1.Enabled = True
End Sub
```

И наконец, вместо приведённого выше метода updatePositions записываем следующий метод, дополненный новым кодом для изменения координат, обнаружения столкновений объектов и уничтожения помидоров.

Листинг 5.7. Метод для изменения координат и обнаружения столкновения объектов.

```
Sub updatePositions()
If (goingRight) Then
cx += xSpeed
Else
```

```
cx -= xSpeed
End If
If ((cx + cheeseImage.Width) >= Me.ClientSize.Width) Then
goingRight = False
'В момент столкновения подаем звуковой сигнал Beep:
Beep()
End If
If (cx <= 0) Then
goingRight = True
'В момент столкновения подаем звуковой сигнал Beep:
Beep()
End If
If (goingDown) Then
cy += ySpeed
Else
cy -= ySpeed
End If
If ((cy + cheeseImage.Height) >= Me.ClientSize.Height) Then
goingDown = False
'В момент столкновения подаем звуковой сигнал Beep:
Beep()
End If
If (cy <= 0) Then
goingDown = True
'В момент столкновения подаем звуковой сигнал Beep:
Beep()
End If
'Задаём прямоугольникам координаты объектов:
cheeseRectangle.X = cx
cheeseRectangle.Y = cy
breadRectangle.X = bx
breadRectangle.Y = by
'Проверяем столкновение объектов с учётом помидоров:
If (goingDown) Then
' only bounce if the cheese is going down
If (cheeseRectangle.IntersectsWith(breadRectangle)) Then
'В момент столкновения подаем звуковой сигнал Beep:
Beep()
' we have a collision
Dim rightIn As Boolean = breadRectangle.Contains( _
cheeseRectangle.Right, _
cheeseRectangle.Bottom)
Dim leftIn As Boolean = breadRectangle.Contains( _
cheeseRectangle.Left, _
cheeseRectangle.Bottom)
' now deal with the bounce
If (rightIn And leftIn) Then
' bounce up
goingDown = False
```

```
Else
' bounce up
goingDown = False
' now sort out horizontal bounce
If (rightIn) Then
goingRight = False
End If
If (leftIn) Then
goingRight = True
End If
End If
End If
Else
' only destroy tomatoes of the cheese is going up
Dim i As Integer
For i = 0 To tomatoes.Length – 1
If (Not tomatoes(i).visible) Then
Continue For
End If
If (cheeseRectangle.IntersectsWith( _
tomatoes(i).rectangle)) Then
'В момент столкновения подаем звуковой сигнал Beep:
Beep()
' hide the tomato
tomatoes(i).visible = False
' bounce down
goingDown = True
' only destroy one at a time
'Чтобы помидоры уничтожались не по два,
'а по одному помидору: Exit For:
Exit For
End If
Next
End If
End Sub 'Конец метода updatePositions.
```

В режиме выполнения (Build, Build Selection; Debug, Start Without Debugging) несколько их помидоров появляются в верхней части экрана в качестве мишеней (рис. 5.7), которые исчезают после попадания в них летающего сыра (рис. 5.8).

Управляя при помощи кнопок Button и мыши перемещением батона хлеба, мы можем отражать сыр вверх таким образом, чтобы уничтожить как можно больше помидоров за меньшее время, набирая при этом очки.

К разработке методики подсчёта очков в игре мы и приступаем.

5.6. Методика подсчёта очков в игре

Игра отличается от любого другого приложения тем, что один или несколько игроков набирают в игре очки, и победителем считается игрок, набравший наибольшее количество очков. А после набора определённого количества очков игра может переходить на более высо-

кие (более сложные) и интересные уровни, после прохождения которых игрок может получить приз, например, в виде изображения какого-нибудь смешного персонажа.

Методика подсчёта очков (score) в игре подразумевает наличие в программе счётчика (scorer) очков и вывода очков на экран (например, методом DrawString) в строке:

```
g.DrawString(messageString, messageFont, messageBrush,
messageRectangle)
```

Видно, что в этом методе DrawString мы дожны определить параметры в виде шрифта messageFont, кисти messageBrush и зарезервированного прятоугольника для записи очков messageRectangle, причём в этот прямоугольник летающие объекты не должны залетать. На рис. 5.9 мы получили 20 очков за 2 сбитых помидора, а на 5.10 – 50 очков за 5 сбитых помидоров.

За каждый сбитый помидор мы можем начислить игроку любое количество очков, например, 10 очков в строке:

```
scoreValue = scoreValue + 10
```

Новые очки сразу же выводятся на экран, информируя игрока.

Приступим к программной реализации методики подсчёта очков в игре в нашем базовом учебном проекте.

Сначала мы должны опустить ряд помидоров пониже, чтобы освободить место вверху для записи очков, поэтому вместо 4 записываем ординату, равную, например, 20:

```
Dim tomatoDrawHeight As Integer = 20
```

В любом месте класса Form1 добавляем новые переменные для счётчика очков.

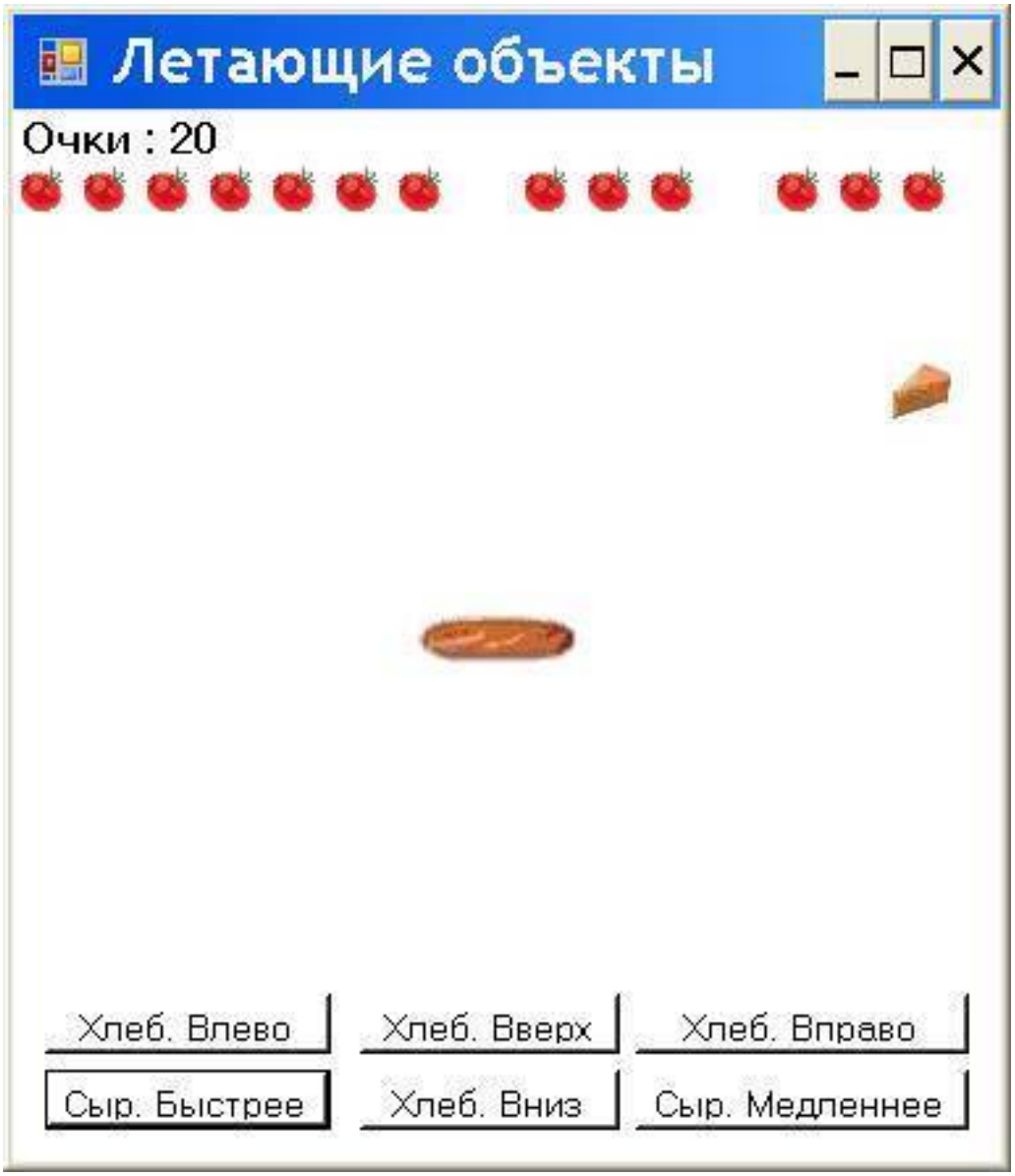


Рис. 5.9. Получили 20 очков за 2 сбитых помидора.

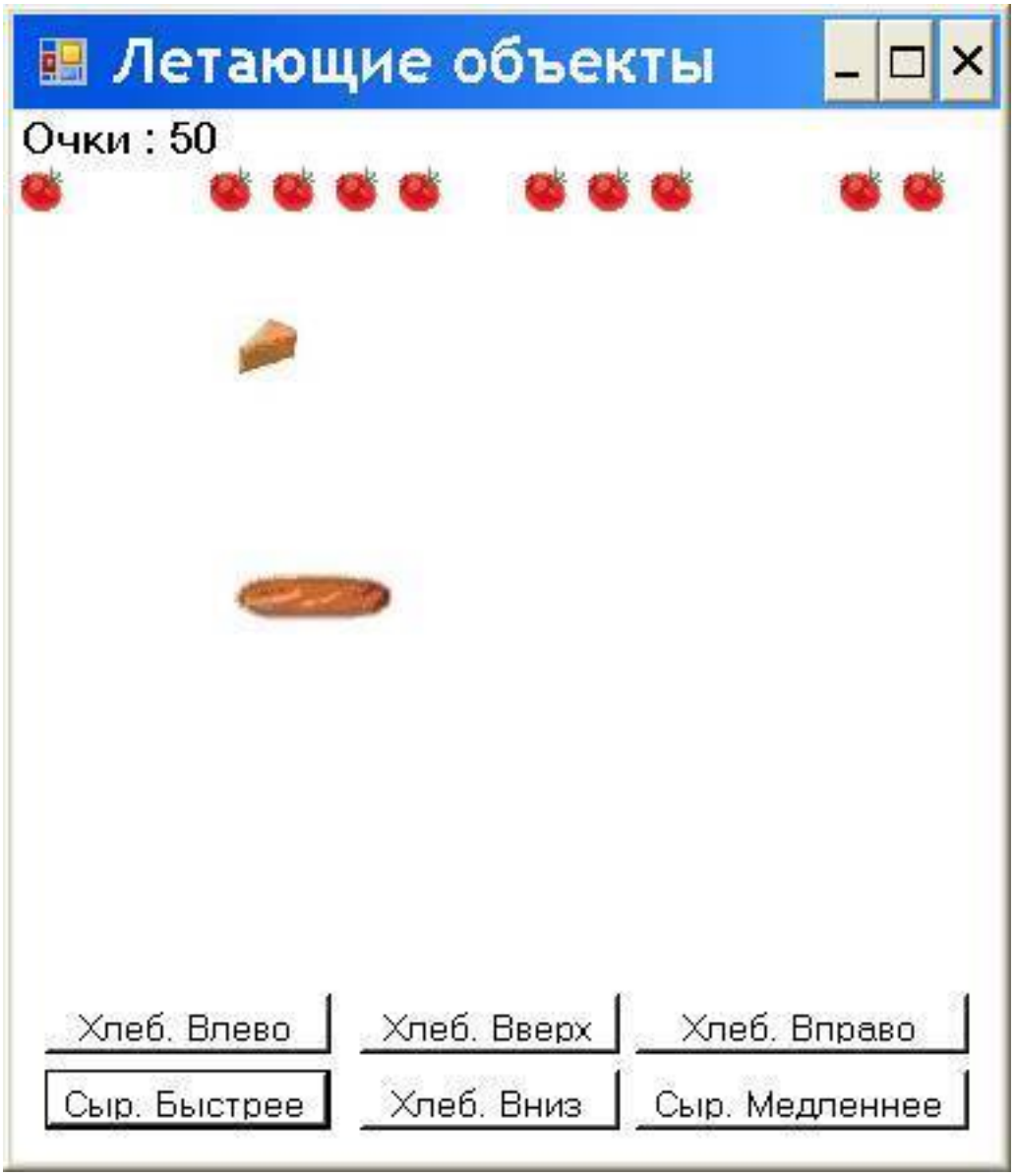


Рис. 5.10. Получили 50 очков.

Листинг 5.8. Новые переменные.

```
' Font for score messages.
Dim messageFont As Font = Nothing
' Rectangle for score display.
Dim messageRectangle As Rectangle
' Height of the score panel.
Dim scoreHeight As Integer = 20 '= заданной выше tomatoDrawHeight.
' Brush used to draw the messages.
Dim messageBrush As SolidBrush
' The string which is drawn as the user message.
Dim messageString As String = "Очки: 0"
' Score in a game.
Dim scoreValue As Integer = 0
```

Приведённый выше код в теле метода Form1_Paint заменяем на тот, который дан на следующем листинге.

Листинг 5.9. Метод для рисования изображения.

```
Private Sub Form1_Paint(ByVal sender As System.Object, _
ByVal e As System.Windows.Forms.PaintEventArgs) _
Handles MyBase.Paint
'Если буфер пустой, создаём новый буфер:
If (backBuffer Is Nothing) Then
backBuffer = New Bitmap(Me.ClientSize.Width, _
Me.ClientSize.Height)
End If
'Создаём объект класса Graphics из буфера:
Using g As Graphics = Graphics.FromImage(backBuffer)
'Очищаем форму:
g.Clear(Color.White)
'Рисуем изображения объектов в буфере backBuffer:
g.DrawImage(cheeseImage, cx, cy)
g.DrawImage(breadImage, bx, by)
Dim i As Integer
For i = 0 To tomatoes.Length – 1
If (tomatoes(i).visible) Then
g.DrawImage(tomatoImage, _
tomatoes(i).rectangle.X, _
tomatoes(i).rectangle.Y)
End If
Next
'Записываем очки игрока:
g.DrawString(messageString, messageFont, _
messageBrush, messageRectangle)
End Using
'Рисуем изображение на форме Form1:
e.Graphics.DrawImage(backBuffer, 0, 0)
End Sub
```

Приведённый выше код в теле метода Form1_Load (для загрузки файлов изображений игровых объектов) заменяем на тот, который дан на следующем листинге.

Листинг 5.10. Метод для загрузки файлов изображений.

```
Private Sub Form1_Load(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles MyBase.Load
'Загружаем в объекты класса System.Drawing.Image
'добавленные в проект файлы изображений заданного формата
'при помощи потока встроенного ресурса (ResourceStream):
cheeseImage = _
New Bitmap(myAssembly.GetManifestResourceStream( _
myName_of_project + "." + "cheese.JPG"))
breadImage = _
New Bitmap(myAssembly.GetManifestResourceStream( _
myName_of_project + "." + "bread.JPG"))
'Инициализируем прямоугольники, описанные вокруг объектов:
cheeseRectangle = New Rectangle(cx, cy, _
cheeseImage.Width, cheeseImage.Height)
breadRectangle = New Rectangle(bx, by, _
```

```
breadImage.Width, breadImage.Height)
'Загружаем файл изображения нового объекта:
tomatoImage = _
New Bitmap(myAssembly.GetManifestResourceStream( _
myName_of_project + "." + "tomato.gif"))
'Инициализируем массив новых объектов и прямоугольников,
'описанные вокруг этих объектов:
initialiseTomatoes()
'Размещаем новые объекты в верхней части экрана:
placeTomatoes()
'Создаём и инициализируем шрифт для записи очков:
messageFont = New Font(FontFamily.GenericSansSerif, 10, _
FontStyle.Regular)
'Резервируем прямоугольник на экране для записи очков:
messageRectangle = New Rectangle(0, 0, _
Me.ClientSize.Width, scoreHeight)
'Задаём цвет кисти для записи очков:
messageBrush = New SolidBrush(Color.Black)
'Включаем таймер:
Timer1.Enabled = True
End Sub
```

И наконец, вместо приведённого выше метода updatePositions записываем следующий метод, дополненный новым кодом для изменения координат, обнаружения столкновений объектов, уничтожения помидоров и подсчёта очков.

Листинг 5.11. Метод для изменения координат и обнаружения столкновения объектов.

```
Sub updatePositions()
If (goingRight) Then
cx += xSpeed
Else
cx -= xSpeed
End If
If ((cx + cheeseImage.Width) >= Me.ClientSize.Width) Then
goingRight = False
'В момент столкновения подаем звуковой сигнал Beep:
Beep()
End If
If (cx <= 0) Then
goingRight = True
'В момент столкновения подаем звуковой сигнал Beep:
Beep()
End If
If (goingDown) Then
cy += ySpeed
Else
cy -= ySpeed
End If
If ((cy + cheeseImage.Height) >= Me.ClientSize.Height) Then
goingDown = False
'В момент столкновения подаем звуковой сигнал Beep:
```

```
Beep()
End If
If (cy <= 0) Then
goingDown = True
'В момент столкновения подаем звуковой сигнал Beep:
Beep()
End If
'Задаём прямоугольникам координаты объектов:
cheeseRectangle.X = cx
cheeseRectangle.Y = cy
breadRectangle.X = bx
breadRectangle.Y = by
' check for collisions.
If (goingDown) Then
' only bounce if the cheese is going down
If (cheeseRectangle.IntersectsWith(breadRectangle)) Then
'В момент столкновения подаем звуковой сигнал Beep:
Beep()
' we have a collision
Dim rightIn As Boolean = breadRectangle.Contains( _
cheeseRectangle.Right, _
cheeseRectangle.Bottom)
Dim leftIn As Boolean = breadRectangle.Contains( _
cheeseRectangle.Left, _
cheeseRectangle.Bottom)
' now deal with the bounce
If (rightIn And leftIn) Then
' bounce up
goingDown = False
Else
' bounce up
goingDown = False
' now sort out horizontal bounce
If (rightIn) Then
goingRight = False
End If
If (leftIn) Then
goingRight = True
End If
End If
End If
Else
' only destroy tomatoes of the cheese is going up
Dim i As Integer
For i = 0 To tomatoes.Length – 1
If (Not tomatoes(i).visible) Then
Continue For
End If
If (cheeseRectangle.IntersectsWith( _
```

```
tomatoes(i).rectangle)) Then
'В момент столкновения подаем сигнал Beep:
Beep()
' hide the tomato
tomatoes(i).visible = False
' bounce down
goingDown = True
' update the score
scoreValue = scoreValue + 10
'Конвертируем очки scoreValue в тип String
'и выводим на экран:
messageString = "Очки: " & _
Convert.ToString(scoreValue)
' only destroy one at a time
'Чтобы помидоры уничтожались не по два,
'а по одному помидору: Exit For:
Exit For
End If
Next
End If
End Sub 'Конец метода updatePositions.
```

В режиме выполнения (Build, Build Selection; Debug, Start Without Debugging), управляя летающим сыром при помощи батона хлеба, кнопок Button и мыши, мы получили 20 очков за 2 сбитых помидора (рис. 5.9) и 50 очков за 5 сбитых помидоров (рис. 5.10).

Отметим, что для управления игрой в дополнение или вместо кнопок Button (чтобы не загромождать форму Form1) можно использовать также и клавиши клавиатуры по описанной далее методике.

Итак, в этой главе мы разработали методику обнаружения столкновений, программирования уничтожений летающих объектов и подсчёта очков.

Глава 6. Методология воспроизведения звуковых файлов

6.1. Основные методики звукового сопровождения приложений

Выше мы уже применяли воспроизведение звуковых файлов по упрощённому варианту и звуковой эффект в виде звукового сигнала Beep (по-русски: Бип). Напомним, что этот сигнал появляется, когда выполнение программы дойдёт до строки: Beep(). Записывая эту строку в соответствующих местах программы, мы подаем звуковой сигнал Beep в различные моменты анимации, например, в момент каждого удара объектов о границу (внутри которой перемещаются объекты) или друг о друга.

В данной главе мы дополним воспроизведение звуковых файлов по упрощённому варианту и подачу звукового сигнала Beep воспроизведением звуковых файлов по более универсальному и сложному варианту в различные моменты выполнения нашего приложения. Естественно, кое-где можно закомментировать приведённую выше строку с методом Beep и оставить проигрывание только звуковых файлов (без наложения на них сигнала Beep).

Для управления проигрыванием звуковых файлов в приложениях и играх на настольных компьютерах, ноутбуках и планшетах разработаны три основные методики:

Методика воспроизведения звуковых файлов на основе пространства имён My.

Методика воспроизведения звуковых файлов на основе встроенного ресурса.

Методика воспроизведения звуковых файлов на основе DirectX.

У каждой из этих методик есть свои области рационального применения, есть свои преимущества и недостатки, которые мы и рассмотрим сейчас.

6.2. Методика воспроизведения звуковых файлов на основе пространства имён My

Сначала для воспроизведения звуков в приложениях и играх для настольных компьютеров, ноутбуков и планшетов мы опишем наиболее популярную методику воспроизведения звуковых файлов на основе пространства имён My (без использования и технологии DirectX, и схемы встроенного (Embedded) ресурса). Приступим к программной реализации этой методики, для общности, в новом проекте.

Для создания проекта в VS щёлкаем кнопку New Project (или File, New, Project). В панели New Project в окне Project Types выбираем тип проекта Visual Basic, Windows, в окне Templates выделяем шаблон Windows Forms Application, в окне Name записываем любое имя проекта, например, Sounds2 и щёлкаем OK. Создаётся проект, появляется форма Form1 (рис. 6.1) в режиме проектирования.

Рис. 6.1.

Форма Form1 в режиме выполнения.

Проектируем (или оставляем по умолчанию) эту форму, как описано в параграфе “Методика проектирования формы”. Например, в панели Properties в свойстве Font можно оставить по умолчанию или установить новый шрифт и его размер (Size). Чтобы изменить заголовок формы, в панели Properties в свойстве Text записываем (или вставляем из буфера обмена: правый щелчок, Paste) текст.

С панели инструментов Toolbox переносим на форму, для примера, две кнопки Button (Sound 1 и Sound 2). В панели Properties в свойстве Text записываем название каждой кнопки Sound&1 и Sound&2 с оператором &, который подчёркивает следующий за ним символ текста, что позволяет нам задействовать кнопку не только мышью, но и, удерживая нажатой клавишу Alt, нажатием клавиши с подчёркнутой буквой английского алфавита или цифрой (например, Alt+1).

На форме в режиме выполнения по умолчанию выделена первая кнопка. Но если мы желаем, чтобы в режиме выполнения на форме была выделена другая кнопка (чтобы нажимать её не только мышью, но и клавишей Enter), то в панели Properties (для Form1) в свойстве AcceptButton выбираем имя этой кнопки.

По варианту 1, для добавления звукового файла drumpad-crash.wav в проект, в меню Project выбираем Add Existing Item, в панели Add Existing Item в окне Files of type устанавливаем All Files, в окне "Look in" находим (в системной папке компьютера C, WINDOWS, Media или в папке с загруженным из Интернета файлом) файл и щёлкаем кнопку Add (или дважды щёлкаем по имени файла). Этот файл мы увидим в панели Solution Explorer. Если мы дважды щёлкнем по этому файлу drumpad-crash.wav, то откроется проигрыватель Windows Media Player, и мы услышим в течение нескольких секунд звучание типа шуршания оркестровых металлических тарелок при воздействии на них металлической метелки.

Аналогично добавляем в проект второй файл drumpad-bass_drum.wav типа удара барабана.

Аналогично можно добавить в проект ещё много звуковых файлов, которые мы желаем послушать во время выполнения приложения или игры.

По второму, закомментированному далее в программе, варианту эти файлы мы не добавляем в проект, а копируем во внешнюю папку с именем, например, Sounds, запоминая путь к этой папке, начиная от локального диска, например, D.

Теперь в панели Properties (для формы Form1) на вкладке Events дважды щёлкаем по имени события Load (Загрузка).

Появляется файл Form1.vb с шаблоном метода Form1_Load, который после записи нашего кода принимает следующий вид.

Листинг 6.1. Метод для загрузки и воспроизведения звуковых файлов.

```
Private Sub Form1_Load(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles MyBase.Load
'Загружаем звуковые файлы формата (.wav)
'по 1-му варианту непосредственно из проекта:
My.Computer.Audio.Play("..\..\drumpad-crash.wav")

'Загружаем звуковые файлы формата (.wav)
'по 2-му варианту из внешней папки:
'My.Computer.Audio.Play( _
'"D:\MyDocs\Sounds\drumpad-bass_drum.wav")
End Sub
```

Видно, что в строке:

```
My.Computer.Audio.Play("..\..\drumpad-crash.wav")
```

мы используем пространство имён My, свойство Computer, свойство Audio и метод Play (рис. 6.2).

Оператор “..\..\” называется relative path – путь относительно (внутри) проекта.

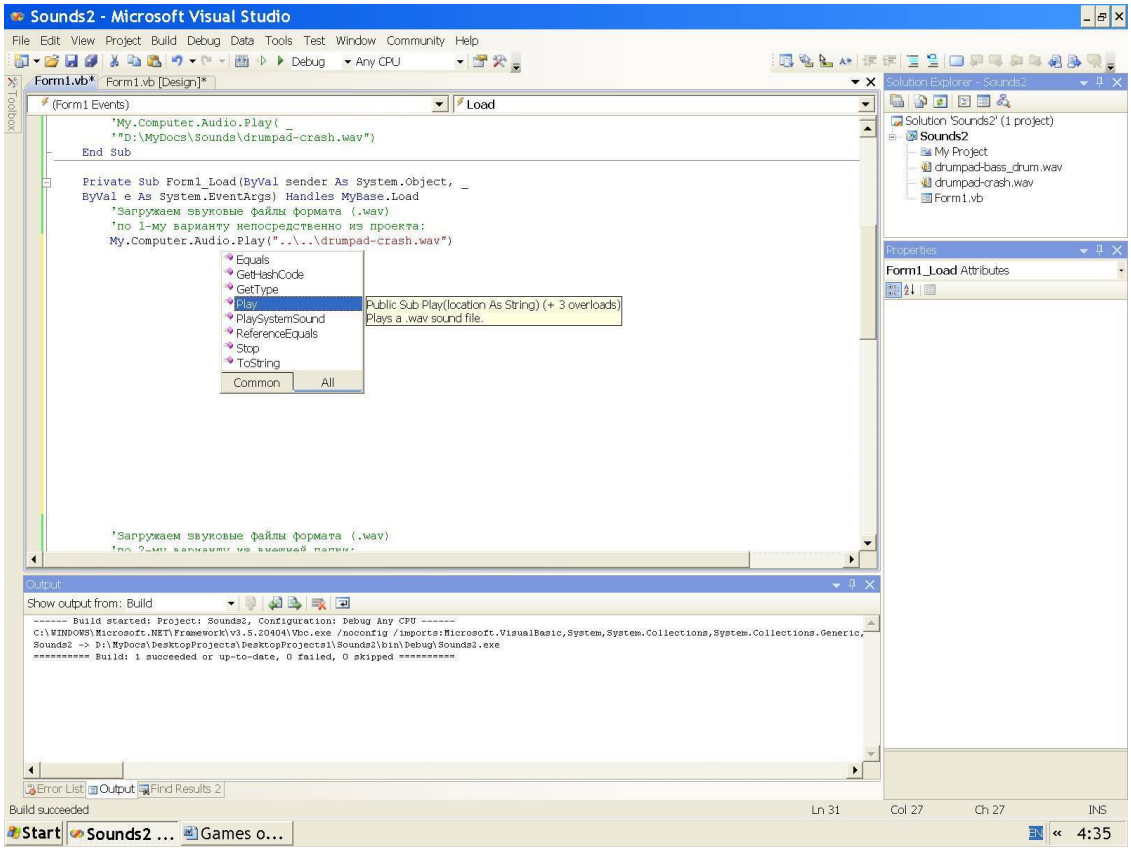


Рис. 6.2. Подсказка с методом Play.

В режиме выполнения (Build, Build Selection; Debug, Start Without Debugging) мы услышим соответствующее (одноразовое) воспроизведение звукового файла, который мы добавили непосредственно в проект (а не в дополнительную папку Sounds проекта).

Если в приведённый выше шаблон метода Form1_Load мы загрузим подряд два файла, как показано ниже:

```
Private Sub Form1_Load(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles MyBase.Load
My.Computer.Audio.Play("..\..\drumpad-crash.wav")
My.Computer.Audio.Play("..\..\drumpad-bass_drum.wav")
End Sub
```

то в режиме выполнения мы услышим только второй звуковой файл.

Чтобы мы услышали сначала полностью первый звуковой файл, а затем второй звуковой файл, мы должны в методе Play в качестве второго параметра использовать константу WaitToComplete из перечисления режимов AudioPlayMode, как показано в следующем коде.

Листинг 6.2. Метод для загрузки и воспроизведения звуковых файлов.

```
Private Sub Form1_Load(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles MyBase.Load
My.Computer.Audio.Play("..\..\drumpad-crash.wav", _
AudioPlayMode.WaitToComplete)
My.Computer.Audio.Play("..\..\drumpad-bass_drum.wav")
End Sub
```

Подчеркнём следующее. Чтобы все игровые действия прекратились на время исполнения мелодии, мы должны в методе Play в качестве второго параметра использовать уже применён-

ную выше константу WaitToComplete из перечисления режимов AudioPlayMode, как показано в следующем коде.

Листинг 6.3. Метод для загрузки и воспроизведения звуковых файлов.

```
Private Sub Form1_Load(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles MyBase.Load
My.Computer.Audio.Play("..\..\drumpad-crash.wav", _
AudioPlayMode.WaitToComplete)
End Sub
```

Чтобы мы услышали непрерывное циклическое (Loop) воспроизведение звукового файла, мы должны в методе Play в качестве второго параметра использовать константу BackgroundLoop из перечисления режимов AudioPlayMode, как показано в следующем коде.

Листинг 6.4. Метод для загрузки и воспроизведения звуковых файлов.

```
Private Sub Form1_Load(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles MyBase.Load
My.Computer.Audio.Play("..\..\drumpad-crash.wav", _
AudioPlayMode.BackgroundLoop)
End Sub
```

Чтобы остановить (Stop) воспроизведение звука, необходимо в каком либо методе, например, в обработчике щелчка кнопки записать строку:

```
My.Computer.Audio.Stop()
```

Теперь после щелчка кнопки звучание прекратится.

Если в игре применяются несколько звуковых файлов, то их целесообразно разместить в одной папке с именем, например, Sounds. Для добавления в проект этой папки, в панели Solution Explorer (рис. 6.3) выполняем правый щелчок по имени проекта, в контекстном меню выбираем Add, New Folder, в поле появившегося значка папки записываем имя папки и нажимаем клавишу Enter.

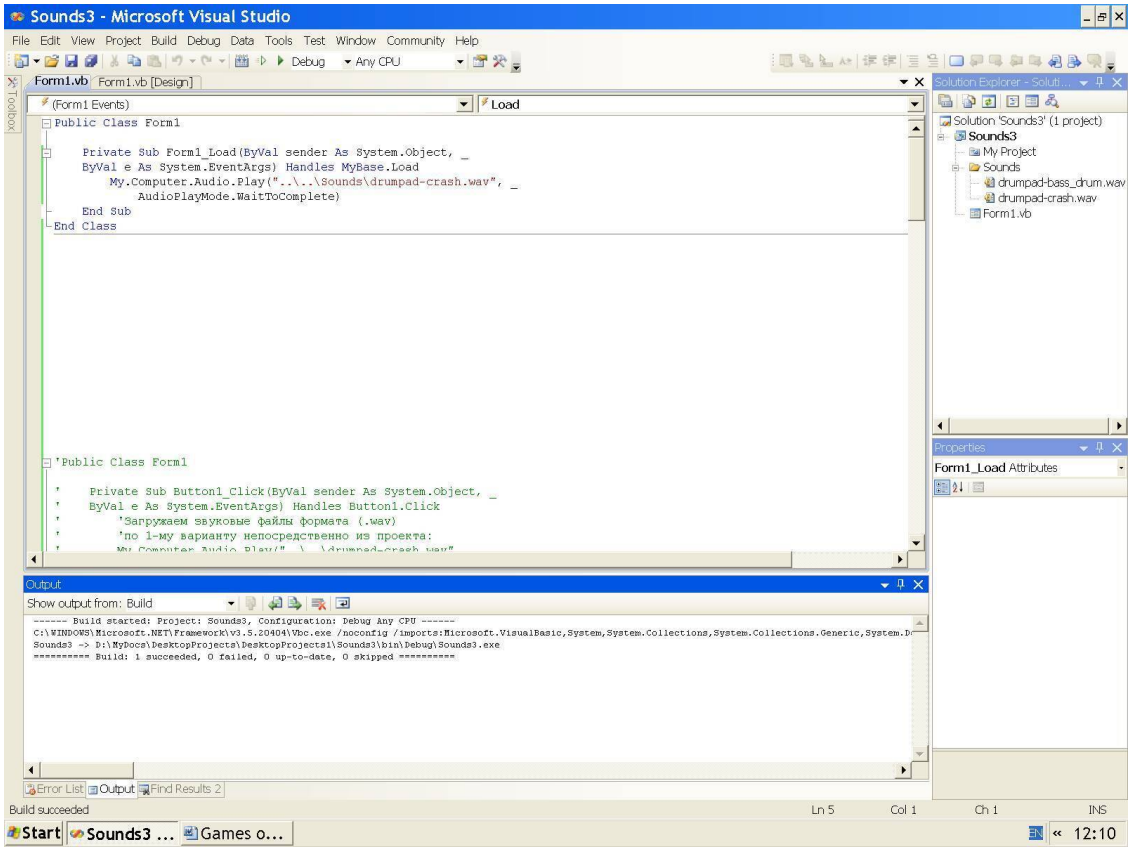


Рис. 6.3. Папка Sounds в панели Solution Explorer.

Добавляем в эту папку (например, из Интернета) первый звуковой файл по стандартной схеме, а именно: выполняем правый щелчок по имени этой папки, в контекстном меню выбираем Add, Existing Item, в панели Add Existing Item в окне “Files of type” выбираем “All Files”, в центральном окне находим и выделяем имя файла и щёлкаем кнопку Add (или дважды щёлкаем по имени файла). В панели Solution Explorer мы увидим этот файл.

Аналогично добавляем в папку Sounds нашего проекта остальные звуковые файлы.

Напомним, что добавлять в проект все файлы можно как по одному, так и все сразу (после их выделения или только одной мышью, или мышью с нажатой клавишей Shift – для выделения всех соседних файлов, или мышью с нажатой клавишей Ctrl – для выделения всех файлов в различных местах).

Теперь, чтобы мы услышали воспроизведение звукового файла из папки Sound нашего проекта, необходимо применить следующий код.

Листинг 6.5. Метод для загрузки и воспроизведения звуковых файлов.

```
Private Sub Form1_Load(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles MyBase.Load
My.Computer.Audio.Play("..\..\Sounds\drumpad-crash.wav", _
AudioPlayMode.WaitToComplete)
End Sub
```

Однако чаще требуется записывать строку типа:

```
My.Computer.Audio.Play("..\..\drumpad-crash.wav")
```

не в методе Form1_Load, а в других методах, которые управляют игровыми объектами.

Для примера, покажем, как управлять звуками при помощи двух кнопок Button, которые мы разместили на форме. В режиме проектирования дважды щёлкаем первую кнопку (или в панели Properties для этой выделенной щелчком кнопки на вкладке Events дважды щёлкаем по

имени события Click). Появляется файл Form1.vb с шаблоном (обработчика щелчка по этой кнопке), который после записи нашего кода принимает вид следующего метода.

Листинг 6.6. Метод для кнопки.

```
Private Sub Button1_Click(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles Button1.Click
'Загружаем звуковые файлы формата (.wav)
'по 1-му варианту непосредственно из проекта:
My.Computer.Audio.Play("..\..\drumpad-crash.wav", _
AudioPlayMode.BackgroundLoop)

'Загружаем звуковые файлы формата (.wav)
'по 2-му варианту из внешней папки:
'My.Computer.Audio.Play( _
'"D:\MyDocs\Sounds\drumpad-bass_drum.wav")
End Sub
```

В режиме проектирования дважды щёлкаем вторую кнопку (или в панели Properties для этой выделенной щелчком кнопки на вкладке Events дважды щёлкаем по имени события Click). Появляется файл Form1.vb с шаблоном (обработчика щелчка по этой кнопке), который после записи нашего кода принимает вид следующего метода.

Листинг 6.7. Метод для кнопки.

```
Private Sub Button2_Click(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles Button2.Click
'Загружаем звуковые файлы формата (.wav)
'по 1-му варианту непосредственно из проекта:
My.Computer.Audio.Play("..\..\drumpad-bass_drum.wav")
'Загружаем звуковые файлы формата (.wav)
'по 2-му варианту из внешней папки:
'My.Computer.Audio.Play( _
'"D:\MyDocs\Sounds\drumpad-crash.wav")
End Sub
```

В режиме выполнения (Build, Build Selection; Debug, Start Without Debugging), нажимая кнопки, мы услышим соответствующее (одноразовое или циклическое) воспроизведение звуковых файлов, которые мы добавили в проект. А именно, после щелчка первой кнопки и выполнения кода:

```
My.Computer.Audio.Play("..\..\drumpad-crash.wav", _
AudioPlayMode.BackgroundLoop)
```

мы услышим непрерывное циклическое (Loop) воспроизведение звукового файла drumpad-crash.wav типа шуршания оркестровых металлических тарелок после воздействия на них металлической метелкой. А после каждого нашего щелчка второй кнопки и выполнения кода:

```
My.Computer.Audio.Play("..\..\drumpad-bass_drum.wav")
```

мы будем слышать одиночное воспроизведение звукового файла drumpad-bass_drum.wav типа удара по барабану. Нажимая вторую кнопку с различной частотой, мы будем импровизировать и создавать различную мелодию из добавленных в проект двух звуковых файлов.

Аналогично по этой методике, добавляя в проект большое количество звуковых файлов, добавляя на форму большое количество кнопок Button (или других элементов управления) и записывая в методы-обработчики щелчков по этим кнопкам различные варианты кода для воспроизведения этих звуковых файлов в различной последовательности и с различным нало-

жением звуков, в режиме выполнения приложения, щёлкая по кнопкам, мы будем получать множество самых разнообразных мелодий.

Отметим, что для управления воспроизведением звуковых файлов в дополнение или вместо кнопок Button (чтобы не загромождать форму Form1) можно использовать и клавиши клавиатуры, и компоненты с панели инструментов Toolbox по описанной далее методике.

6.3. Методика приостановки и возобновления звуков на основе пространства имён My

Для приостановки и возобновления воспроизведения звуковых файлов во время выполнения приложения можно разработать много вариантов кода, например, воспользоваться каким-либо элементом управления или компонентом. Так как далее для задания режимов почти всех игр мы будем применять выпадающее меню типа MenuStrip, то для решения поставленной здесь задачи применим это меню. С панели инструментов Toolbox переносим на форму элемент управления MenuStrip и щёлкаем по нему (ниже формы в режиме проектирования). На форме Form1 появляются окна с надписью Type Here (Печатайте здесь), в которые записываем команды на русском языке (по второму варианту, можно записывать в панели Properties в свойстве Text), для примера, для управления двумя звуковыми файлами: Звуки, Звук 1, Звук 2, рис. 6.4. Теперь в панели Properties в свойстве Name изменяем эти русские команды на соответствующие английские: Sounds, Sound1, Sound2. Рассмотрим первую команду Звук 1.

Щёлкаем по этой команде “Звук 1” и в панели Properties (для этой команды) значение свойства Checked задаём как True (рис. 6.5), чтобы на форме слева от этой команды появился флажок. А чтобы этот флажок можно было удалить и снова установить (после щелчка по команде мышью), в панели Properties для этой команды значение свойства CheckOnClick также задаём как True.

Дважды щёлкаем по этой команде “Звук 1” (в режиме проектирования). Появляется файл Form1.vb с автоматически сгенерированным шаблоном метода, выше которого объявляем булеву переменную, а в шаблон записываем код, как показано на следующем листинге.

Листинг 6.8. Код для приостановки и возобновления звука.

```
'Объявляем логическую переменную OffOn и задаём ей True:
Dim OffOn As Boolean = True
Private Sub Sound1ToolStripMenuItem_Click( _
ByVal sender As System.Object, ByVal e As System.EventArgs) _
Handles Sound1ToolStripMenuItem.Click
'Изменяем значение на противоположное:
OffOn = Not OffOn
'Выключаем Stop и включаем Play звук 1:
If (OffOn = False) Then
My.Computer.Audio.Stop()
Else
My.Computer.Audio.Play("..\..\drumpad-crash.wav", _
AudioPlayMode.BackgroundLoop)
End If
End Sub
```

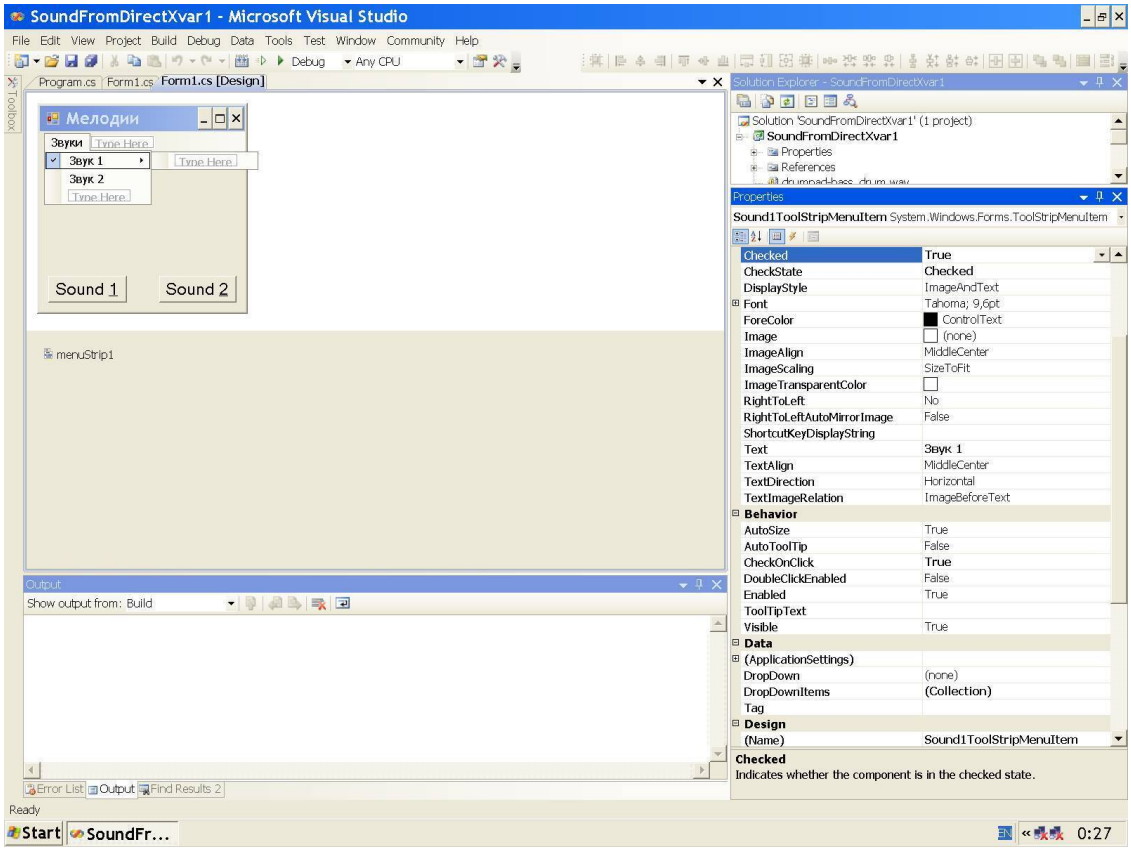


Рис. 6.4. Команды MenuStrip.

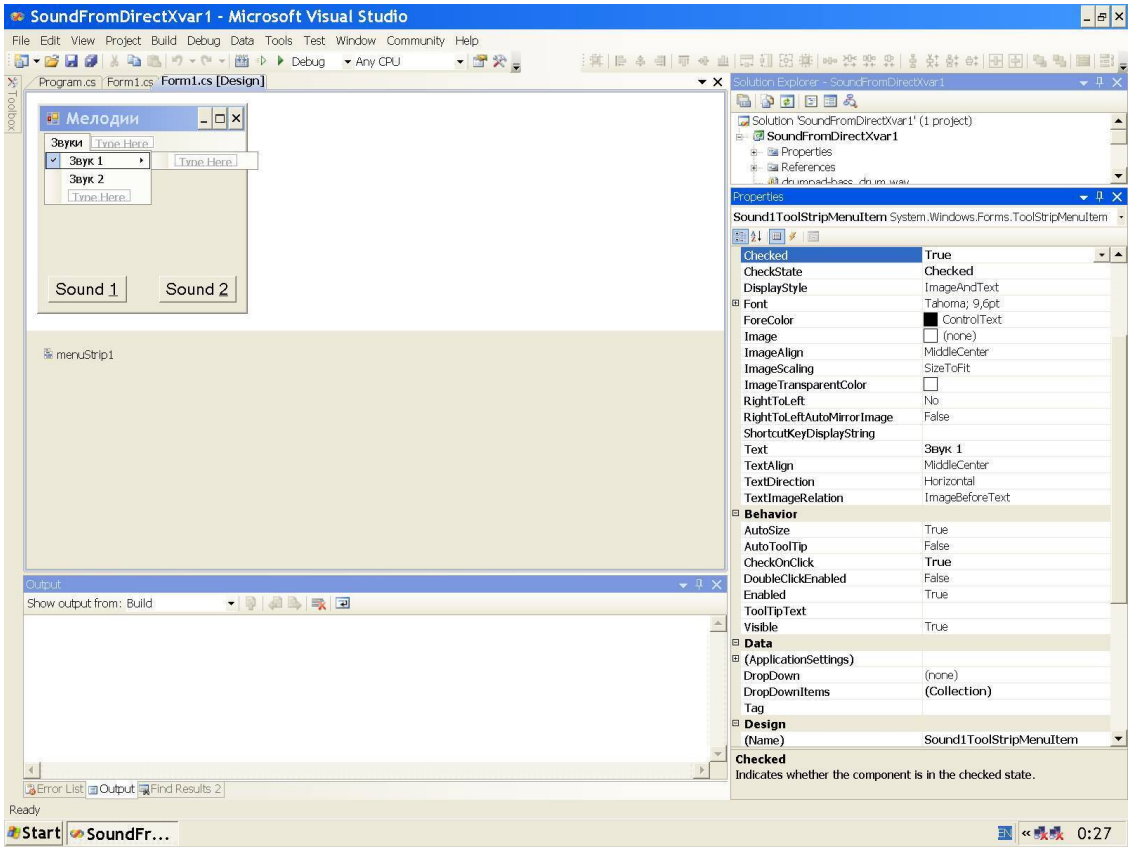


Рис. 6.5. Задаём свойства команды.

Теперь в режиме выполнения (Build, Build Selection; Debug, Start Without Debugging), поочерёдно удаляя или устанавливая мышью флажок напротив этой команды “Звук 1”, мы будем выключать методом Stop и включать методом Play циклическое (Loop) непрерывное воспроизведение звукового файла, который мы добавили в проект.

Если после каждой установки флажка нам нужно воспроизводить звуковой файл только один раз, то вместо строки:

```
My.Computer.Audio.Play("..\..\drumpad-crash.wav", _
AudioPlayMode.BackgroundLoop)
```

записываем:

```
My.Computer.Audio.Play("..\..\drumpad-crash.wav")
```

Аналогично можно использовать команду “Звук 2” для управления вторым звуковым файлом.

Аналогично по этой методике мы можем добавить в проект много звуковых файлов, а в меню MenuStrip – много команд для приостановки и возобновления звукового сопровождения разнообразных игр (в режиме выполнения).

6.4. Методика воспроизведения звуковых файлов на основе встроенного ресурса

Недостатком предыдущей методики на основе пространства имён My является невозможность её использования для мобильных устройств (карманных компьютеров, коммуникаторов, смартфонов, мобильных телефонов и т.п.) с операционной системой Windows Mobile.

И среда выполнения .NET Framework для настольных компьютеров, и среда выполнения .NET Compact Framework для мобильных устройств позволяет записать неуправляемый код для воспроизведения звуковых файлов по методике встроенного ресурса (Embedded Resource) с использованием платформы Platform Invoke (сокращенно P/Invoke) и динамически подключаемой библиотеки (dynamic link library) winmm.dll (для настольного компьютера) или CoreDll.dll (для мобильного устройства) из более общей библиотеки Windows API (Application Programming Interfaces).

Данная универсальная методика на основе встроенного ресурса удобна при переносе какого-либо приложения (со звуковым сопровождением) с настольного компьютера на мобильное устройство и наоборот.

В данном параграфе мы разработаем универсальную методику воспроизведения звуковых файлов на основе нового файла Sound.vb, в котором имеется одноименный класс Sound.

Мы назвали эту методику универсальной потому, что она может быть применена во многих самых разнообразных играх, и далее будет нами применена в некоторых типичных играх.

В приложении класс Sound должен обеспечить:

выключение звука, если пользователь не желает его слушать;

непрерывный циклический звук;

одиночный звук.

Для каждого звука мы создаём объект класса Sound. Есть также несколько перегруженных конструкторов этого класса, чтобы мы могли создать объект при использовании внутреннего ресурса в виде пути к звуковому файлу filename или потока (Stream), как показано в следующем коде:

```
'Метод для считывания звукового файла:
Private Sub readStream(ByVal soundStream As Stream)
'Создаём массив soundBytes с элементами типа Byte
'и инициализируем размерность этого массива
```

```
'как длину звукового файла soundStream.Length:
soundBytes = New Byte(soundStream.Length) {}
'Из потока soundStream считываем звуковой файл
'в массив soundBytes:
soundStream.Read(soundBytes, 0, soundStream.Length)
End Sub
'Объявляем метод-конструктор класса Sound
'с параметром в виде потока soundStream класса Stream:
Public Sub New(ByVal soundStream As Stream)
'Вызываем метод для считывания звукового файла:
readStream(soundStream)
End Sub
'Объявляем метод-конструктор класса Sound
'с параметром в виде пути filename к звуковому файлу:
Public Sub New(ByVal filename As String)
Dim execAssem As System.Reflection.Assembly = _
System.Reflection.Assembly.GetExecutingAssembly()
Dim soundStream As Stream = _
execAssem.GetManifestResourceStream(filename)
If (soundStream Is Nothing) Then
System.Windows.Forms.MessageBox.Show( _
"Missing file : " + filename, "Audio Load")
Return
End If
readStream(soundStream)
End Sub
```

Метод readStream выполняет фактическую загрузку звука. Операционная система управляет звуком. Класс Sound использует платформу Platform Invoke (P/Invoke), чтобы при помощи этого метода вызвать и управлять звуком, как показано в следующем коде:

```
Public Enum Flags
SND_ALIAS = &H10000
SND_ALIAS_ID = &H110000
SND_FILENAME = &H20000
SND_RESOURCE = &H40004
SND_SYNC = &H0
SND_ASYNC = &H1
SND_NODEFAULT = &H2
SND_MEMORY = &H4
SND_LOOP = &H8
SND_NOSTOP = &H10
SND_NOWAIT = &H2000
SND_VALIDFLAGS = &H17201F
SND_RESERVED = &HFF000000
SND_TYPE_MASK = &H170007
End Enum
Private Declare Function PlaySound _
Lib "winmm.dll" Alias "PlaySound" (ByVal szSound() As Byte, _
ByVal hModule As IntPtr, ByVal dwFlags As Integer) As Integer
```

В этом коде ряд флажков (Flags) непосредственно управляет генерацией звука. Класс Sound содержит статический член, который в настоящее время имеет ссылку на непрерывный циклический звук (или любой другой). Если звук был выключен и затем снова включён, циклический звук возобновляется. Класс Sound содержит методы Play и PlayLoop для одиночного и циклического воспроизведения звукового файла, как показано в следующем коде:

```
'Метод для разового воспроизведения звукового файла:
Public Sub Play()
loopSound = Nothing
If (Sound.Enabled) Then
PlaySound(soundBytes, IntPtr.Zero, _
Fix(Flags.SND_ASYNC Or _
Flags.SND_MEMORY))
End If
End Sub
'Метод для циклического воспроизведения звукового файла:
Public Sub PlayLoop()
loopSound = soundBytes
If (Sound.Enabled) Then
PlaySound(soundBytes, IntPtr.Zero, _
Fix(Flags.SND_ASYNC Or _
Flags.SND_MEMORY Or _
Flags.SND_LOOP))
End If
End Sub
```

Метод StopSound в классе Sound останавливает проигрывающийся звук, задавая массиву soundBytes нулевое значение (Zero) следующим образом:

```
Public Sub StopSound()
If (Not loopSound Is Nothing) Then
PlaySound(Nothing, IntPtr.Zero, _
0)
End If
End Sub
```

Метод ResumeSound возобновляет звук только тогда, если до этого воспроизводился циклический звук, как показано в следующем коде:

```
Public Sub ResumeSound()
If (Not Sound.Enabled) Then
Return
End If
If (Not loopSound Is Nothing) Then
PlaySound( _
loopSound, _
IntPtr.Zero, _
Flags.SND_ASYNC Or Flags.SND_MEMORY Or _
Flags.SND_LOOP)
End If
End Sub
```

Теперь от описания приступим к конкретной реализации.

А именно, для воспроизведения звуковых файлов, которые добавлены в проект по описанной ранее схеме, при помощи универсального (для многих приложений и игр) файла

Sound.vb, в панели Solution Explorer выполняем правый щелчок по имени проекта и в контекстном меню выбираем Add, New Item. В панели Add New Item выделяем шаблон Code File, в окне Name записываем имя Sound.vb и щёлкаем кнопку Add.

В проект (и в панель Solution Explorer) добавляется этот файл, открывается пустое окно редактирования кода, в которое записываем следующий код.

Листинг 6.9. Файл Sound.vb.

```
Imports System.IO 'Для класса Stream.
Public Class Sound
'Переменная включает и выключает звук:
Public Shared Enabled As Boolean = True
'Свойство для приостановки и возобновления звука:
Public Property EnabledProperty() As Boolean
Get
Return Enabled
End Get
Set(ByVal value As Boolean)
Enabled = value
If (value) Then
ResumeSound()
Else
StopSound()
End If
End Set
End Property
'Массив для циклического звучания:
Private loopSound() As Byte = Nothing
'Массив для одноразового воспроизведения звука:
Private soundBytes() As Byte

'Флажки (константы) для управления методами
'операционной системы Windows CE, а именно,
'методом PlaySound из библиотеки CoreDll.dll
'при воспроизведении звукового файла:
Public Enum Flags
SND_ALIAS = &H10000
SND_ALIAS_ID = &H110000
SND_FILENAME = &H20000
SND_RESOURCE = &H40004
SND_SYNC = &H0
SND_ASYNC = &H1
SND_NODEFAULT = &H2
SND_MEMORY = &H4
SND_LOOP = &H8
SND_NOSTOP = &H10
SND_NOWAIT = &H2000
SND_VALIDFLAGS = &H17201F
SND_RESERVED = &HFF000000
SND_TYPE_MASK = &H170007
End Enum
```

```
'Из пространства имён System.Runtime.InteropServices
'импортируем библиотеку CoreDll.dll и
'объявляем функцию PlaySound этой библиотеки:
Private Declare Function PlaySound _
Lib "winmm.dll" Alias "PlaySound" (ByVal szSound() As Byte, _
ByVal hModule As IntPtr, ByVal dwFlags As Integer) As Integer
'Метод для считывания звукового файла:
Private Sub readStream(ByVal soundStream As Stream)
'Создаём массив soundBytes с элементами типа Byte
'и инициализируем размерность этого массива
'как длину звукового файла soundStream.Length:
soundBytes = New Byte(soundStream.Length) {}
'Из потока soundStream считываем звуковой файл
'в массив soundBytes:
soundStream.Read(soundBytes, 0, soundStream.Length)
End Sub
'Объявляем метод-конструктор класса Sound
'с параметром в виде потока soundStream класса Stream:
Public Sub New(ByVal soundStream As Stream)
'Вызываем метод для считывания звукового файла:
readStream(soundStream)
End Sub
'Объявляем метод-конструктор класса Sound
'с параметром в виде пути filename к звуковому файлу:
Public Sub New(ByVal filename As String)
Dim execAssem As System.Reflection.Assembly = _
System.Reflection.Assembly.GetExecutingAssembly()
Dim soundStream As Stream = _
execAssem.GetManifestResourceStream(filename)
If (soundStream Is Nothing) Then
System.Windows.Forms.MessageBox.Show( _
"Missing file : " + filename, "Audio Load")
Return
End If
readStream(soundStream)
End Sub
'Метод для разового воспроизведения звукового файла:
Public Sub Play()
loopSound = Nothing
If (Sound.Enabled) Then
PlaySound(soundBytes, IntPtr.Zero, _
Fix(Flags.SND_ASYNC Or _
Flags.SND_MEMORY))
End If
End Sub
'Метод для циклического воспроизведения звукового файла:
Public Sub PlayLoop()
loopSound = soundBytes
If (Sound.Enabled) Then
```

```
PlaySound(soundBytes, IntPtr.Zero, _
Fix(Flags.SND_ASYNC Or _
Flags.SND_MEMORY Or _
Flags.SND_LOOP))
End If
End Sub
'Метод остановки воспроизведения звукового файла:
Public Sub StopSound()
If (Not loopSound Is Nothing) Then
PlaySound(Nothing, IntPtr.Zero, _
0)
End If
End Sub
'Метод для восстановления воспроизведения звукового файла:
Public Sub ResumeSound()
If (Not Sound.Enabled) Then
Return
End If
If (Not loopSound Is Nothing) Then
PlaySound( _
loopSound, _
IntPtr.Zero, _
Flags.SND_ASYNC Or Flags.SND_MEMORY Or _
Flags.SND_LOOP)
End If
End Sub
End Class
```

Этот файл Sound.vb можно использовать во многих приложениях и играх для воспроизведения звуковых файлов, добавляя его в проект по стандартной схеме Project, Add Existing Item, как мы сейчас покажем на конкретном примере нового проекта.

Для создания проекта в VS щёлкаем кнопку New Project (или File, New, Project). В панели New Project в окне Project Types выбираем тип проекта Visual Basic, Windows, в окне Templates выделяем шаблон Windows Forms Application, в окне Name записываем любое имя проекта, например, Sounds4 и щёлкаем OK. Создаётся проект, появляется форма Form1 (рис. 6.6) в режиме проектирования.

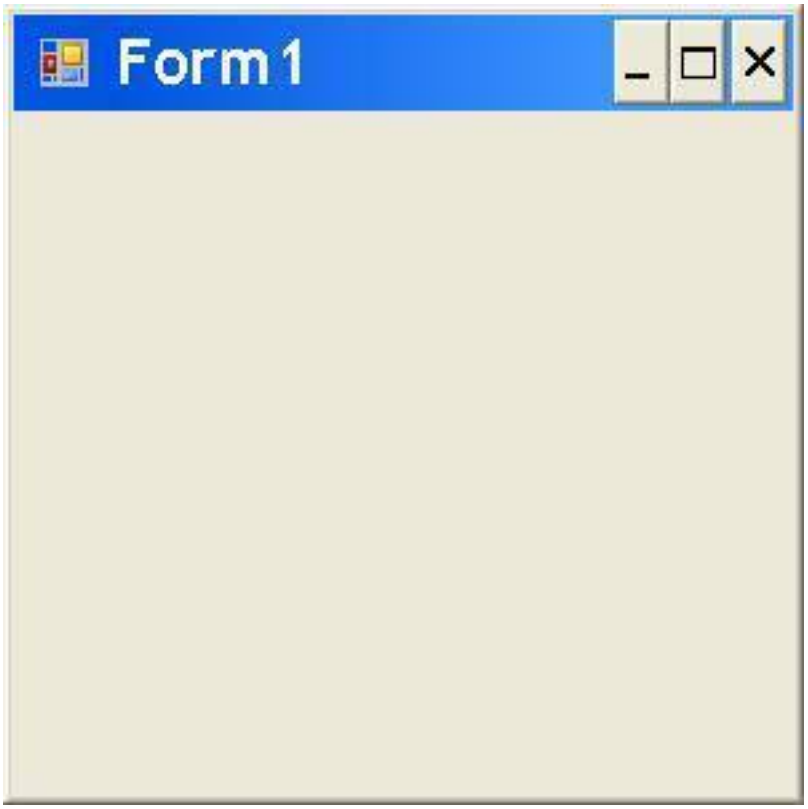


Рис. 6.6. Форма Form1 в режиме выполнения.

Проектируем (или оставляем по умолчанию) эту форму, как описано в параграфе “Методика проектирования формы”.

Для добавления звукового файла move.wav в проект, в меню Project выбираем Add Existing Item, в панели Add Existing Item в окне Files of type устанавливаем All Files, в окне "Look in" находим (например, в папке с загруженным из Интернета файлом) файл и щёлкаем кнопку Add (или дважды щёлкаем по имени файла). Этот файл мы увидим в панели Solution Explorer.

В панели Solution Explorer выделяем имя этого файла, а в панели Properties (для данного файла) в свойстве Build Action (Действие при построении) вместо заданного по умолчанию выбираем значение Embedded Resource (Встроенный ресурс).

Аналогично добавляем и встраиваем в проект второй файл win.wav типа мажорной мелодии.

Аналогично можно добавить и встроить в проект ещё много звуковых файлов, которые мы желаем послушать во время выполнения приложения или игры.

Открываем файл Form1.vb (например, так: File, Open, File) и вверху записываем директиву для подключения требуемого пространства имен:

Imports System.Reflection 'Для класса Assembly.

Напомним, что эту строку можно и не записывать, но тогда нам придётся перед каждым классом записывать эти пространства имён System.Reflection.

Теперь в панели Properties (для формы Form1) на вкладке Events дважды щёлкаем по имени события Load (Загрузка).

Появляется файл Form1.vb с шаблоном метода Form1_Load, который после записи нашего кода принимает следующий вид.

Листинг 6.10. Метод для загрузки и воспроизведения звуковых файлов.

```
'Объявляем объекты класса Sound для каждого звукового файла:
Dim moveSound As Sound
```

```
Dim winSound As Sound
'Создаём объект myAssembly класса System.Reflection.Assembly
'и присваиваем ему ссылку на исполняемую сборку приложения:
Dim myAssembly As Assembly = Assembly.GetExecutingAssembly()
'Создаём объект myAssemblyName
'класса System.Reflection.AssemblyName и присваиваем ему
'имя сборки, которое состоит из имени проекта,
'Version, Culture, PublicKeyToken:
Dim myAssemblyName As AssemblyName = myAssembly.GetName()
'Из имени сборки при помощи свойства Name
'выделяем имя проекта myName_of_project типа String:
Dim myName_of_project As String = myAssemblyName.Name
Private Sub Form1_Load(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles MyBase.Load
'Загружаем звуковые файлы:
'Вариант 1:
'moveSound = _
' New Sound(myAssembly.GetManifestResourceStream( _
' myName_of_project + "." + "move.wav"))
winSound = _
New Sound(myAssembly.GetManifestResourceStream( _
myName_of_project + "." + "win.wav"))
'Загружаем звуковые файлы:
'Вариант 2:
'moveSound = New Sound(myName_of_project + "." + "move.wav")
'winSound = New Sound(myName_of_project + "." + "win.wav")
'Воспроизведение звукового файла:
'moveSound.Play()
'Воспроизведение звукового файла:
winSound.Play()
End Sub
```

В режиме выполнения (Build, Build Selection; Debug, Start Without Debugging) мы услышим соответствующее (одноразовое) воспроизведение звукового файла, который мы добавили непосредственно в проект (а не в дополнительную папку Sounds проекта).

Теперь строки типа:

```
moveSound.Play()
winSound.Play()
```

мы можем записывать в тех местах любой программы, где необходимо воспроизводить соответствующий звуковой файл.

Для циклического (Loop) непрерывного воспроизведения звукового файла вместо строки:

```
winSound.Play()
```

следует записать:

```
winSound.PlayLoop()
```

Далее можно экспериментировать, как описано выше в первой методике.

Важно отметить, если в игре применяются несколько звуковых файлов, то их целесообразно разместить в одной папке с именем, например, Sounds. Для добавления в проект этой папки, в панели Solution Explorer (рис. 6.7) выполняем правый щелчок по имени проекта, в

контекстном меню выбираем Add, New Folder, в поле появившегося значка папки записываем имя папки и нажимаем клавишу Enter.

Добавляем в эту папку (например, из Интернета) первый звуковой файл по стандартной схеме, а именно: выполняем правый щелчок по имени этой папки, в контекстном меню выбираем Add, Existing Item, в панели Add Existing Item в окне “Files of type” выбираем “All Files”, в центральном окне находим и выделяем имя файла и щёлкаем кнопку Add (или дважды щёлкаем по имени файла). В панели Solution Explorer мы увидим этот файл.

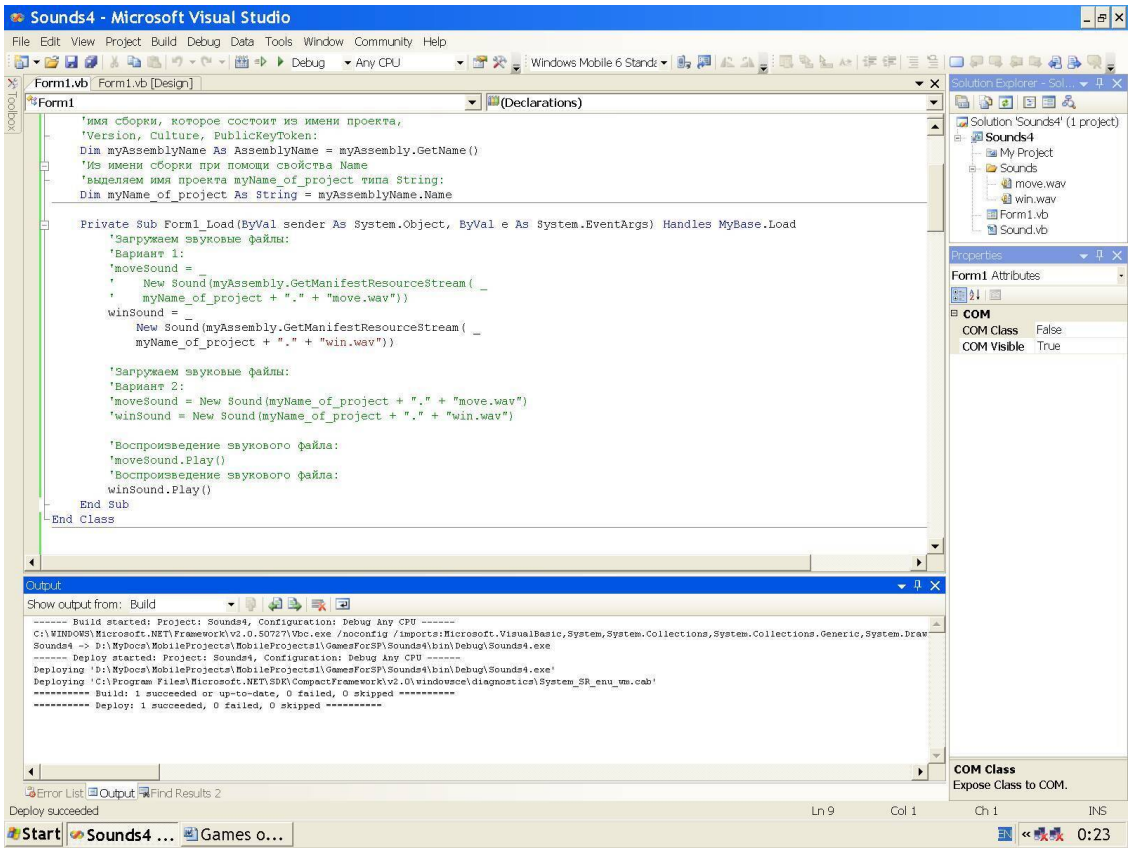


Рис. 6.7. Папка Sounds в панели Solution Explorer.

В панели Solution Explorer выделяем имя этого файла, а в панели Properties (для данного файла) в свойстве Build Action (Действие при построении) вместо заданного по умолчанию выбираем значение Embedded Resource (Встроенный ресурс).

Аналогично добавляем в нашу новую папку Sounds данного проекта и встраиваем в виде ресурса остальные звуковые файлы.

Один и тот же звуковой файл мы не можем добавить как непосредственно в проект, так и в папку Sounds (надо выбрать что-то одно, иначе будет ошибка). Вследствие этого весь предыдущий код для воспроизведения звуковых файлов остаётся тем же независимо от того, звуковой файл добавлен непосредственно в проект, или добавлен в наше новую папку Sounds.

Таким образом, по этой методике, добавляя в проект большое количество звуковых файлов, добавляя на форму большое количество кнопок Button (или других элементов управления) и записывая в методы-обработчики щелчков по этим кнопкам различные варианты кода для воспроизведения этих звуковых файлов в различной последовательности и с различным наложением звуков, в режиме выполнения приложения, щёлкая по кнопкам, мы будем получать множество самых разнообразных мелодий.

Отметим, что для управления воспроизведением звуковых файлов в дополнение или вместо кнопок Button (чтобы не загромождать форму Form1) можно использовать и клавиши клавиатуры, и компоненты с панели инструментов Toolbox по описанной ранее методике.

6.5. Методика приостановки и возобновления звуков на основе встроенного ресурса

Для приостановки и возобновления воспроизведения звуковых файлов во время выполнения приложения можно разработать много вариантов кода, например, воспользоваться каким-либо элементом управления или компонентом.

Так как далее для задания режимов почти всех игр мы будем применять выпадающее меню типа MenuStrip, то для решения поставленной здесь задачи применим это меню. С панели инструментов Toolbox переносим на форму элемент управления MenuStrip и щёлкаем по нему (ниже формы в режиме проектирования). На форме Form1 появляются окна с надписью Type Here (Печатайте здесь), в которые записываем команды на русском языке (по второму варианту, можно записывать в панели Properties в свойстве Text), для примера, для управления двумя звуковыми файлами: Звуки, Звук 1, Звук 2, рис. 6.8. Теперь в панели Properties в свойстве Name изменяем эти русские команды на соответствующие английские: Sounds, Sound1, Sound2. Рассмотрим первую команду Звук 1.

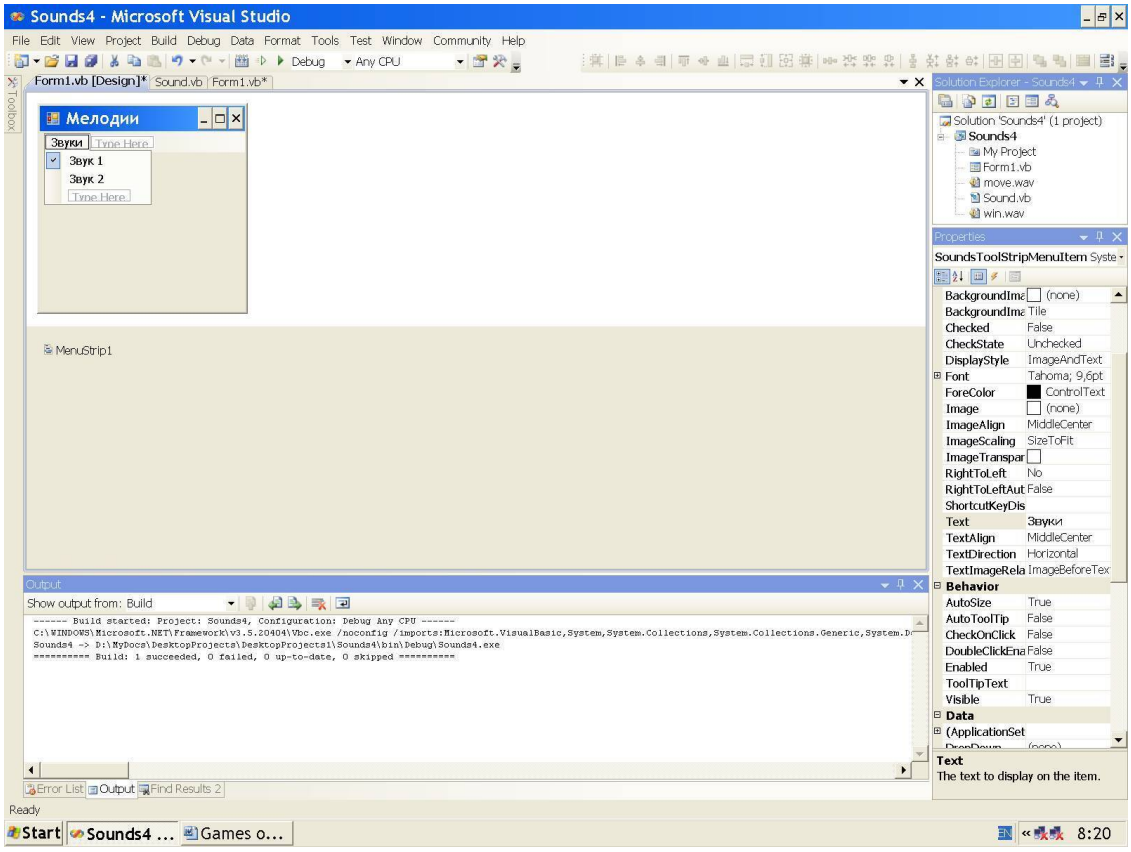


Рис. 6.8. Команды MenuStrip.

Щёлкаем по этой команде “Звук 1” и в панели Properties (для этой команды) значение свойства Checked задаём как True, чтобы на форме слева от этой команды появился флажок. А чтобы этот флажок можно было удалить и снова установить (после щелчка по команде мышью), в панели Properties для этой команды значение свойства CheckOnClick также задаём как True.

Дважды щёлкаем по этой команде “Звук 1” (в режиме проектирования). Появляется файл Form1.vb с автоматически сгенерированным шаблоном метода, выше которого объявляем булеву переменную, а в шаблон записываем код, как показано на следующем листинге.

Листинг 6.11. Код для приостановки и возобновления звука.

```
'Объявляем логическую переменную OffOn и задаём ей True:
Dim OffOn As Boolean = True
Private Sub Sound1ToolStripMenuItem_Click( _
ByVal sender As System.Object, ByVal e As System.EventArgs) _
Handles Sound1ToolStripMenuItem.Click
'Изменяем значение на противоположное:
OffOn = Not OffOn
'Выключаем Stop и включаем Play звук 1:
If (OffOn = False) Then
winSound.StopSound()
Else
winSound.PlayLoop()
End If
End Sub
```

Теперь в режиме выполнения (Build, Build Selection; Debug, Start Without Debugging), поочерёдно удаляя или устанавливая мышью флажок напротив этой команды “Звук 1”, мы будем выключать методом StopSound и включать методом PlayLoop циклическое (Loop) непрерывное воспроизведение звукового файла, который мы добавили в проект.

Если после каждой установки флажка нам нужно воспроизводить звуковой файл только один раз, то вместо строки:

```
winSound.PlayLoop()
```

записываем (как уже было показано выше):

```
winSound.Play()
```

Аналогично можно использовать команду “Звук 2” для управления вторым звуковым файлом. Аналогично по этой методике мы можем добавить в проект много звуковых файлов, а в меню MenuStrip – много команд для приостановки и возобновления звукового сопровождения разнообразных игр (в режиме выполнения).

6.6. Вариант воспроизведения звуковых файлов на основе встроенного ресурса

Учитывая важность методики воспроизведения звуковых файлов на основе встроенного ресурса, которую можно применять в приложениях и для настольного компьютера, и для мобильного устройства, приведём второй вариант этой методики.

Основное отличие данного варианта от предыдущего заключается в том, что вместо объявления объектов класса Sound для каждого звукового файла (по первому варианту):

```
Dim moveSound As Sound
Dim winSound As Sound
```

мы объявляем динамические массивы (array) типа Byte для каждого звукового файла (по второму варианту):

```
Dim array_btSoundBounce() As Byte
Dim array_btSoundExplode() As Byte
```

Остальные пояснения будут понятны из нового проекта.

Для создания проекта в VS щёлкаем кнопку New Project (или File, New, Project). В панели New Project в окне Project Types выбираем тип проекта Visual Basic, Windows, в окне Templates

выделяем шаблон Windows Forms Application, в окне Name записываем любое имя проекта, например, Sounds6 и щёлкаем OK. Создаётся проект, появляется форма Form1 (рис. 6.9) в режиме проектирования.

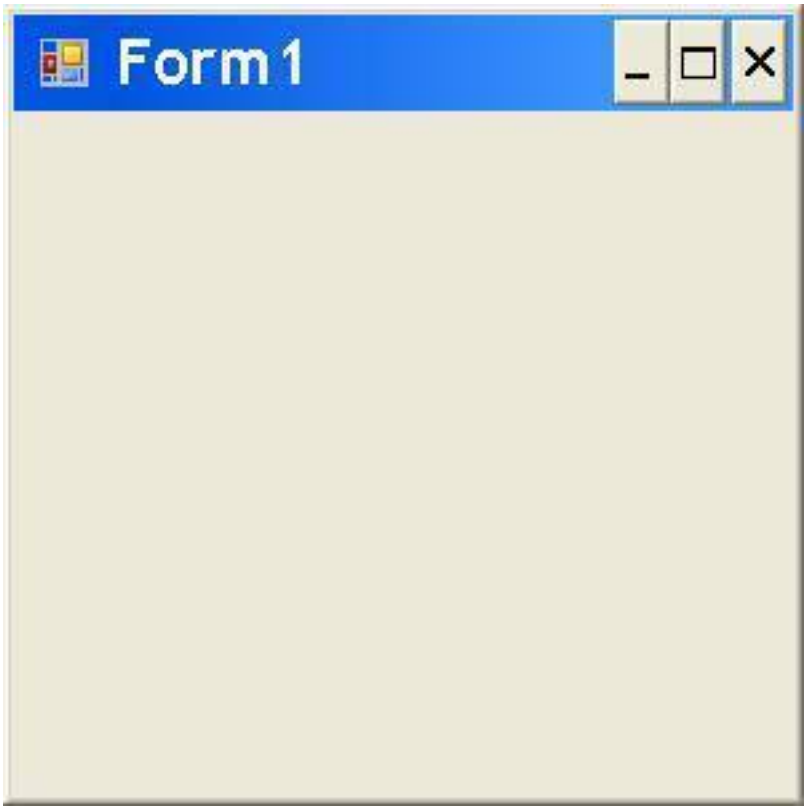


Рис. 6.9. Форма Form1 в режиме выполнения.

Проектируем (или оставляем по умолчанию) эту форму, как описано в параграфе “Методика проектирования формы”.

Для добавления звукового файла bounce.wav (типа удара) в проект, в меню Project выбираем Add Existing Item, в панели Add Existing Item в окне Files of type устанавливаем All Files, в окне "Look in" находим (например, в папке с загруженным из Интернета файлом) файл и щёлкаем кнопку Add (или дважды щёлкаем по имени файла). Этот файл мы увидим в панели Solution Explorer.

В панели Solution Explorer выделяем имя этого файла, а в панели Properties (для данного файла) в свойстве Build Action (Действие при построении) вместо заданного по умолчанию выбираем значение Embedded Resource (Встроенный ресурс).

Аналогично добавляем и встраиваем в проект второй файл explode.wav типа взрыва. Аналогично можно добавить и встроить в проект ещё много звуковых файлов, которые мы желаем послушать во время выполнения приложения или игры.

Открываем файл Form1.vb (например, так: File, Open, File) и вверху записываем директиву для подключения требуемого пространства имен:

Imports System.Reflection 'Для класса Assembly.

Напомним, что эту строку можно и не записывать, но тогда нам придётся перед каждым классом записывать эти пространства имён System.Reflection.

Теперь в панели Properties (для формы Form1) на вкладке Events дважды щёлкаем по имени события Load (Загрузка).

Появляется файл Form1.vb с шаблоном метода Form1_Load, который после записи нашего кода принимает следующий вид.

Листинг 6.12. Метод для загрузки и воспроизведения звуковых файлов.

```
'Объявляем динамические массивы array типа Byte
'для каждого звукового файла:
Dim array_btSoundBounce() As Byte
Dim array_btSoundExplode() As Byte
'Загружаем в проект файлы изображений и звуков по такой схеме:
'Создаём объект myAssembly класса Assembly и присваиваем ему
'ссылку на исполняемую сборку нашего приложения:
Dim myAssembly As Assembly = Assembly.GetExecutingAssembly()
'Создаём объект myAssemblyName
'класса System.Reflection.AssemblyName и присваиваем ему
'имя сборки, которое состоит из имени проекта,
'Version, Culture, PublicKeyToken:
Dim myAssemblyName As AssemblyName = myAssembly.GetName()
'Из имени сборки при помощи свойства Name
'выделяем имя проекта типа string:
Dim myName_of_project As String = myAssemblyName.Name
Private Sub Form1_Load(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles MyBase.Load
'Загружаем звуковые файлы в массивы:
array_btSoundBounce = Sound.ReadBytesFromStream( _
myAssembly.GetManifestResourceStream( _
myName_of_project + "." + "bounce.wav"))
array_btSoundExplode = Sound.ReadBytesFromStream( _
myAssembly.GetManifestResourceStream( _
myName_of_project + "." + "explode.wav"))
'Воспроизводим звук удара объекта о границы формы:
Sound.Play(array_btSoundBounce)
'Воспроизводим звук взрыва объекта:
'Sound.Play(array_btSoundExplode)
End Sub
```

В панели Solution Explorer выполняем правый щелчок по имени проекта и в контекстном меню выбираем Add, New Item (или Project, Add New Item). В панели Add New Item выделяем шаблон Code File, в окне Name записываем имя Sound.vb и щёлкаем кнопку Add.

В проект (и в панель Solution Explorer) добавляется этот файл, открывается пустое окно редактирования кода, в которое записываем следующий код.

Листинг 6.13. Файл Sound.vb.

```
Imports System.IO 'Для класса Stream.
Public Class Sound
'Константы для управления воспроизведением звукового файла:
Const SND_SYNC = &H0
Const SND_ASYNC = &H1
Const SND_NODEFAULT = &H2
Const SND_MEMORY = &H4
Const SND_LOOP = &H8
Const SND_NOSTOP = &H10
'Из пространства имён System.Runtime.InteropServices
'импортируем библиотеку winmm.dll и
'объявляем функцию PlaySoundBytes этой библиотеки:
```

```
Private Declare Function PlaySoundBytes _
Lib "winmm.dll" Alias "PlaySound" (ByVal szSound() As Byte, _
ByVal hModule As IntPtr, ByVal dwFlags As Integer) As Integer
'Метод для считывания звукового файла:
Public Shared Function ReadBytesFromStream( _
ByRef strBytes As Stream) As Byte()
Dim btRetVal() As Byte
btRetVal = New Byte(strBytes.Length) {}
strBytes.Read(btRetVal, 0, Fix(strBytes.Length))
Return btRetVal
End Function
'Метод для разового воспроизведения звукового файла:
Public Shared Sub Play(ByRef btBytes() As Byte)
PlaySoundBytes(btBytes, IntPtr.Zero, _
SND_ASYNC Or SND_MEMORY)
End Sub
End Class
```

Этот файл Sound.vb можно использовать во многих приложениях и играх для воспроизведения звуковых файлов, добавляя его в проект по стандартной схеме Project, Add Existing Item.

В режиме выполнения (Build, Build Selection; Debug, Start Without Debugging) мы услышим соответствующее (одноразовое) воспроизведение звукового файла, который мы добавили или непосредственно в проект, или в нашу дополнительную папку Sounds проекта.

6.7. Методика воспроизведения звуковых файлов на основе DirectX

Теперь для воспроизведения звуков в приложениях и играх только для настольных компьютеров опишем методику на основе технологии DirectX, точнее, её компонента DirectSound в виде одноименного пространства имён Microsoft.DirectX.DirectSound. Сейчас, применительно к начальным главам по двухмерным играм, эту очень мощную методику мы опишем кратко, а более подробно эта методика будет описана в конце книги для трёхмерных игр.

Приступим к программной реализации воспроизведения звуковых файлов, для общности, в новом проекте.

Для создания проекта в VS щёлкаем кнопку New Project (или File, New, Project). В панели New Project в окне Project Types выбираем тип проекта Visual Basic, Windows, в окне Templates выделяем шаблон Windows Forms Application, в окне Name записываем любое имя проекта и щёлкаем OK. Создаётся проект, появляется форма Form1 (рис. 6.10) в режиме проектирования.

Рис. 6.10. Форма Form1 в режиме выполнения.

Проектируем (или оставляем по умолчанию) эту форму, как описано в параграфе “Методика проектирования формы”. Например, в панели Properties в свойстве Font оставляем по умолчанию или устанавливаем новый шрифт и его размер (Size). Чтобы изменить заголовок формы, в панели Properties в свойстве Text записываем (или вставляем из буфера обмена: правый щелчок, Paste) текст.

На форме размещаем (с панели Toolbox), для примера, две кнопки Button (Sound 1 и Sound 2). В панели Properties в свойстве Text записываем название каждой кнопки Sound &1 и Sound &2 с оператором &, который подчёркивает следующий за ним символ текста, что позволяет нам задействовать кнопку не только мышью, но и, удерживая нажатой клавишу Alt, нажатием клавиши с подчёркнутой буквой английского алфавита или цифрой (например, Alt +1).

На форме в режиме выполнения по умолчанию выделена первая кнопка. Но если мы желаем, чтобы в режиме выполнения на форме была выделена другая кнопка (чтобы нажимать её не только мышью, но и клавишей Enter), то в панели Properties (для Form1) в свойстве AcceptButton выбираем имя этой кнопки.

По варианту 1, для добавления звукового файла drumpad-crash.wav в проект, в меню Project выбираем Add Existing Item, в панели Add Existing Item в окне Files of type устанавливаем All Files, в окне "Look in" находим (в системной папке компьютера C, WINDOWS, Media или, например, в папке с загруженным из Интернета файлом) файл и щёлкаем кнопку Add (или дважды щёлкаем по имени файла). Этот файл мы увидим в панели Solution Explorer. Если мы дважды щёлкнем по этому файлу drumpad-crash.wav, то откроется проигрыватель Windows Media Player, и мы услышим в течение нескольких секунд звучание типа шуршания оркестровых металлических тарелок при воздействии на них металлической метелки.

Аналогично добавляем в проект второй файл drumpad-bass_drum.wav типа удара барабана.

Аналогично можно добавить в проект ещё много звуковых файлов, которые мы желаем послушать во время выполнения приложения или игры.

По второму, закомментированному далее в программе, варианту эти файлы мы не добавляем в проект, а копируем во внешнюю папку с именем, например, Sounds, запоминая путь к этой папке, начиная от локального диска, например, D.

В проекте создаём ссылку на тот компонент DirectX, который потребуется для приложения. Для этого в меню Project выбираем Add Reference, а в панели Add Reference на вкладке (.NET) – компонент Microsoft.DirectX.DirectSound и щёлкаем OK. Если в панели Add Reference пользователь не увидит компонента Microsoft.DirectX.DirectSound, то ему следует с сайта корпорации Microsoft бесплатно загрузить последнюю версию DirectX SDK и стандартно установить на свой компьютер.

Открываем файл Form1.vb (например, так: File, Open, File) и вверху записываем директиву для подключения этого же пространства имён (которое мы добавили в виде ссылки):

```
Imports Microsoft.DirectX.DirectSound
```

Напомним, что эту строку можно и не записывать, но тогда нам придётся перед каждым классом записывать эти пространства имён Microsoft.DirectX.DirectSound.

Теперь в любом месте класса Form1 записываем следующий универсальный (для всех последующих приложений и игр) код для инициализации DirectX.

Листинг 6.14. Универсальный код.

```
'Параметр устр-ва в виде панели визуализации renderWindow
'класса Control из пространства имён System.Windows.Forms:
Dim renderWindow As Control
'Добавляем переменные для воспроизведения звуков:
'Устройство DeviceOfSound класса Device из DirectSound:
Dim DeviceOfSound As Microsoft.DirectX.DirectSound.Device
'Класс вторичного буфера SecondaryBuffer
'для нескольких звуковых объектов-файлов:
Dim mySound1 As SecondaryBuffer ' = Nothing по умолчанию.
Dim mySound2 As SecondaryBuffer ' = Nothing по умолчанию.
'Инициализируем и устанавливаем параметры DirectX:
Public Function InitializeDirectX() As Boolean
Try
'renderWindow связываем с this формой (или эл-м):
renderWindow = Me
'Создаём и инициализируем переменные для звука:
'Устройство DeviceOfSound класса Device из DirectSound:
DeviceOfSound = _
New Microsoft.DirectX.DirectSound.Device()
'Уровень доступа к устройству – многозадачный Normal:
DeviceOfSound.SetCooperativeLevel(renderWindow, _
CooperativeLevel.Normal)
'Инициализация DirectX прошла успешно:
Return True
Catch
'Перехвачена ошибка инициализации DirectX:
Return False
End Try
End Function 'Конец метода InitializeDirectX.
```

Напомним, что именно этот универсальный код можно будет копировать и записывать в наши последующие приложения и игры (после нашего напоминания).

Теперь в панели Properties (для формы Form1)) на вкладке Events дважды щёлкаем по имени события Load (Загрузка).

Появляется файл Form1.vb с шаблоном метода Form1_Load, который после записи нашего кода принимает следующий вид.

Конец ознакомительного фрагмента.

Текст предоставлен ООО «ЛитРес».

Прочитайте эту книгу целиком, купив полную легальную версию на ЛитРес.

Безопасно оплатить книгу можно банковской картой Visa, MasterCard, Maestro, со счета мобильного телефона, с платежного терминала, в салоне МТС или Связной, через PayPal, WebMoney, Яндекс.Деньги, QIWI Кошелек, бонусными картами или другим удобным Вам способом.