Доктор технических наук
Жарков Валерий Алексеевич

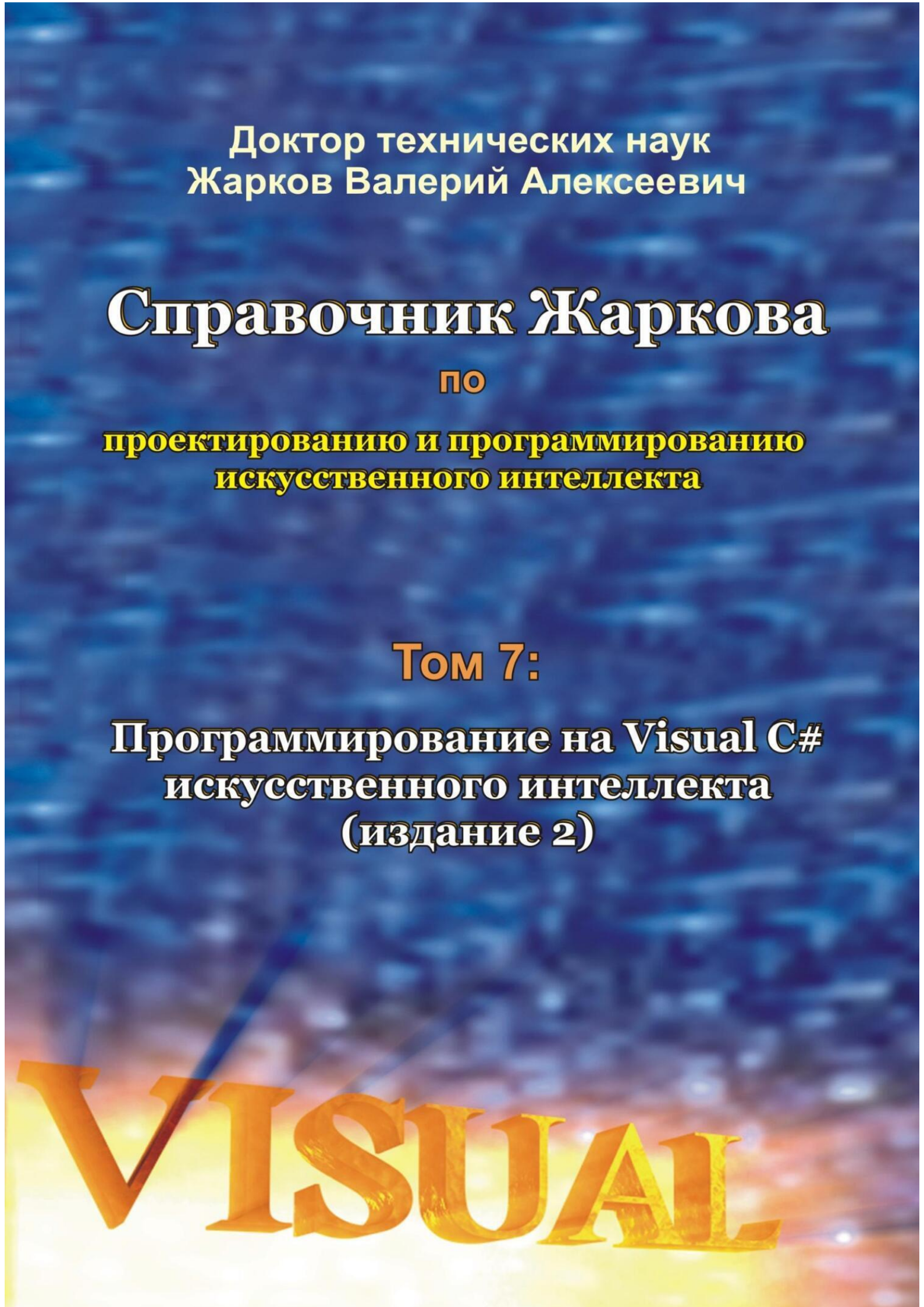
Справочник Жаркова

по

проектированию и программированию искусственного интеллекта

Том 7:

Программирование на Visual C# искусственного интеллекта (издание 2)



Валерий Жарков

Справочник Жаркова по проектированию и программированию искусственного интеллекта. Том 7: Программирование на Visual C# искусственного интеллекта. Издание 2

«Автор»
2023

Жарков В. А.

Справочник Жаркова по проектированию и программированию искусственного интеллекта. Том 7: Программирование на Visual C# искусственного интеллекта. Издание 2 / В. А. Жарков — «Автор», 2023

В серии книг “Справочник Жаркова по проектированию и программированию искусственного интеллекта” в нескольких томах собрано лучшее программирование искусственного интеллекта (ИИ) в двух- и трёхмерных играх и приложениях, разработанных как автором, так и взятые из Интернета за многие годы и доработанные автором. Программирование ИИ на Visual C# написано в IX частях, которые разделены на два тома 7 и 8. В томе 7 дано. I. Краткие основы Visual C#. II. Учебное программирование игр и приложений с подвижными объектами. III. Программирование ИИ в играх в “Крестики-нолики”. IV. Программирование ИИ в спортивных играх на примере игры в теннис. V. Программирование ИИ в играх по сборке фигур из элементов одинакового цвета или одинаковой геометрии. VI. Распространение приложения. Даны полные тексты программ на VC# для Игрока с Компьютером, которые можно переписать на другой язык. Книги предназначены для желающих изучить программирование ИИ в играх и приложениях на базе VC# и других языков.

Содержание

Валерий Жарков
Справочник Жаркова по проектированию и программированию искусственного интеллекта. Том 7: Программирование на Visual C# искусственного интеллекта. Издание 2

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, планшетов и смартфонов на основе самого популярного, совершенного и перспективного языка программирования высокого уровня Visual C# 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 дают 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 (заменяющей 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 C# и давать подробные объяснения по теории языка VC#. Если у читателя возникнуть вопросы,

он легко отыщет книгу по данному языку (некоторые полезные по данной тематике книги 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 VC#, по возможности не используя другие книги; опытному пользователю также будут полезны эти пояснения, поскольку книги по разработке ИИ на основе новых версий языка Visual C# 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 с использованием 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 из списка литературы (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 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 данного тома справочника.

Введение. **Часть I. Краткие основы Visual C#.** Глава 1. Основные определения книги. Глава 2. Методика разработки приложений для выполнения расчётов с эффектами анимации. Глава 3. Методика разработки приложений на нескольких формах и передачи данных с одной формы на другую. **Часть II. Учебная методология программирования игр и приложений с подвижными объектами.** Глава 4. Методика анимации и управления подвижными объектами. Глава 5. Методика обнаружения столкновений, программирования уничтожений летающих объектов и подсчёта очков. Глава 6. Методология воспроизведения звуковых файлов. Глава 7. Методика улучшения графики и добавления фона экрана. Глава 8. Методика программирования игры с летающими объектами на основе спрайтов. Глава 9. Игра с летающими объектами на основе спрайтов, двух форм и возможности приостановки и повторного запуска игры. Глава 10. Игра с изменяемой траекторией летающих объектов. **Часть III. Методология программирования искусственного интеллекта в играх в "Крестики-нолики" для Игрока с Компьютером и двух Игроков.** Глава 11. Методика программирования искусственного интеллекта в игре в "Крестики-нолики" на сетке 3x3. Глава 12. Методика программирования искусственного интеллекта в игре в «Крестики-нолики» на сетке 6x7. **Часть IV. Методология программирования искусственного интеллекта в спортивных играх на примере игры в теннис для Игрока с Компьютером или двух Игроков.** Глава 13. Методика программирования искусственного интеллекта в игре в теннис на основе элементов управления на одной форме. Глава 14. Методика программирования искусственного интеллекта в игре в теннис на основе элементов управления на двух формах. **Часть V. Методоло-**

гия программирования искусственного интеллекта в играх по сборке фигур из элементов одинакового цвета или одинаковой геометрии для Игрока с Компьютером или двух Игроков. Глава 15. Методика программирования искусственного интеллекта в игре по сборке фигур одинакового цвета с учётом соединения игроков через локальную сеть или Интернет. **Часть VI. Развёртывание, публикация и распространение разработанной игры или приложения с искусственным интеллектом.** Глава 16. Методика распространения игры или приложения. Заключение. Литература.

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) 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 те результаты, которые необходимы именно им и характерны для их профессиональных или непрофессиональных (игровых) интересов.

Книга предназначена для всех желающих быстро изучить основы программирования искусственного интеллекта в разнообразных двухмерных и трёхмерных компьютерных играх и приложениях на базе самого популярного, совершенного и перспективного (в мире программирования) языка высокого уровня **Visual C#** последних версий для настольных компьютеров, ноутбуков, планшетов и смартфонов, на этих основах сразу же проектировать ИИ в сложных играх и приложениях и применять их 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 (доктор технических наук Жарков Валерий Алексеевич) продолжит описывать программирование ИИ в следующих играх и приложениях.

Вопросы, замечания и предложения по тематике книги можно направлять по email с сайта ZharkovPress.ru.

Часть I. Краткие основы Visual C#

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 C# 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 [Литература] и 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 C#, 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 C# 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 и Visual Studio), или специалиста любого профиля, вводу исходных данных и выводу результатов расчёта, а также решению различных задач при помощи диалоговых панелей и форм (точнее, при помощи принципов визуального программирования) и объектно-ориентированного программирования (ООП) на языке высокого уровня Visual C#;

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 заменят (или дополнят) другие известные системы 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) на языке Visual C# среды разработки Visual Studio последней или предыдущих версий по следующей стандартной и 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 C#, Windows и проверяем, чтобы в окне Templates был выделен (как правило, по умолчанию) шаблон Templates, Visual C#, Windows Classic Desktop, Windows Forms App (.NET Framework); в окне Name записываем имя проекта, например, Calculator.

В расположенном ниже окне Location мы видим каталог (папку 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 при помощи значка Create New Folder.

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.

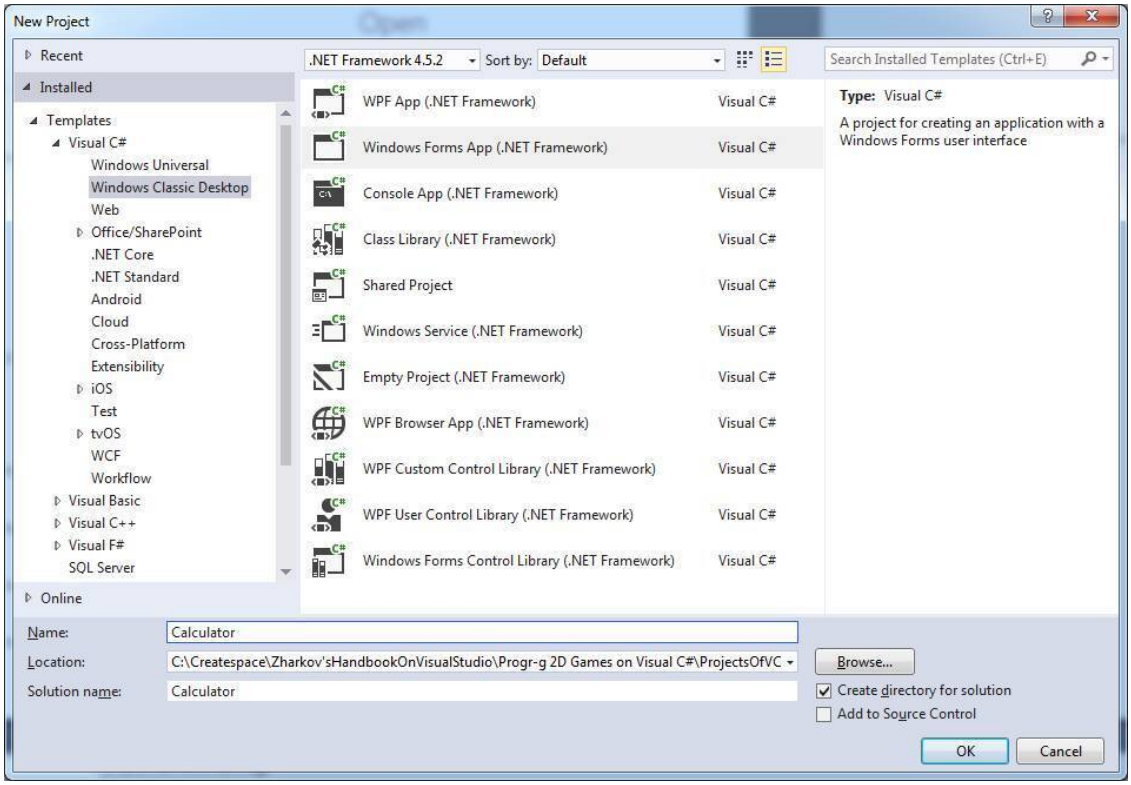


Рис. 2.1. В окне Project Types выбираем тип проекта Visual C#, Windows.

4. В панели New Projects (рис. 2.1) щёлкаем OK.

Visual C# создаёт проект приложения и выводит форму Form1 в режиме проектирования (иначе, дизайна или редактирования) с вкладкой Form1.cs [Design], при помощи которой далее можно будет открывать эту форму (рис. 2.2).

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-редакторе Microsoft Word);

если мы создаём решение (Solution) 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 нашем случае Calculator.

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

Compile complete – 0 errors, 0 warnings

Calculator -> D:\MyDocs\DesktopProjects\DesktopProjects2\Calculator\bin\Debug \Calculator.exe

========== Build: 1 succeeded or up-to-date, 0 failed, 0 skipped ==========

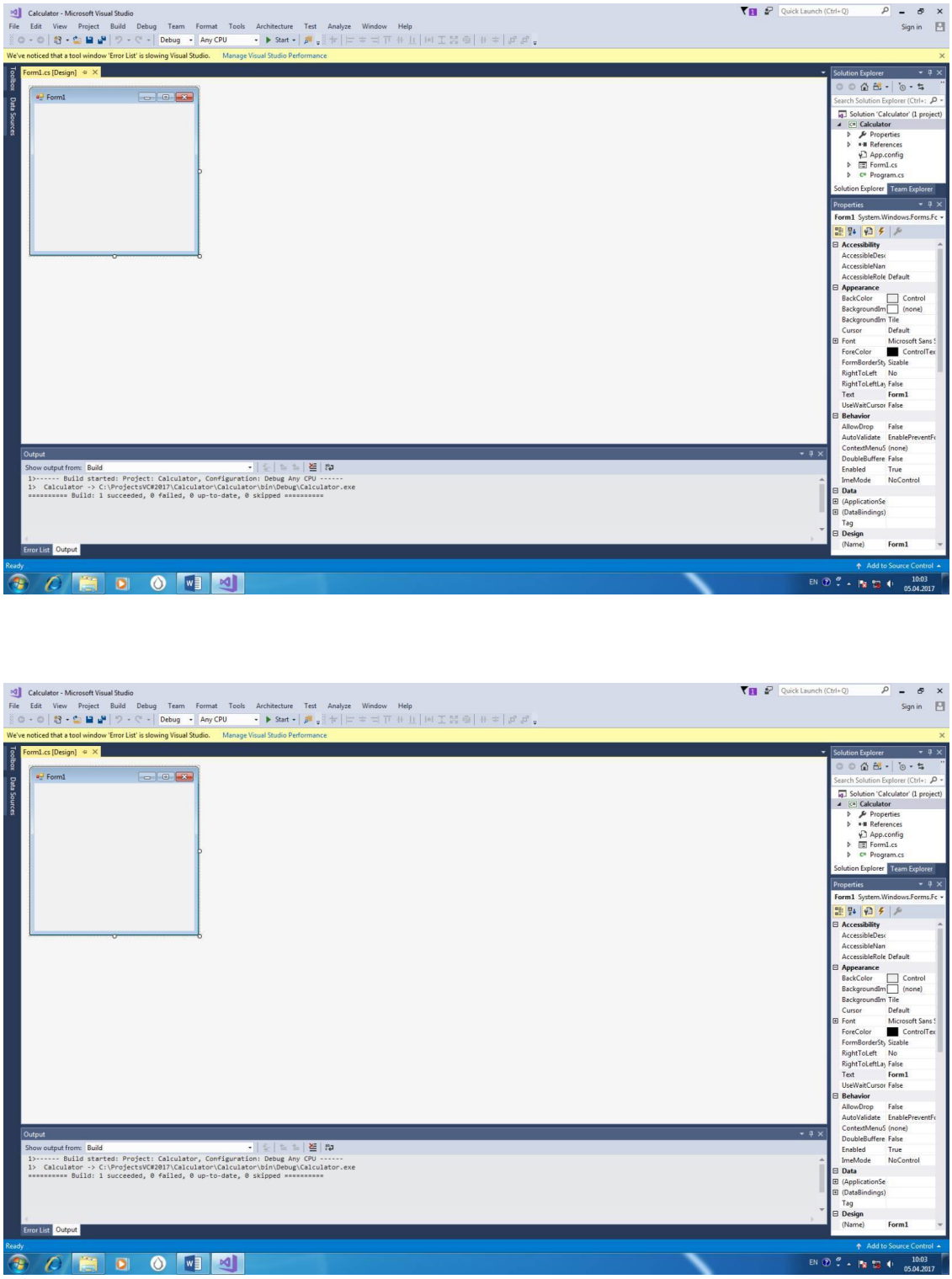


Рис. 2.2. Форма Form1 в режиме проектирования (иначе, дизайна или редактирования).

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 соответствующего 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 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. 2.3) визуального графического проектирования формы выполняем следующие типичные шаги инструкции.

1. Панель Properties (Свойства) со свойствами формы Form1 (рис. 2.4) 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 (диалоговая панель) в режиме проектирования, то на рабочем столе щёлкаем вкладку Form1.cs [Design] или в панели Solution Explorer (Исследователь-Проводник Решения) дважды щёлкаем пункт Form1.cs.

2. В панели Properties с заголовком Form1 (рис. 2.4) 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. 2.4.

5. При помощи свойства BackColor (рис. 2.4) мы можем установить (вместо установленного по умолчанию цвета Control) из списка другой цвет клиентской области Form1. Перечень этих цветов показан в наших предыдущих книгах [Литература]. Напомним, что в клиентскую область не входит верхняя полоска с текстом и граница формы.

6. При помощи свойства BackgroundImage (Фон), рис. 2.4, 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 поочередно сменять друг друга в режиме анимации; затем на этом изменяющемся фоне можно размещать элементы управления (например, TextBox и Button) и компоненты (например, Timer), как показано в наших предыдущих книгах [Литература].

7. Напомним, что на инструментальной панели Toolbox (рис. 2.3) 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. 2.3) щёлкаем элемент GroupBox (рамка группы). Выполняем щелчок на форме. Чтобы изменить название groupBox1, в панели Properties в свойстве Text вместо groupBox1 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 рамки группы.

8. Аналогично размещаем первое окно текста TextBox (для ввода первого слагаемого); в панели Properties появляется новый заголовок textBox1; для удаления (в окне) надписи textBox1, в панели Properties в свойстве Text стираем текст (или последовательно нажимаем две клавиши пробела и Enter).

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

10. Чтобы пользователю было понятно, что означает каждое окно, около каждого окна вводим надпись Label. Например, первую надпись label1 мы можем изменить или сразу с клавиатуры, или в панели Properties с заголовком label1 в свойстве Text вместо label1 записываем (или вставляем из буфера обмена: правый щелчок, Paste), например, число 1 (number 1).

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 button1 мы можем изменить или сразу с клавиатуры, или в панели Properties в свойстве Text вместо button1 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 Class View появились эти элементы управления. Форма приняла окончательный внешний вид (рис. 2.6).

Отметим, что на приведённых рисунках видны отличия новой платформы Visual Studio от предыдущих версий этой же платформы, которые подробно описаны в наших предыдущих книгах из списка литературы и с сайта .

Важно отметить, что все программы, которые далее мы будем разрабатывать, применимы для всех версий и редакций среды разработки (платформы) Visual Studio.

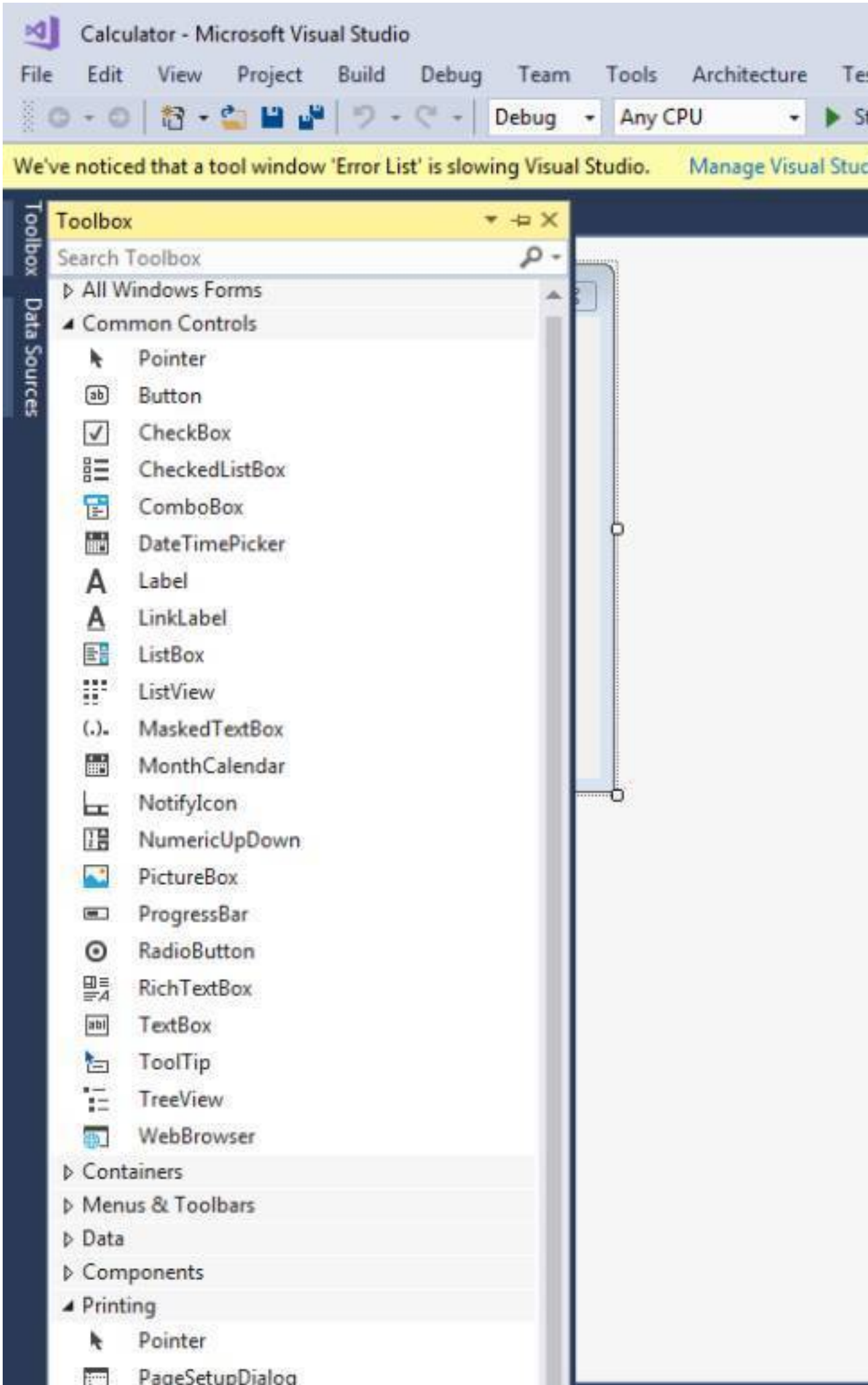

Calculator - Microsoft Visual Studio
File Edit View Project Build Debug Team Tools Architecture
Debug
Any CPU
We've noticed that a tool window 'Error List' is slowing Visual Studio.
Manage Visual Stud
Toolbox
Data Sources
Toolbox
Search Toolbox
All Windows Forms
Common Controls
Pointer
Button
CheckBox
CheckedListBox
ComboBox
DateTimePicker
Label
LinkLabel
ListBox
ListView
MaskedTextBox
MonthCalendar
NotifyIcon
NumericUpDown
PictureBox
ProgressBar
RadioButton
RichTextBox
TextBox
ToolTip
TreeView
WebBrowser
Containers
Menus & Toolbars
Data
Components
Printing
Pointer
PageSetupDialog

Рис. 2.3. Панель инструментов Toolbox. **Рис. 2.4.** Панели Solution Explorer и Properties.

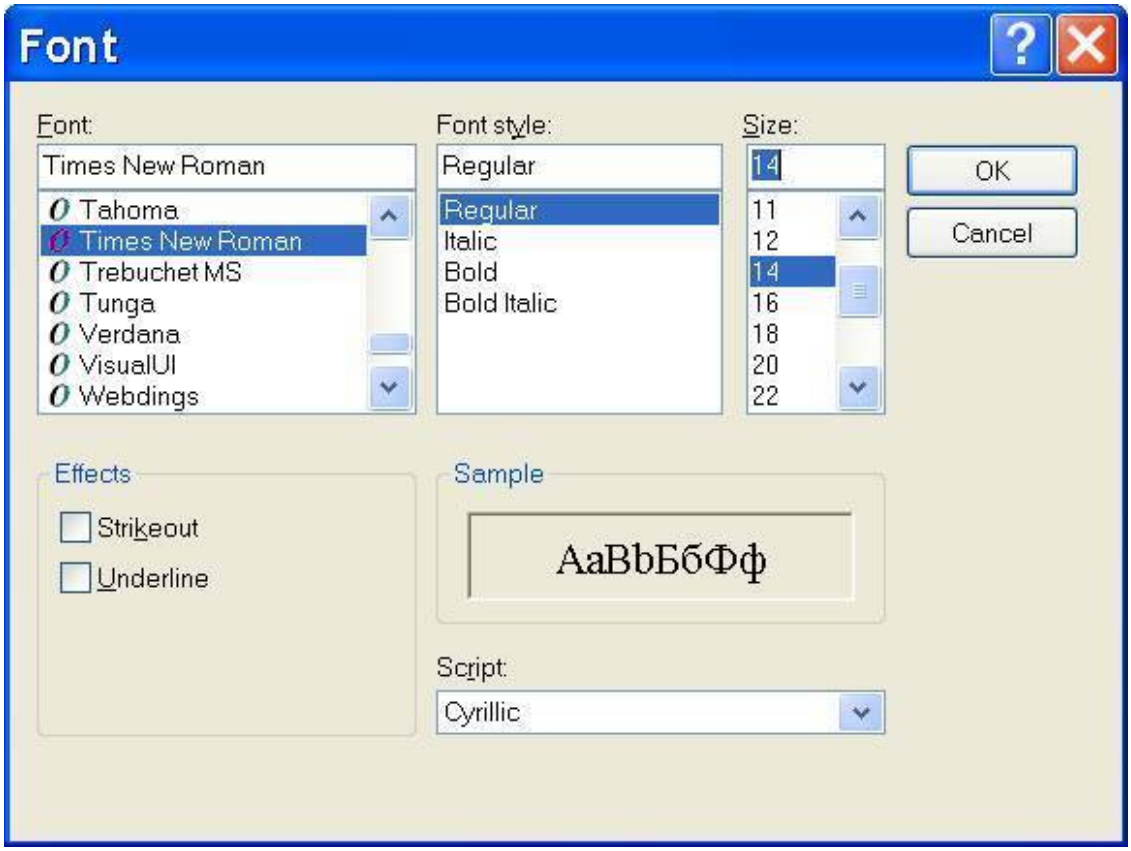


Рис. 2.5. В панели Font устанавливаем шрифт.

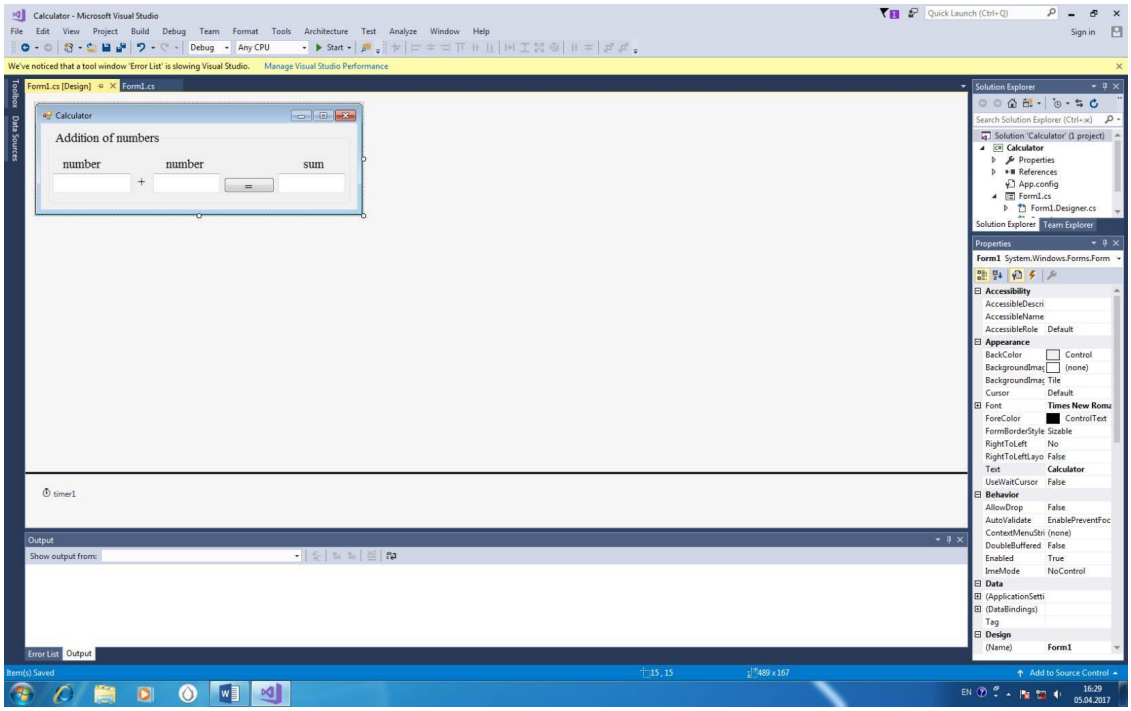


Рис. 2.6. Форма после графического (визуального) проектирования.

2.4. Код программы

Теперь в файл Form1.cs 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. 2.4). 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.cs со следующим автоматически сгенерированным кодом:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
namespace Calculator
{
public partial class Form1 : Form
{
public Form1()
{
InitializeComponent();
}
}
}
```

И в этот файл Form1.cs уже автоматически добавлен следующий шаблон метода для обработки события в виде щелчка кнопки:

```
private void button1_Click(object sender, EventArgs e)
{

}
```

Так как выше в коде уже подключено корневое пространство имён (using 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 использованием элементов управления, например, TextBox) программирования, приведём четыре (по мере усложнения) варианта кода для сложения двух чисел.

Первый вариант – с тремя переменными:

Листинг 2.1. Наш основной код с тремя переменными.

```
double a, b, c;
a = Convert.ToDouble(textBox1.Text);
```

```
b = Convert.ToDouble(textBox2.Text);
c = a + b;
textBox3.Text = c.ToString();
```

Второй вариант – с двумя переменными:

```
double a, b;
a = Convert.ToDouble(textBox1.Text);
b = Convert.ToDouble(textBox2.Text);
textBox3.Text = (a + b).ToString();
```

Третий вариант – с одной переменной:

```
double a;
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 вертикальных стрелок) мы увидим в выходном окне Output сообщение компилятора C# 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 соответствующим изображением).

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.

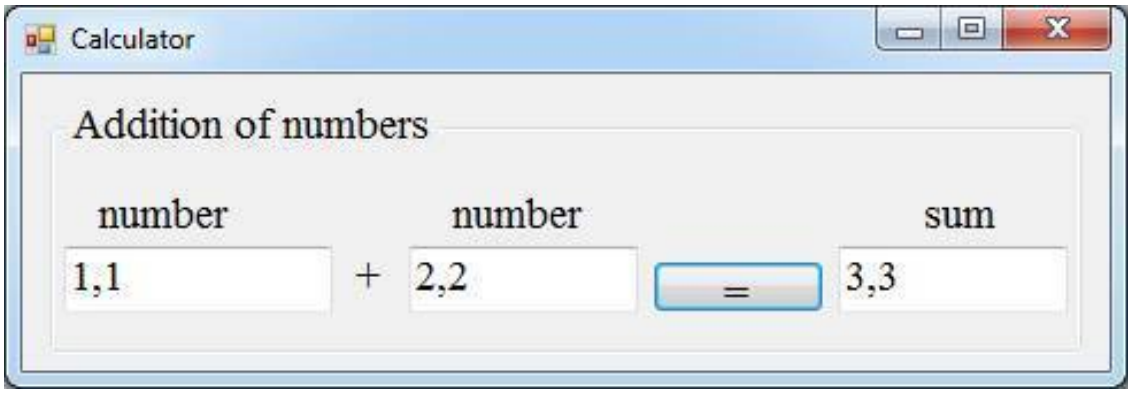


Рис. 2.7. Пример сложения двух чисел.

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, эти варианты кода на листинге 2.1 соответствуют российским (и международным) стандартам, когда десятичная дробь записывается с запятой. Поэтому в дальнейшем мы будем применять именно эти варианты 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 языка программирования Visual C# (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.

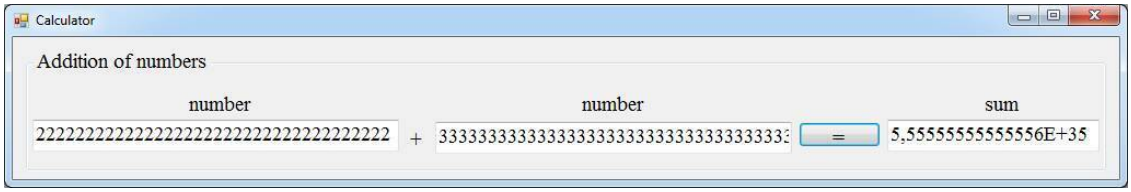


Рис. 2.8. Результат сложения двух больших чисел.

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 – тридцать один (31).

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.

Теперь разработаем методику создания анимации, на основании которой разработаем первый эффект анимации для данной Form1.

2.7. Общая методика создания анимации

Разработаем общую методику создания анимации в различных приложениях 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 схеме:

1. На панели инструментов Toolbox щёлкаем строку Timer (рис. 2.3).

2. Щёлкаем на форме.

Ниже Form1 появляется значок с надписью timer1 (рис. 2.9), который можно захватить мышью и перенести в другое место.

Отметим, что в отличие от элементов управления компоненты располагаются вне формы и поэтому на форме в режиме выполнения не видны.

3. В панели Properties с заголовком timer1:

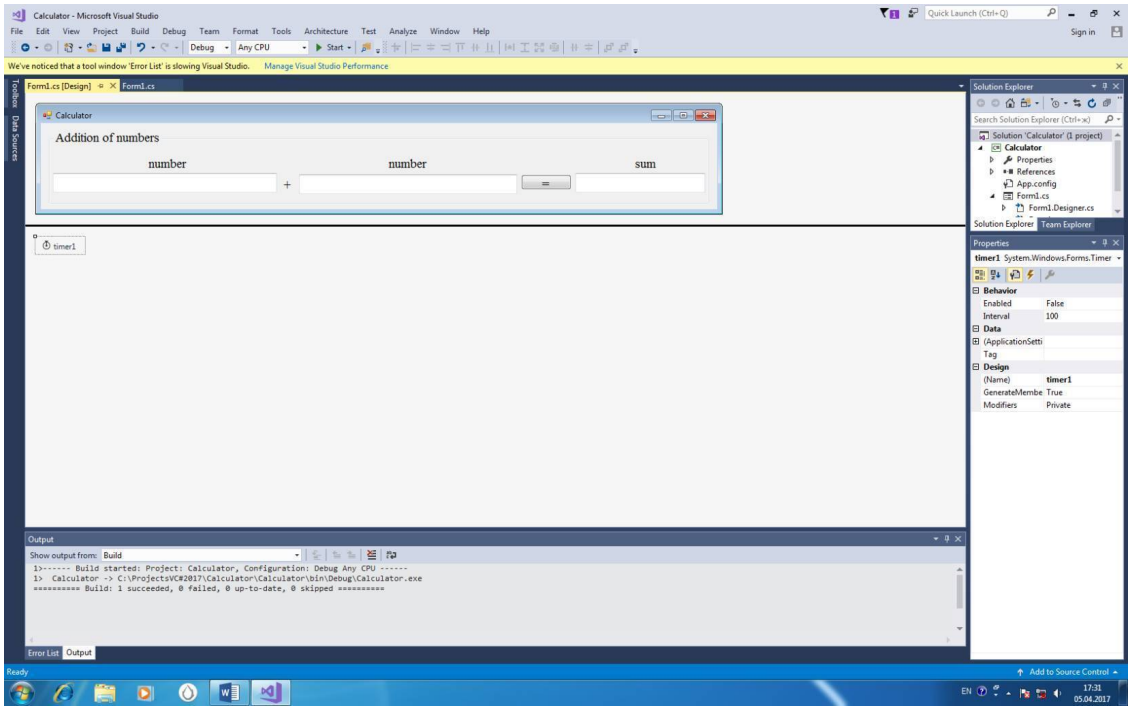
в свойстве Enabled вместо False выбираем True (рис. 2.10);

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 следующего кода.

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

```
private void InitializeTimer()
{
//We turn on the timer:
timer1.Enabled = true;
//We generate the Tick event through each Interval of time:
timer1.Interval = 500;
}
```



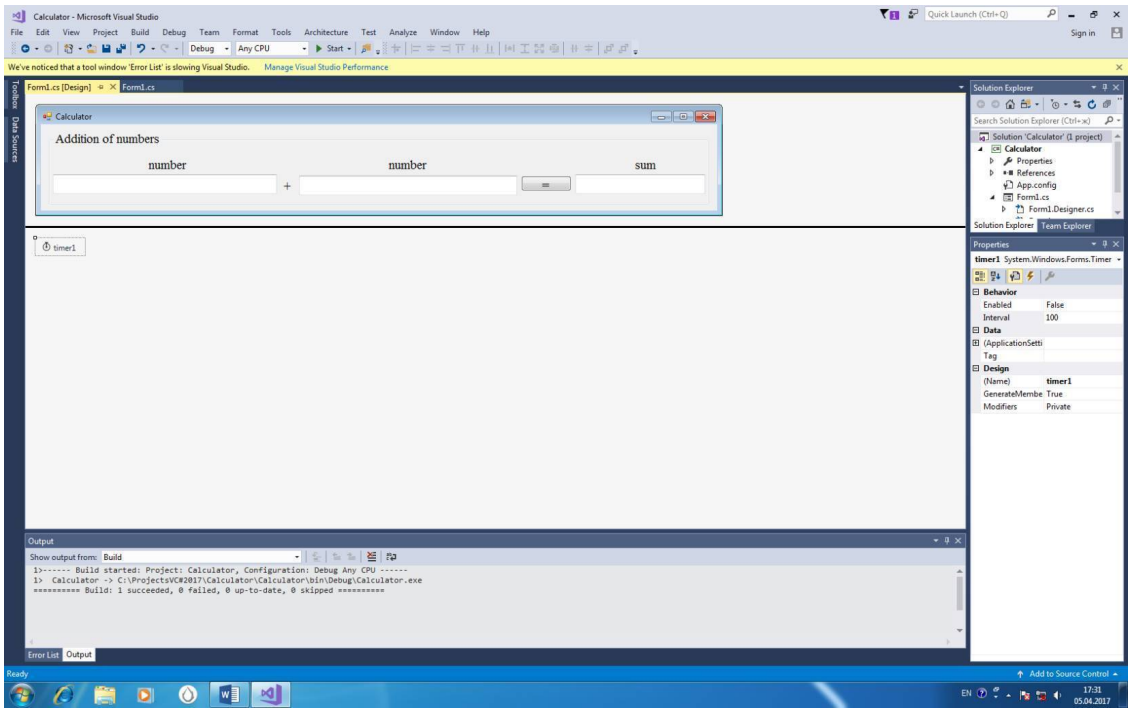


Рис. 2.9. Значок компонента Timer. **Рис. 2.10.** Панель Properties.

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.

Следовательно, мы закончили визуальную разработку анимационного эффекта и теперь нам необходимо написать код программы. Этот код может иметь много вариантов.

Рассмотрим, например, три варианта:

1) анимация выполняется безостановочно столько, сколько выполняется наше приложение, и анимационный объект (текст, кнопка, цвет и т.д.) изменяется за каждый интервал времени Interval;

2) анимация выполняется безостановочно столько, сколько выполняется наше приложение, но анимационный объект изменяется не за каждый интервал времени Interval, а через заданное нами число интервалов N_Interval ;

3) анимация выполняется столько времени, какое число интервалов времени мы задали, и после этого времени анимационный объект останавливается в заданном нами положении.

Сначала изучим первый вариант, когда анимация выполняется безостановочно за каждый интервал времени Interval. Для этого дважды щёлкаем значок timer1 (рис. 2.8) в режиме проектирования (или в панели Properties на вкладке Events дважды щёлкаем по имени единственного события Tick).

Появляется файл Form1.cs с автоматически сгенерированным шаблоном метода, выше которого объявляем булеву переменную, а в шаблон записываем такой код.

Листинг 2.3. Код для создания анимации. Вариант 1.

```
//We declare the Boolean myText variable and appropriate it
//”false” (by default too "false"):
bool myText = false;
private void timer1_Tick(object sender, EventArgs e)
{
//We set the alternation of " Calculator " and
//" Calculator with animation ":
if (myText == false)
{
this.Text = " Calculator ";
//We change the myText value to opposite:
myText = true; //or so: myText =! myText;
}
else
{
this.Text = " Calculator with animation ";
//We change the myText value to opposite:
myText = false; //or so: myText =! myText;
}
}
```

В этом коде использовано правило переноса текста с одной строки на другую.

Теперь изучим второй вариант, когда анимация выполняется безостановочно столько, сколько выполняется наше приложение, но анимационный объект изменяется не за каждый интервал времени Interval, а через заданное нами число интервалов N_Interval. Для этого в файле Form1.cs выше того же (что и в предыдущем варианте) автоматически сгенерированного шаблона метода объявляем переменные, а в шаблон записываем код, как показано на следующем листинге.

Листинг 2.4. Код для создания анимации. Вариант 2.

```
//We declare the counter of number of intervals
//and set its initial value:
private int counter = 0; //by default too it is equal to 0.
//We declare and set the number N_Interval of intervals,
//after which one text is replaced by another:
private int N_Interval = 3;
```

```
private void timer1_Tick(object sender, EventArgs e)
{
//We check, value of the counter
//equally or yet is not present to number N_Interval of
//intervals,
//after which one text is replaced by another:
if (counter >= N_Interval)
{
//If value of the counter collected and is equal
//N_Interval, we output other text:
this.Text = "Calculator";
//We nullify the value of the counter again:
counter = 0;
}
else
{
//If value of the counter did not collect yet
//and N_Interval is not equal,
//that we output the first text:
this.Text = "Calculator with animation";
//We increase value of the counter on 1:
counter = counter + 1;
}
}
```

И наконец, изучим третий вариант, когда анимация выполняется столько времени, какое число интервалов времени N_Interval_Stop мы задали, и после этого времени анимационный объект останавливается в заданном нами положении. Для этого в файле Form1.cs выше того же (что и в предыдущем варианте) автоматически сгенерированного шаблона метода объявляем большее число переменных, а в шаблон записываем более общий код, как показано на следующем листинге.

Листинг 2.5. Код для создания анимации. Вариант 3.

```
//We declare the counter of number of intervals
//and set its initial value:
private int counter = 0;
//We declare and set the number N_Interval of intervals,
//after which one text is replaced by another:
private int N_Interval = 3;
//We declare and nullify the i_Interval_Stop counter,
//which calculates the number of intervals
//to an animation stop:
private int i_Interval_Stop = 0;
//We declare and set the number N_Interval_Stop of intervals,
//on reaching which animation stops:
private int N_Interval_Stop = 10;
private void timer1_Tick(object sender, EventArgs e)
{
//Value of the i_Interval_Stop counter,
//which calculates the number of intervals
//to an animation stop, we increase on 1:
```

```
i_Interval_Stop = i_Interval_Stop + 1;
//We check the number i_Interval_Stop of intervals,
//on reaching which animation stops:
if (i_Interval_Stop >= N_Interval_Stop)
timer1.Enabled = false;
//We check, value of the counter
//equally or yet is not present to number N_Interval of
//intervals,
//after which one text is replaced by another:
if (counter >= N_Interval)
{
//If value of the counter collected and is equal
//N_Interval, we output other text:
this.Text = "Calculator";
//We nullify value of the counter again:
counter = 0;
}
else
{
//If value of the counter did not collect yet
//and N_Interval is not equal,
//that we output the first text:
this.Text = "Calculator with animation";
//We increase value of the counter on 1:
counter = counter + 1;
}
}
```

Так как здесь мы впервые применили метод timer1_Tick, а далее постоянно будем его применять, то дадим краткие пояснения.

Автоматически сгенерированный заголовок метода

private void timer1_Tick(object sender, EventArgs e)

говорит нам о том, что метод timer1_Tick обрабатывает (Handles) событие Tick, периодически (с заданным интервалом при помощи свойства Interval) возбуждаемое объектом (таймером) timer1. В строке (bool myText = false;) мы объявляем булеву глобальную переменную myText выше метода timer1_Tick. Если бы переменную myText мы задали в виде локальной переменной внутри метода timer1_Tick, то при каждом новом вызове (с заданным интервалом) этого метода timer1_Tick значение переменной myText оставалось бы неизменным, и анимации не получилось бы.

По какому-либо одному варианту кода, например, по первому варианту строим программу и запускаем на выполнение обычным образом: Build, Build Selection; Debug, Start Without Debugging.

В ответ Visual C# выполняет программу и на рабочий стол 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. 2.10), и таким образом создаётся эффект анимации.

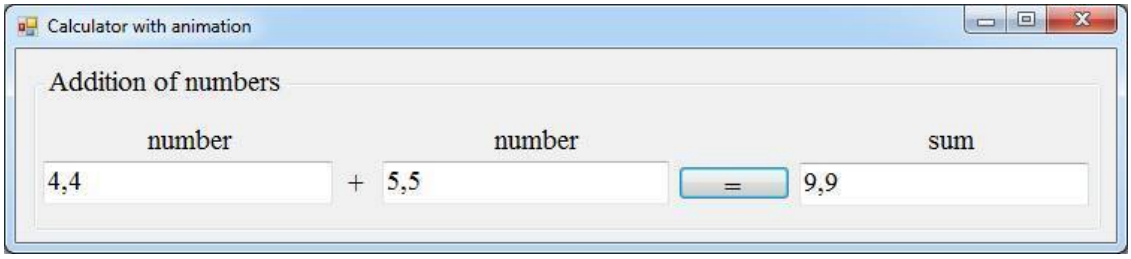


Рис. 2.10. Форма с анимационным заголовком.

Если на листингах 2.3 – 2.5 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. 2.10).

Аналогично создаётся анимация по второму варианту (листинг 2.4) и третьему варианту (листинг 2.5); каждый вариант имеет свои особенности. И читатель может выбрать наиболее понравившийся вариант анимации. Мы же в дальнейшем будем применять, в основном, первый вариант, как наиболее простой.

Следовательно, мы закончили разработку методики создания трёх вариантов анимации на примере анимационного заголовка формы. Подчеркнем, что мы разработали именно **общую методику создания анимации**, так как если в программах на листингах 2.3 – 2.5 вместо ключевого слова this записать значение свойства Name для какого-нибудь элемента управления (label1, button1 и т.д.), то мы получим эффект анимации для текста на этом элементе управления.

2.8. Методика приостановки и возобновления анимации

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 2.6. 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 2.7. 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 Button, в свойстве Text записываем &Stop/Start Animation и дважды щёлкаем по этой кнопке (или в

панели Properties для этой кнопки на вкладке Events дважды щёлкаем по имени события Click). Появляется файл Form1.cs с автоматически сгенерированным шаблоном метода, выше которого объявляем булеву переменную, а в шаблон записываем код, как показано на следующем листинге:

Листинг 2.8. Код для приостановки и возобновления анимации.

```
//We declare the Boolean OffOn variable and set it "false":
bool OffOn = false;
private void button2_Click(object sender, EventArgs e)
{
//We set alternation of a stop and resuming of animation
//after each click of the button2 button:
if (OffOn == false)
{
//We stop the animation:
timer1.Enabled = false;
//We change the OffOn value to opposite:
OffOn = true; //or so: OffOn =! OffOn;
}
else
{
//We resume the animation:
timer1.Enabled = true;
//We change OffOn value to opposite:
OffOn = false; //or so: OffOn =! OffOn;
}
}
```

Для проверки этого кода по первому варианту (листинг 2.3) запускаем программу, например, так: Ctrl+F5. В ответ Visual C# выполняет программу и на рабочий стол 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.

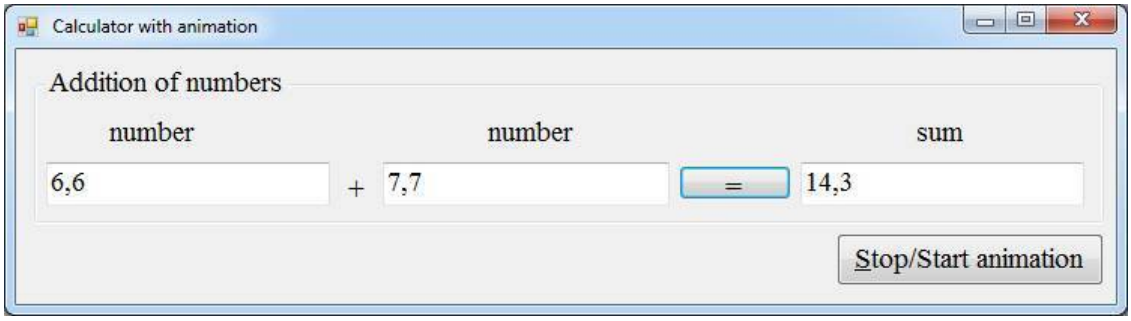


Рис. 2.11. Анимация прекращается и возобновляется после щелчка кнопки Stop/Start Animation.

Анимация прекращается и возобновляется поочередно после каждого щелчка кнопки (рис. 2.11). 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.

Если мы желаем, чтобы анимация прекращалась и возобновлялась после каждого щелчка по данной форме, то в панели Properties для этой формы на вкладке Events дважды щёлкаем по имени события Click и в появившийся шаблон метода записываем код, подобный коду на листинге 2.8.

Аналогично можно разработать другие варианты анимации, а также варианты приостановки и возобновления анимации и мультипликации, как показано в наших предыдущих книгах, например, [Литература] или с сайта ZharkovPress.ru.

2.9. Общая методика использования методов из других платформ на основе динамически подключаемых библиотек

Часто при решении задачи требуется использовать метод (процедуру или функцию), которой в данной программе на данном языке нет, но мы точно знаем, что она имеется в другой программе (комплексе, пакете, платформе) на том же или другом языке и там выполняет то, что нам нужно. Для использования в нашем приложении метода (функции) из любого другого языка, например, из языка Visual Basic, необходимо создать ссылку на этот язык.

Для создания ссылки, например, на Visual Basic выполняем такие шаги.

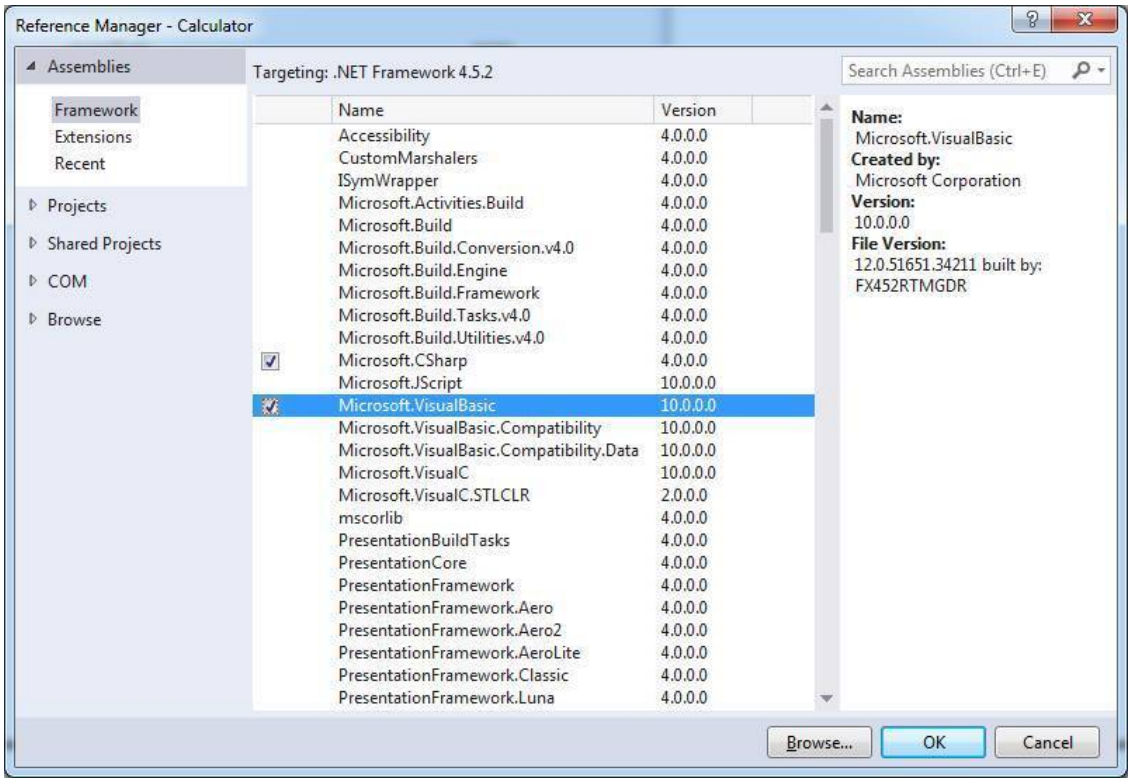
1. В меню Project выбираем Add Reference (или в панели Solution Explorer делаем правый щелчок на имени проекта и в появившемся контекстном меню выбираем Add Reference).

Мы увидим панель Add Reference (рис. 2.12).

2. В панели Add Reference на вкладке (.NET) выделяем динамически подключаемую библиотеку (dynamic link library), например, Microsoft.VisualBasic и щёлкаем кнопку OK.

Эта ссылка добавляется в список ссылок в панели Solution Explorer (рис. 2.13).

Таким образом мы создали ссылку на Visual Basic и теперь можем подключать к нашему приложению методы (процедуры и функции) из этого языка, как показано в следующем параграфе.



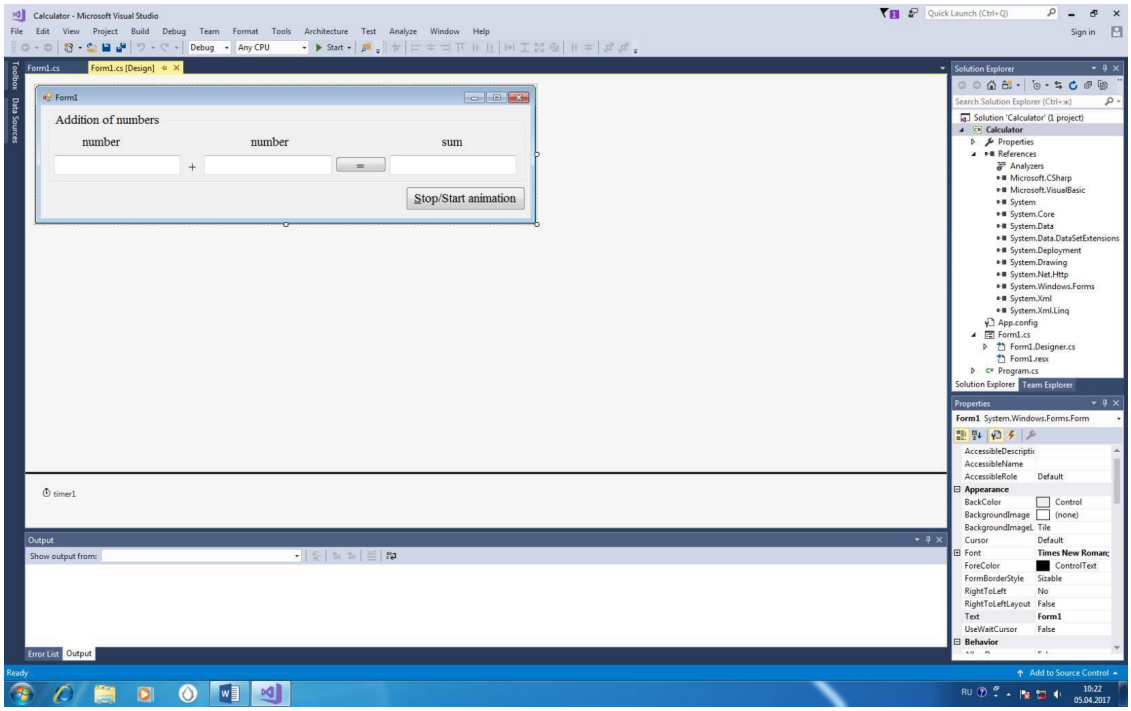


Рис. 2.12. Выбираем Microsoft.VisualBasic. **Рис. 2.13.** Панель Solution Explorer.

2.10. Методика использования методов из других платформ на примере подачи звукового сигнала

Целесообразно, чтобы в работающем приложении эффекты анимации сопровождались звуковыми эффектами, и самым простым из них является звуковой сигнал.

В одной из наших предыдущих книг мы уже писали, что подача звукового сигнала в Visual Basic основана на том, что в тело функции для обработки какого-либо события в любом приложении следует записать стандартную функцию Beep(). Если в комплексе Visual Basic мы запишем Beep() в функцию для обработки события Tick таймера, то звуковой сигнал будет периодически создаваться согласно генерируемому событию Tick с заданным нами интервалом времени.

Если мы запишем Beep() в функцию для обработки, например, события Tick таймера в комплексе Visual C#, то выйдет сообщение об ошибке, что в Visual C# такой функции нет.

Согласно приведённой в предыдущем параграфе методике использования в нашем приложении метода (функции) из любого другого комплекса, создаём ссылку на Visual Basic. Когда в шаблон функции для обработки, например, события Tick таймера в Visual C# мы запишем (после пространства имён Microsoft) оператор в виде точки (.), то увидим подсказку из списка пространств имён, которые мы можем применить в данном приложении (рис. 2.14).

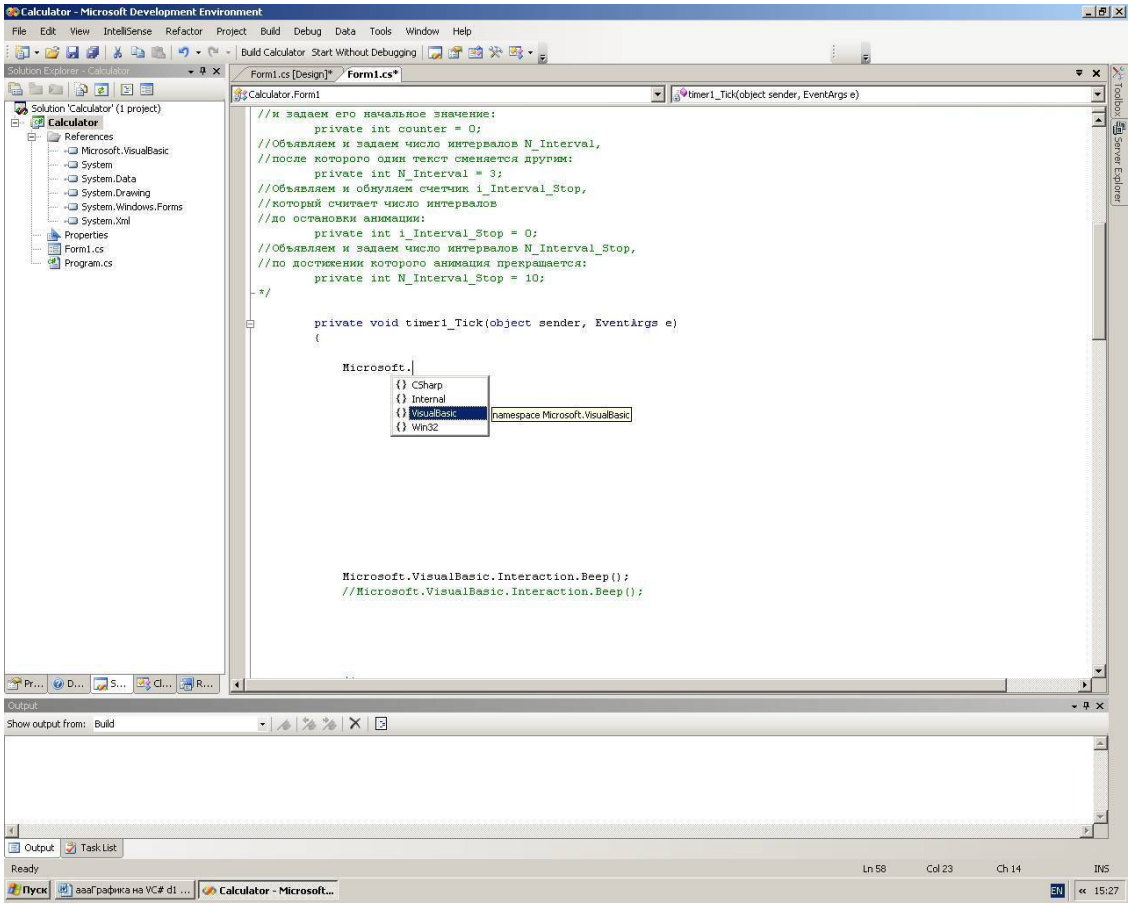


Рис. 2.14. Пространства имён, которые можно применить в приложении VC#.

Дважды щёлкаем по имени Visual Basic, тем самым вставляя это имя в код (Microsoft.VisualBasic).

Когда далее после имен Microsoft.VisualBasic мы запишем тот же оператор (.), то увидим подсказку из списка классов пространства имён VisualBasic, которые мы можем применить в данном приложении (рис. 2.15). Отметим, что если бы ранее мы не создали ссылку на динамически подключаемую библиотеку (dynamic link library) Microsoft.VisualBasic, то сейчас мы бы не увидели этого списка классов. Дважды щёлкаем по имени класса Interaction, тем самым вставляя это имя в код:

Microsoft.VisualBasic.Interaction

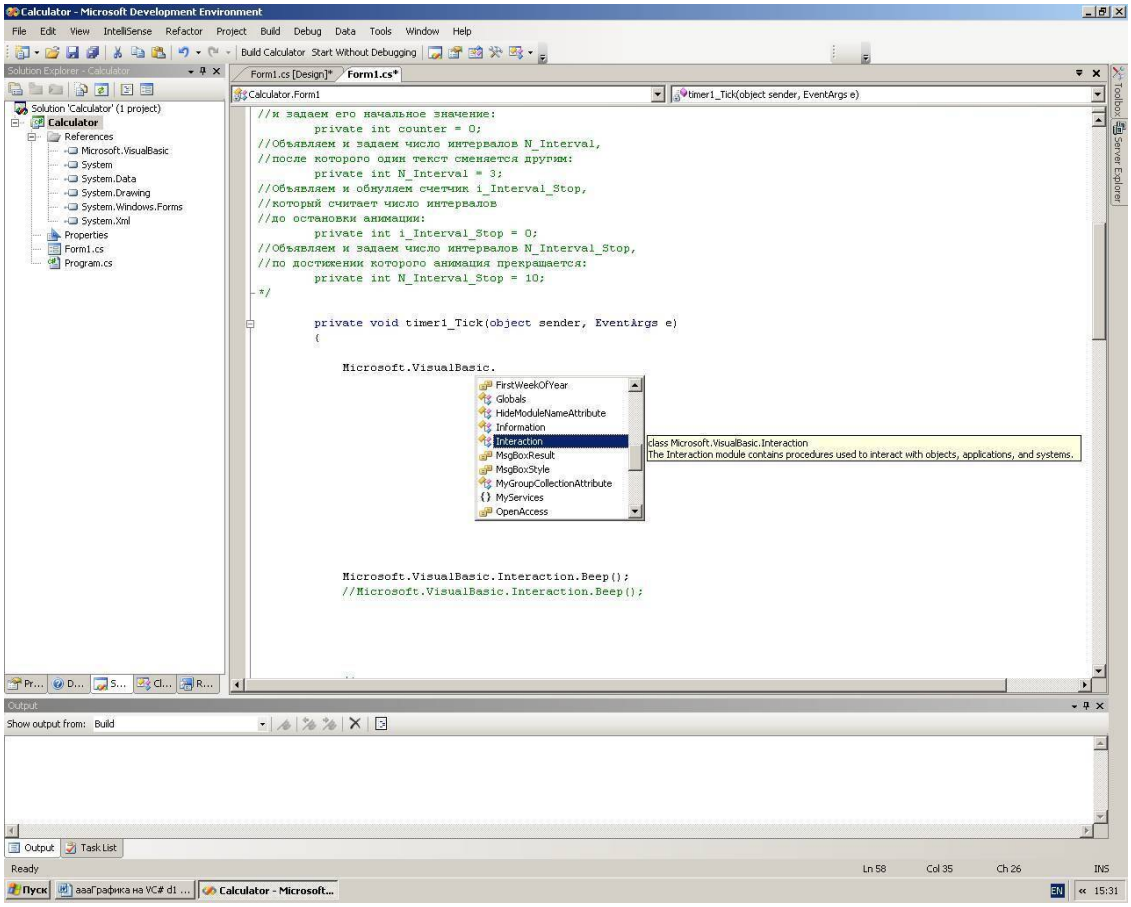


Рис. 2.15. Список классов, которые можно применить в приложении.

Когда далее после имен Microsoft.VisualBasic.Interaction мы запишем оператор (.), то увидим подсказку в виде списка методов (процедур и функций) Visual Basic из класса Interaction, которые мы можем применить в данном приложении (рис. 2.16). Дважды щёлкаем по имени функции Beep(), тем самым вставляя это имя в код (Microsoft.VisualBasic.Interaction.Beep()).

Таким образом, мы закончили запись в шаблон метода для обработки, например, события Tick таймера в Visual C# одной строки кода с методом Beep():

Microsoft.VisualBasic.Interaction.Beep();

Этот звуковой сигнал Beep будет периодически создаваться согласно генерируемому событию Tick с заданным нами интервалом времени Interval.

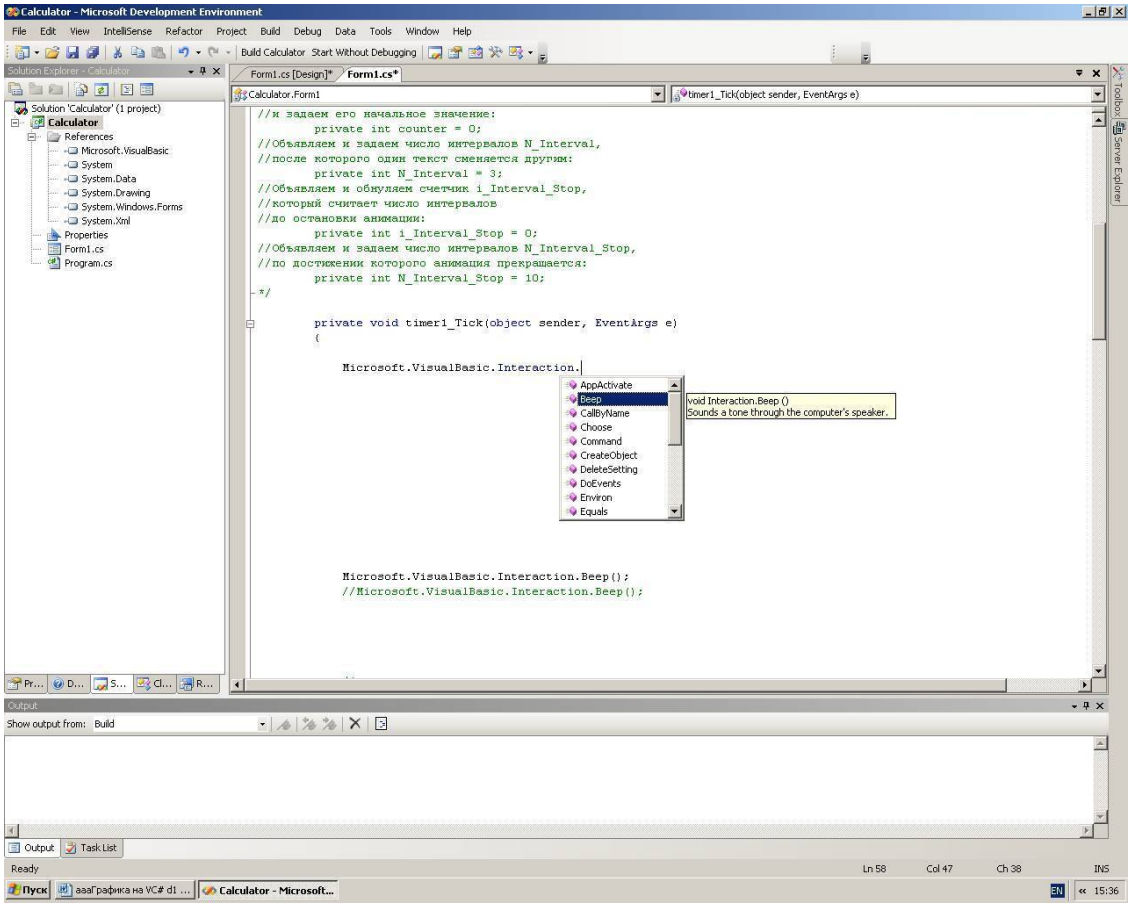


Рис. 2.16. Список методов, которые можно применить в приложении.

Если мы хотим ограничить число звуковых сигналов величиной N_Beep, то выше шаблона метода объявляем и инициализируем (например, 10) эту переменную N_Beep:

```
//We declare the counter of number of intervals
//and set its initial value:
int i = 0;
//We declare and set the number N_Beepof of intervals,
//after which a giving of a sound signal will stop:
int N_Beep = 10;
```

Далее в шаблоне метода организовываем цикл по переменной i:

```
i = i + 1;
if (i <= N_Beep)
Microsoft.VisualBasic.Interaction.Beep();
```

Далее 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 далее 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 новейшей версии Visual C# с учётом эффектов 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 расчёта на эту же форму, то 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 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 с рисунками приведено в предыдущей главе), опишем создание проекта для нового приложения-диалога.

1. В VS щёлкаем значок New Project (или выбираем File, New, Project).

2. В панели New Projects (показанной выше) в окне Project Types выделяем Visual C#; в окне Templates проверяем, что выделен (по умолчанию) шаблон Templates, Visual C#, Windows Classic Desktop, Windows Forms App (.NET Framework); в окне Name печатаем имя проекта, например, Calculator2_2 (первая цифра 2 означает второй вариант калькулятора, а вторая цифра 2 – на двух формах). Щёлкаем OK.

В ответ Visual C# создаёт проект нашего приложения и выводит форму Form1 (аналогично проекту в предыдущей главе).

По разработанной выше методике осуществляем визуальное проектирование формы (рис. 3.1) и вводим элементы управления (рамку группы, окна, кнопки, тексты) и компонент таймер (свойства таймера Timer: значение Enabled изменяем на True; значение Interval, например, оставляем по умолчанию, равным 100 миллисекундам).

3.2. Проектирование следующей формы

Для ввода в проект новой формы в меню Project выбираем Add Windows Form (или в панели Solution Explorer делаем правый щелчок по имени проекта и выбираем Add, Add Windows Form). Мы увидим панель Add New Item (рис. 3.2).

В панели Add New Item оставляем все по умолчанию и щёлкаем кнопку Add (в предыдущих версиях VS щёлкаем кнопку Open). В ответ Visual C# выводит рабочий стол VS с новой

Form2, такой же, как Form1 (рис. 3.3) и добавляет в панель Solution Explorer новый пункт Form2.cs (рис. 3.4). Если форма Form2 не появилась, то в панели Solution Explorer дважды щёлкаем по пункту Form2.cs (рис. 3.4).

Аналогично, как первую, проектируем данную форму Form2 (рис. 3.3).

По этой схеме можно добавлять и большее количество форм, сколько необходимо для каждого конкретного приложения.

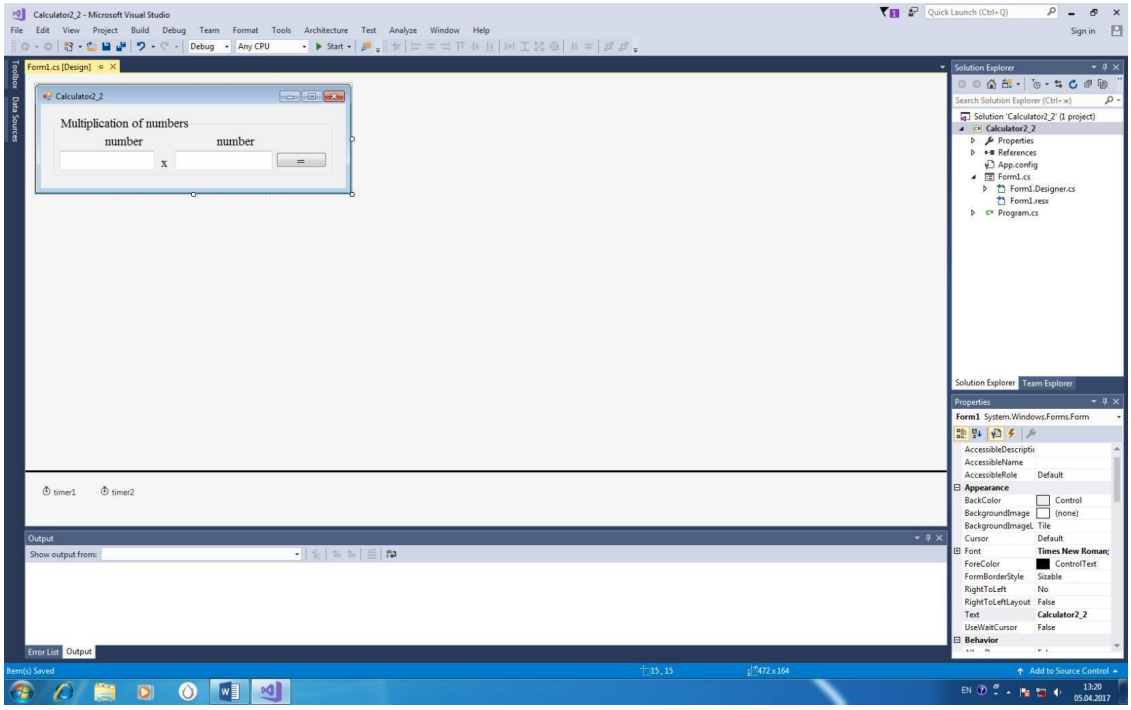


Рис. 3.1. Первая форма в режиме проектирования.

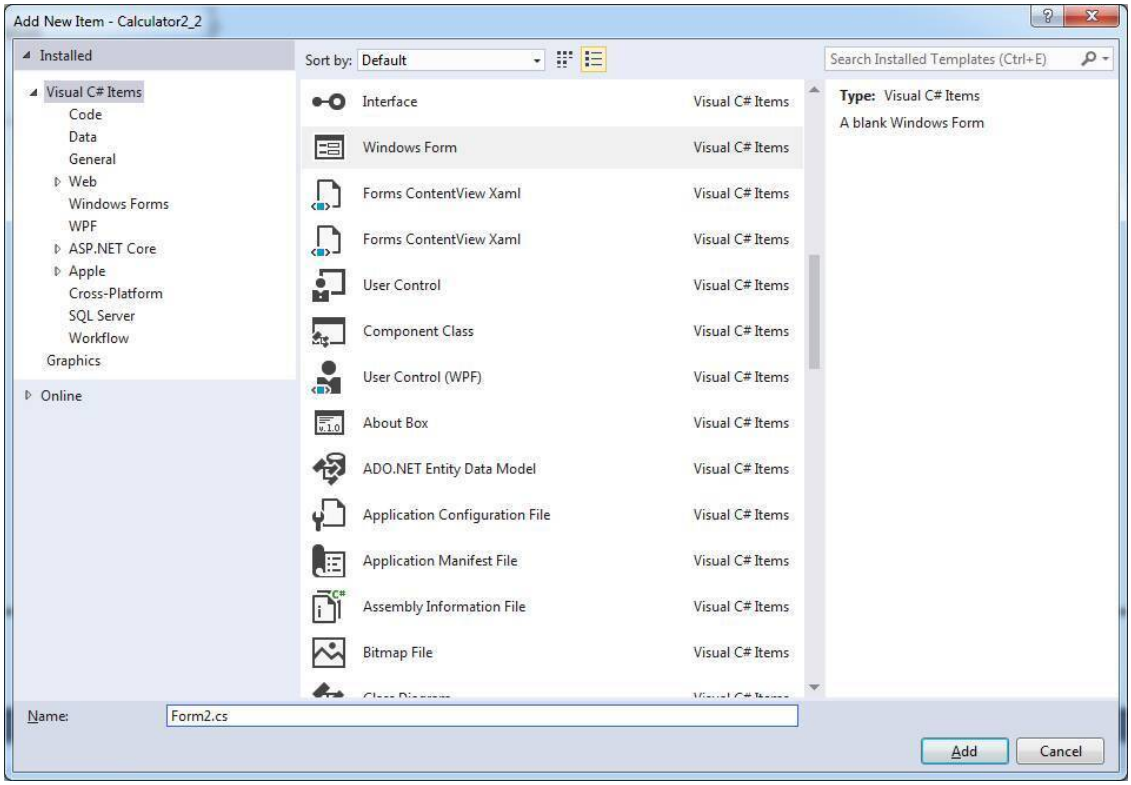
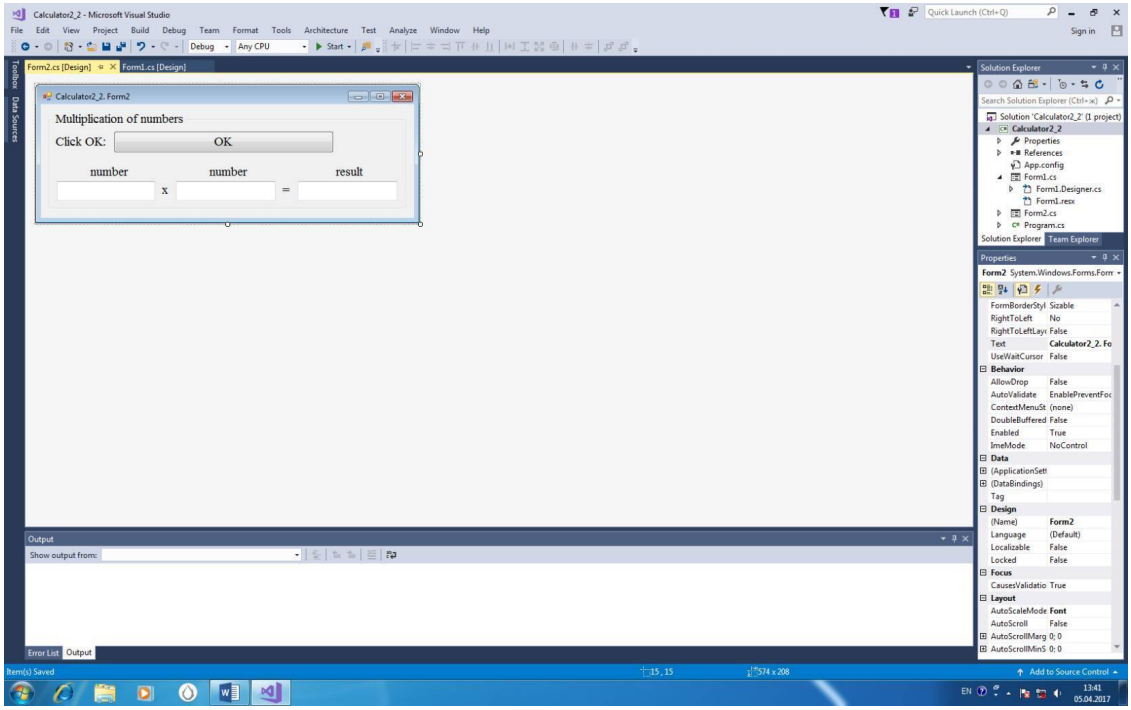


Рис. 3.2. В панели Add New Item оставляем все по умолчанию и щёлкаем Add.



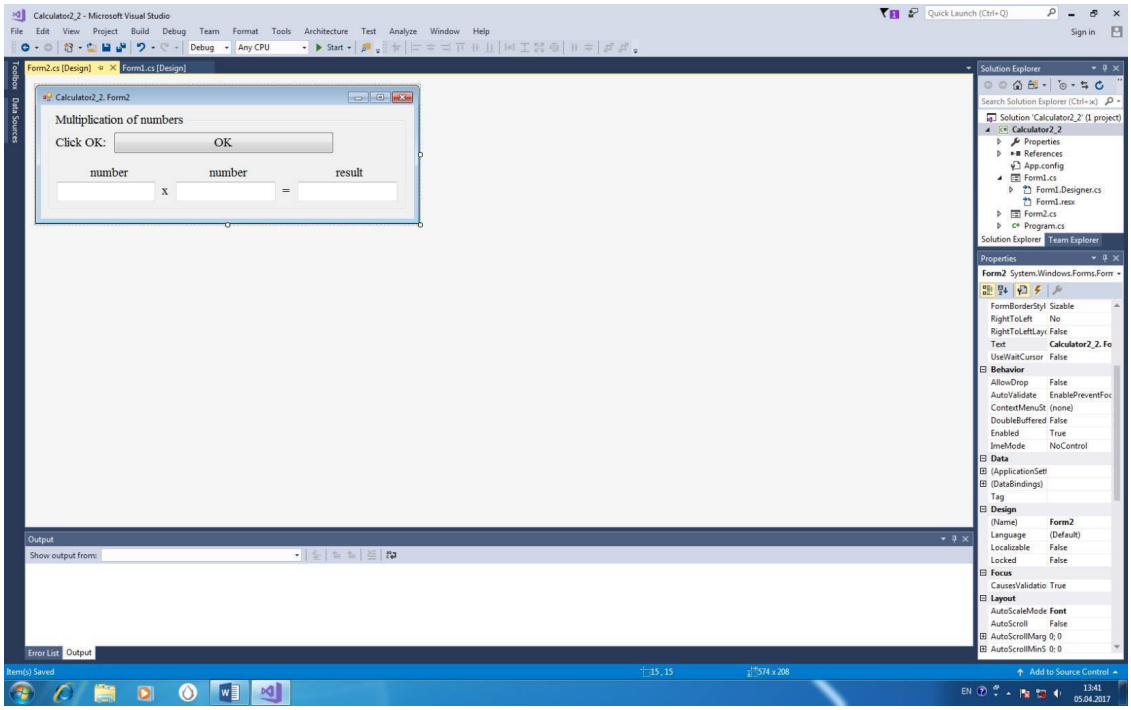


Рис. 3.3. Form2 после проектирования. **Рис. 3.4.** Панель Solution Explorer.

3.3. Код программы

Дважды щёлкаем кнопку на Form1 в режиме проектирования. Появившийся шаблон (после записи нашего кода) принимает вид следующего метода.

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

```
private void button1_Click(object sender, EventArgs e)
{
double A, B;
A = Convert.ToDouble(textBox1.Text);
B = Convert.ToDouble(textBox2.Text);
Form2 myForm2 = new Form2();
myForm2.C = A;
myForm2.D = B;
myForm2.Show();
}
```

Напомним, что в Visual C#, в отличие от Visual Basic, имена переменных чувствительны к регистру букв. Метод Show() показывает немодальную форму, а модальную форму выводит метод ShowDialog().

Дважды щёлкаем кнопку 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 следующего метода.

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

```
public double C, D;
private void button1_Click(object sender, EventArgs e)
{
double F, G;
F = C;
```

```
G = D;
textBox1.Text = F.ToString();
textBox2.Text = G.ToString();
textBox3.Text = (F * G).ToString();
}
```

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.

Более подробно эти программы объяснены в наших книгах с сайта ZharkovPress.ru.

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;

бегущий справа – налево заголовок следующей формы.

Алгоритм бегущего слева – направо заголовка первой формы 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 timer1 ниже первой формы в режиме проектирования. Появляется файл Form1.cs с шаблоном метода timer1_Tick для обработки события Tick, периодически (с заданным интервалом) возбуждаемого объектом (таймером) timer1. Перед шаблоном объявляем глобальную переменную, а внутри этого шаблона записываем наш код, как показано на следующем листинге.

Листинг 3.3. Код для бегущего слева – направо заголовка.

```
//We declare and nullify the global variable:
int i = 0;
private void timer1_Tick(object sender, EventArgs e)
{
//We write the text of heading in the myString variable:
string myString = "Calculator2_2 ";
//On the right – to the left appear the “i”-letter of
//a heading of form:
this.Text = myString.Substring(0, i);
//We organize a cycle of output of
//the following “i”-letter:
i = i + 1;
if (i == myString.Length)
i = 1;
}
```

Наши комментарии в коде позволять читателю грамотно создать аналогичный бегущий заголовок в его приложении.

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 timer1 ниже второй формы в режиме проектирования.

Появляется файл Form2.cs с шаблоном; перед этим шаблоном объявляем глобальную переменную, а внутри шаблона записываем наш код, как показано на следующем листинге.

Листинг 3.4. Код для бегущего справа – налево заголовка.

```
//We declare the global variable "myString"
//and write in it the text of heading:
public static string myString = "Calculator2_2. Form2 ";
//We declare the global variable “i”
//and equate its value – to number of signs of heading:
int i = myString.Length;
private void timer1_Tick(object sender, EventArgs e)
{
//At the left – to the right leaves one “i”-letter
//of a heading:
this.Text = myString.Substring(0, i);
//We organize a removal cycle of
//the following “i”-letter of heading:
i = i – 1;
if (i == -1)
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

this.Text = myString.Substring(0, i);

после ключевого слова this 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 C# выполняет программу и на рабочий стол выводит первую форму с пустыми окнами и мигающим курсором в первом окне (рис. 3.5). Мы видим также бегущий слева – направо заголовок формы.

2. В первое окно вводим первый сомножитель.

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

Появляется вторая форма (рис. 3.6) с нулями во всех трёх окнах. Мы видим также бегущий справа – налево заголовок формы.

4. На второй форме щёлкаем кнопку OK.

В ответ Visual C# на второй форме показывает (рис. 3.6):

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 C# закрывает вторую форму, но оставляет открытой первую форму. Мы можем ввести другие значения в окна первой формы и аналогично получить результат умножения других чисел.

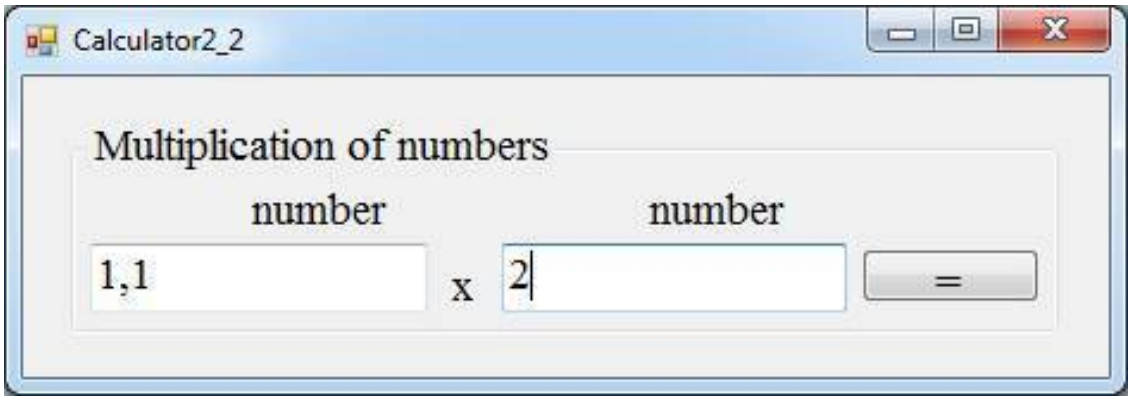


Рис. 3.5. Первая форма.

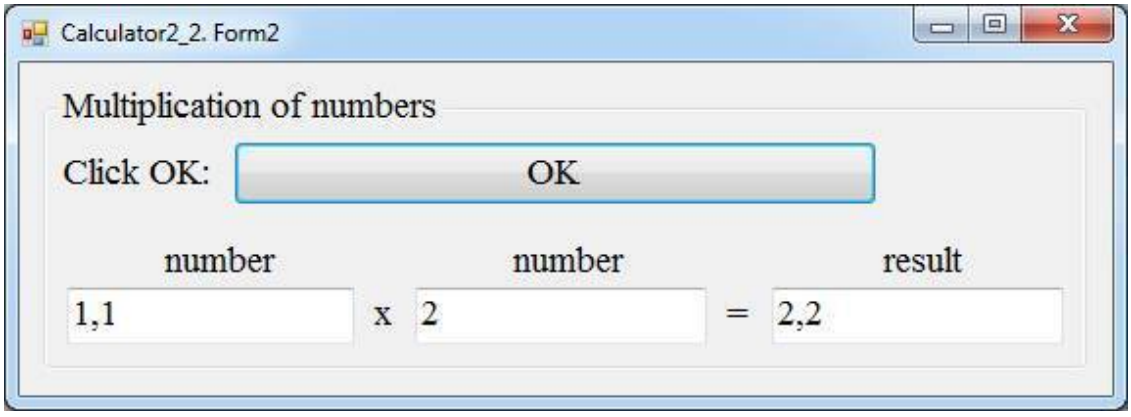


Рис. 3.6. Вторая форма.

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

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

Появляется второй вид второй формы с нулями во всех окнах.

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

Аналогично можно получить любое количество видов 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 данном томе из серии книг, следуя её названию, а также следуя приведённым в предыдущих главах основам, приступим к разработке игр и приложений на платформе Visual Studio для настольных компьютеров, ноутбуков, планшетов и смартфонов.

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

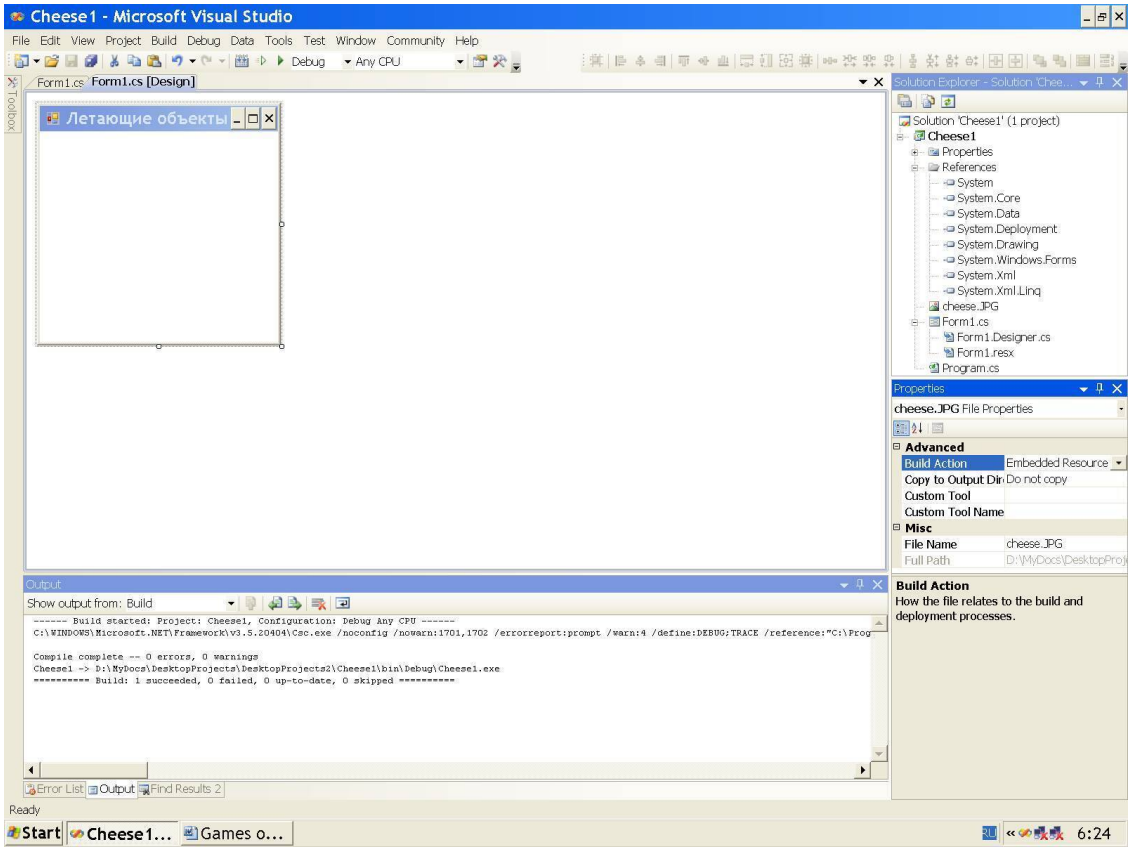
Разработаем общую обучающую методику создания типичной и широко распространённой игры, когда в качестве летающих игровых объектов используются продукты питания, следуя статье с сайта microsoft.com: Rob Miles. Games Programming with Cheese: Part One. Так как эта статья написана по программированию игры на смартфоне 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, при помощи новейшей версии Visual Studio.

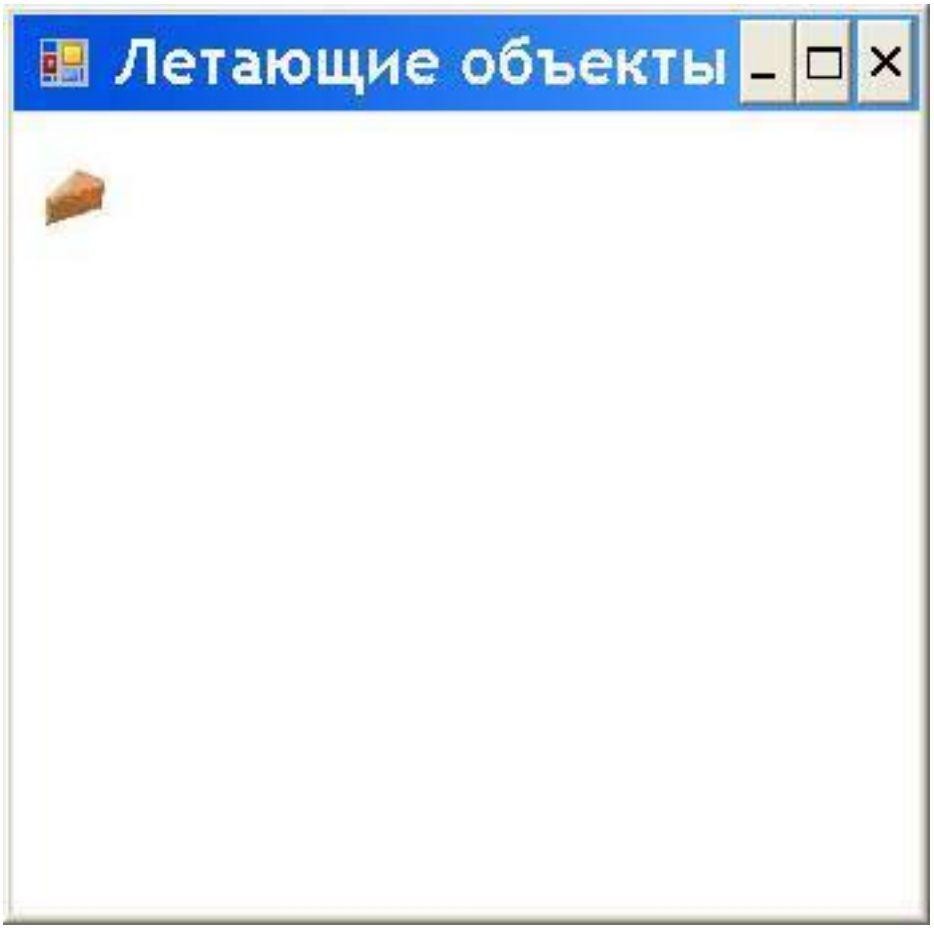
Общие требования к программному обеспечению для разработки этой игры приведены выше. Методично и последовательно начнём решать типичные задачи (с подробными объяснениями) по созданию данной базовой учебной игры и всех подобных игр типа аркады (arcade).

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 объем файла любого изображения, то можно воспользоваться каким-либо графическим редактором, например, Paint, который поставляется с любой операционной системой Windows.

Игру с летающими 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 C#, Windows, в окне Templates выделяем шаблон Templates, Visual C#, Windows Classic Desktop, Windows Forms App (.NET Framework), в окне Name записываем (или оставляем по умолчанию) имя проекта 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.


Cheese1 - Microsoft Visual Studio
File Edit View Project Build Debug Data Tools Test Window Community Help
Debug
Any CPU
Form1.cs [Design]
Летающие объекты
Solution Explorer - Solution 'Chee...
Solution 'Cheese1' (1 project)
Cheese1
Properties
References
System
System.Core
System.Data
System.Deployment
System.Drawing
System.Windows.Forms
System.Xml
System.Xml.Linq
cheese.JPG
Form1.cs
Form1.Designer.cs
Form1.resx
Program.cs
Properties
cheese.JPG File Properties
Advanced
Build Action
Embedded Resource
Copy to Output Dir
Do not copy
Custom Tool
Custom Tool Name
Misc
File Name
cheese.JPG
Full Path
Build Action
How the file relates to the build and deployment processes.
Output
Show output from: Build
Build started: Project: Cheese1, Configuration: Debug Any CPU
Compile complete -- 0 errors, 0 warnings
Cheese1 -> D:\MyDocs\DesktopProjects\DesktopProjects2\Cheese1\bin\Debug\Cheese1.exe
Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped
Error List
Output
Find Results 2
Ready
Start
Cheese1...
Games o...
6:24


Летающие объекты

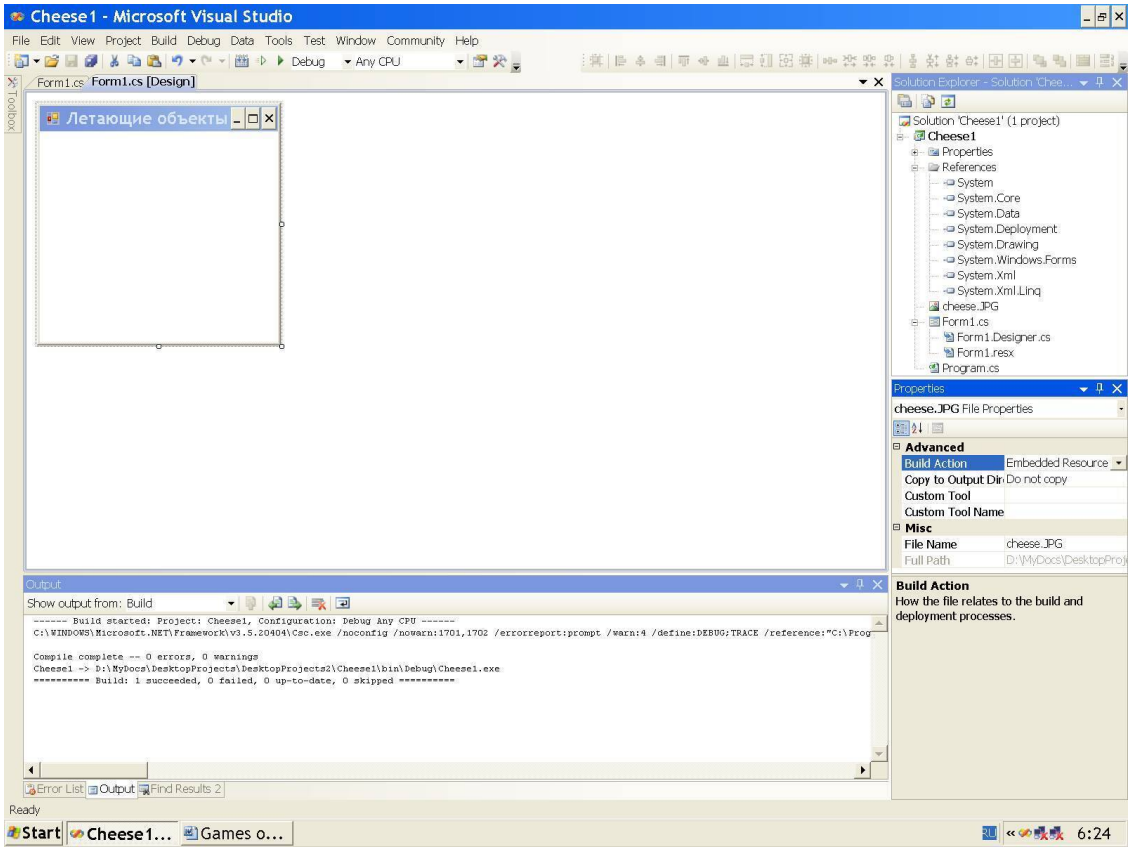


Рис. 4.1. Форма. **Рис. 4.2.** Файл рисунка в SE (слева) и в Properties (справа).

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.cs записываем пространство имён System.Reflection для управления классом Assembly:

using System.Reflection; //For the Assembly class.

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 ранее.

Листинг 4.1. Метод для построения изображения.

```
//We declare the object of class System.Drawing.Image
//for a product:
Image cheeseImage;
```

```
//We load into the project the image files according
//to such scheme:
//We create an object myAssembly of the Assembly class
//and appropriate to it
//the link to the executed assembly of our application:
static Assembly myAssembly = Assembly.GetExecutingAssembly();
//We create the myAssemblyName object of
//the System.Reflection.AssemblyName class and appropriate to
//it the assembly name, which consists of a project name,
//Version, Culture, PublicKeyToken:
static AssemblyName myAssemblyName = myAssembly.GetName();
//From the assembly name by means of the Name property
//we select a string project name:
static string myName_of_project = myAssemblyName.Name;
private void Form1_Paint(object sender, PaintEventArgs e)
{
//We load into object of the System.Drawing.Image class
//the image file of the set format, added to the project,
//by means of ResourceStream:
cheeseImage =
new Bitmap(myAssembly.GetManifestResourceStream(
myName_of_project + "." + "cheese.JPG"));
//We draw the image on the Form1:
e.Graphics.DrawImage(cheeseImage, 10, 20);
}
```

Строим и запускаем программу на выполнение обычным образом:

Build, Build Selection; Debug, Start Without Debugging.

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.

```
//Current abscissa of an object:
int cx = 50;
//Current ordinate of an object:
int cy = 100;

private void updatePositions()
{
```

```
cx++; //or cx = cx + 1;
cy++; //or cy = cy + 1;
}
```

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 булево 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 булево значение False.

Кроме того, таймер не должен быть включенным, пока программа не загрузит изображение. Поэтому в приведённый выше метод Form1_Paint дописываем в самом низу:

```
//We turn on the timer:
timer1.Enabled = true;
```

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

Листинг 4.3. Метод для рисования изображения.

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
//We load into object of the System.Drawing.Image class
//the image file of the set format, added to the project,
//by means of ResourceStream:
cheeseImage =
new Bitmap(myAssembly.GetManifestResourceStream(
myName_of_project + "." + "cheese.JPG"));
//We draw the image on the Form1:
e.Graphics.DrawImage(cheeseImage, cx, cy);
//We turn on the timer:
timer1.Enabled = true;
}
```

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.

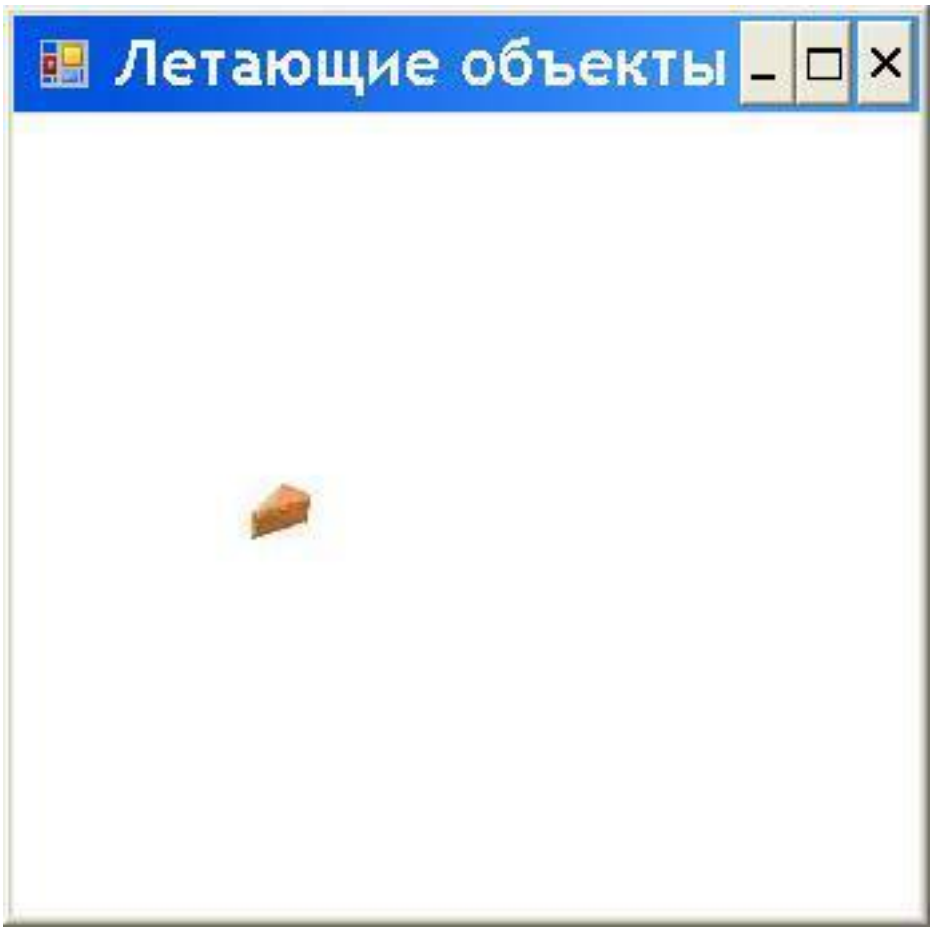
Листинг 4.4. Метод для смены кадров на экране и перемещения фигуры.

```
private void timer1_Tick(object sender, EventArgs e)
{
//We call the method:
updatePositions();
//We redraw the screen:
Invalidate();
}
```

Строим и запускаем программу на выполнение обычным образом:

Build, Build Selection; Debug, Start Without Debugging.

В ответ Visual C# выводит форму Form1 в режиме выполнения, на которой изображение типа встроенного нами рисунка сыра cheese.jpg перемещается из верхнего левого угла по диагонали сверху вниз (в нижний правый угол) и скрывается (рис. 4.3).



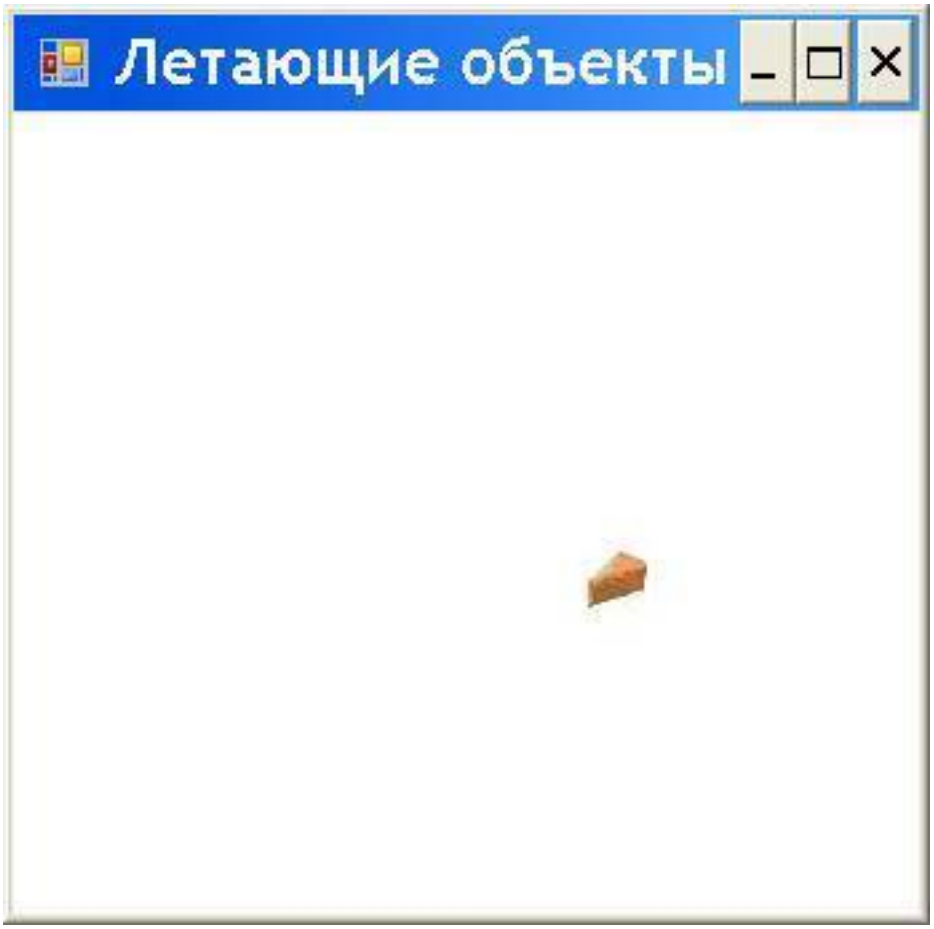


Рис. 4.3. Объект перемещается по диагонали сверху вниз. **Рис. 4.4.** Отскок объекта.

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 .

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 полёте 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 таким образом:

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.

```
//Movement on an axis "x" to the right:
bool goingRight = true;
//Movement on an axis of "y" to the down:
bool goingDown = true;
private void updatePositions()
{
if (goingRight)
{
cx++;
}
else
{
cx–;
}
if ((cx + cheeseImage.Width) >= this.ClientSize.Width)
{
goingRight = false;
}
if (cx <= 0)
{
goingRight = true;
```

```
}
if (goingDown)
{
cy++;
}
else
{
cy–;
}
if ((cy + cheeseImage.Height) >= this.ClientSize.Height)
{
goingDown = false;
}
if (cy <= 0)
{
goingDown = true;
}
}
```

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 this.ClientSize.Width и this.ClientSize.Height). Вследствие этого программа будет нормально работать для любых размеров объекта и 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)
{
cx += xSpeed;
}
else
{
cx -= xSpeed;
}
```

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)
{
cy += ySpeed;
}
else
{
cy -= ySpeed;
}
```

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

```
private void changeSpeed(int change)
{
xSpeed += change;
ySpeed += change;
}
```

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 выше методике использования в нашем приложении 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. Таким образом, в данном проекте приведённый выше метод updatePositions заменяем на следующий.

Листинг 4.6. Отскок объекта от границ.

```
//The current increment of movement on an axis "x":
int xSpeed = 1;
//The current increment of movement on an axis "y":
int ySpeed = 1;
//The method for increase in traverse speed:
private void changeSpeed(int change)
{
xSpeed += change;
ySpeed += change;
}
//The method for change of coordinates of an object:
private void updatePositions()
{
if (goingRight)
```

```
{
cx += xSpeed;
}
else
{
cx -= xSpeed;
}
if ((cx + cheeseImage.Width) >= this.ClientSize.Width)
{
goingRight = false;

//At time of collision, the sound signal Beep is given:
Microsoft.VisualBasic.Interaction.Beep();
}
if (cx <= 0)
{
goingRight = true;
//At time of collision, the sound signal Beep is given:
Microsoft.VisualBasic.Interaction.Beep();
}
if (goingDown)
{
cy += ySpeed;
}
else
{
cy -= ySpeed;
}
if ((cy + cheeseImage.Height) >= this.ClientSize.Height)
{
goingDown = false;

//At time of collision, the sound signal Beep is given:
Microsoft.VisualBasic.Interaction.Beep();
}
if (cy <= 0)
{
goingDown = true;

//At time of collision, the sound signal Beep is given:
Microsoft.VisualBasic.Interaction.Beep();
}
}
```

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 Propertiesc 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 void button1_Click(object sender, EventArgs e)
{
changeSpeed(1);
}
```

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 void button2_Click(object sender, EventArgs e)
{
changeSpeed(-1);
}
```

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 при-

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

Листинг 4.9. Метод для рисования изображения.

```
//We declare the object of class System.Drawing.Image
//for the subject:
Image breadImage; // = null by default.
//Current abscissa of a subject:
int bx = 0;
//Current ordinate of a subject:
int by = 0;
private void Form1_Paint(object sender, PaintEventArgs e)
{
//We load into the object of class System.Drawing.Image
//the image file of the set format, added to the project,
//by means of the ResourceStream:
cheeseImage =
new Bitmap(myAssembly.GetManifestResourceStream(
myName_of_project + "." + "cheese.JPG"));
breadImage =
new Bitmap(myAssembly.GetManifestResourceStream(
myName_of_project + "." + "bread.JPG"));
//We draw the images on the Form1:
e.Graphics.DrawImage(cheeseImage, cx, cy);
e.Graphics.DrawImage(breadImage, bx, by);
//We turn on the timer:
timer1.Enabled = true;
}
```

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.


Летающие объекты
Быстрее
Медленнее

Рис. 4.7. Подвижный сыр и неподвижный хлеб. **Рис. 4.8.** Сыр закрывает хлеб.

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).

Листинг 4.10. Метод для рисования изображения.

```
//Buffer in the view of object of class Bitmap:
Bitmap backBuffer = null;
private void Form1_Paint(object sender, PaintEventArgs e)
{
//We load into object of class System.Drawing.Image
//the image file of the set format, added to the project,
//by means of ResourceStream:
cheeseImage =
new Bitmap(myAssembly.GetManifestResourceStream(
myName_of_project + "." + "cheese.JPG"));
breadImage =
new Bitmap(myAssembly.GetManifestResourceStream(
myName_of_project + "." + "bread.JPG"));
//If it is necessary, we create the new buffer:
if (backBuffer == null)
{
backBuffer = new Bitmap(this.ClientSize.Width,
this.ClientSize.Height);
}
//We create the object of class Graphics from the buffer:
using (Graphics g = Graphics.FromImage(backBuffer))
{
//We clear the form:
g.Clear(Color.White);
//We draw the image in the backBuffer buffer:
g.DrawImage(breadImage, bx, by);
g.DrawImage(cheeseImage, cx, cy);
}
//We draw the image on the Form1:
e.Graphics.DrawImage(backBuffer, 0, 0);

//We turn on the timer:
timer1.Enabled = true;
} //End of the method Form1_Paint.
```

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 override void OnPaintBackground(
System.Windows.Forms.PaintEventArgs e)
{
//We prohibit to redraw a background.
}
```

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 размещёнием кнопок, для формы Form1 в панели Properties увеличиваем её размеры Size, например, до 384; 473.

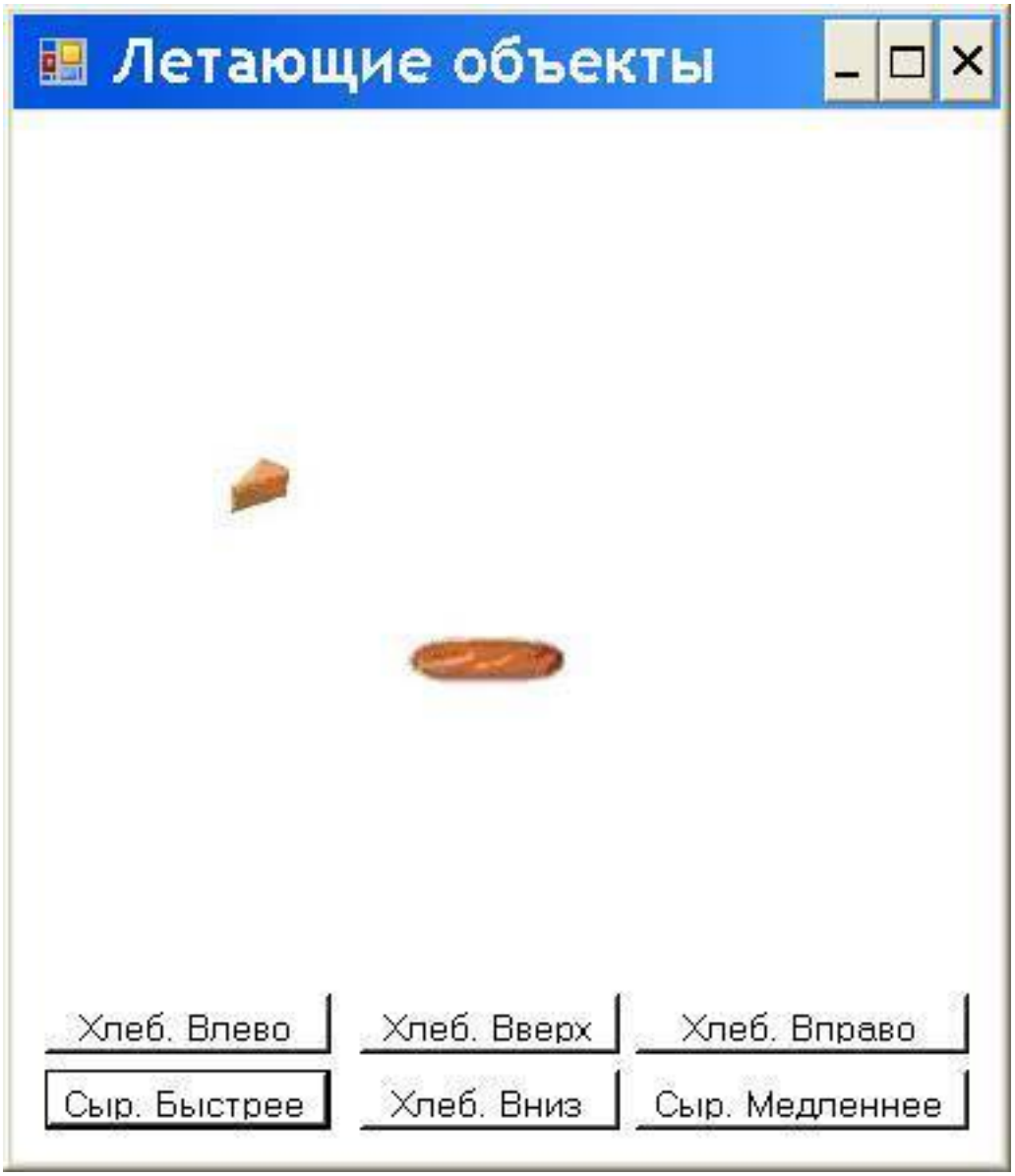

Летающие объекты
Хлеб. Влево
Хлеб. Вверх
Хлеб. Вправо
Сыр. Быстрее
Хлеб. Вниз
Сыр. Медленнее

Рис. 4.9. Подвижный сыр и управляемый нами хлеб. **Рис. 4.10.** Сыр закрывает хлеб.

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 следующий вид.

Листинг 4.12. Методы для обработки событий.

```
private void button3_Click(object sender, EventArgs e)
{
//We move an object up:
by -= ySpeed;
}
private void button4_Click(object sender, EventArgs e)
{
//We move an object down:
by += ySpeed;
}
private void button5_Click(object sender, EventArgs e)
{
//We move an object to the left:
bx -= xSpeed;
}
private void button6_Click(object sender, EventArgs e)
{
//We move an object to the right:
bx += xSpeed;
}
private void Form1_MouseMove(object sender, MouseEventArgs e)
{
//We determine the coordinates of mouse pointer on form:
int mouseX = e.X;
int mouseY = e.Y;
//We set the coordinates of bread, equal to coordinates
//of mouse:
bx = mouseX;
by = mouseY;
}
```

Для удобства, задаём другие значения начальным координатам хлеба (например, int bx = 150; int by = 200;)

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).

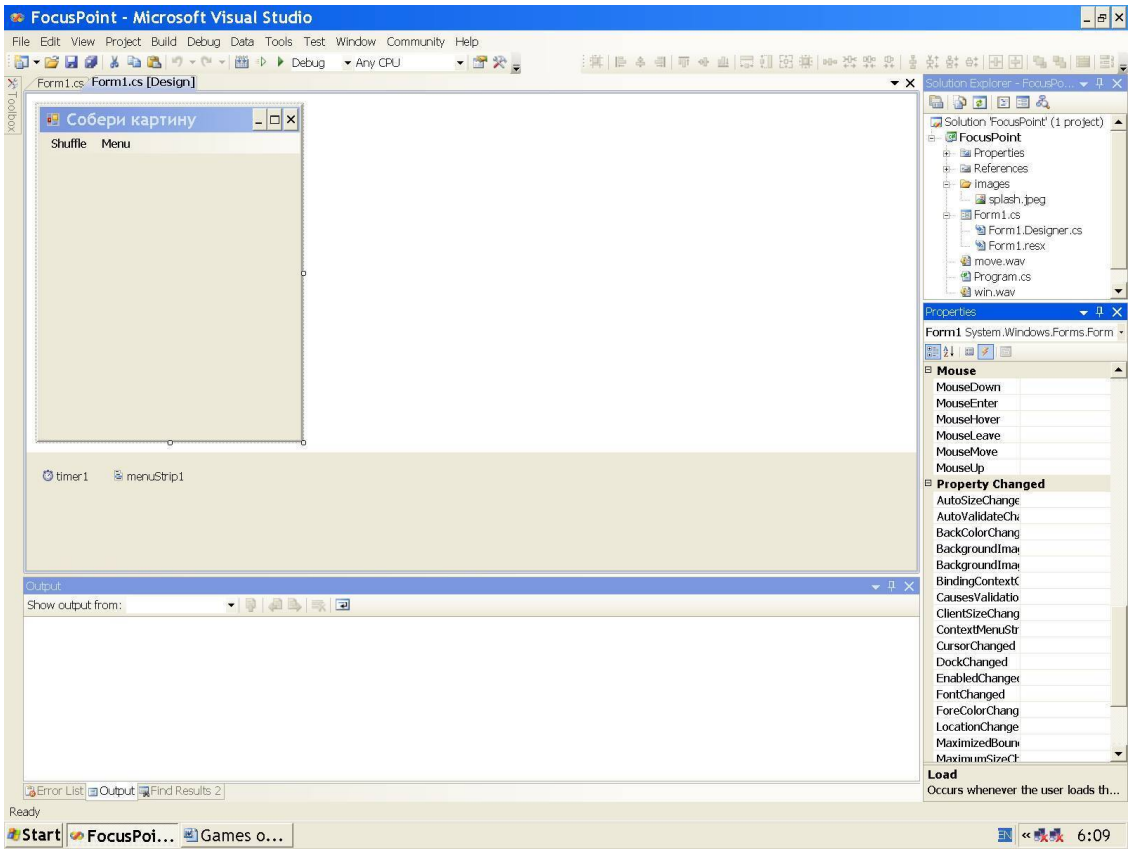


Рис. 4.11. События для управления игрой.

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.

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.

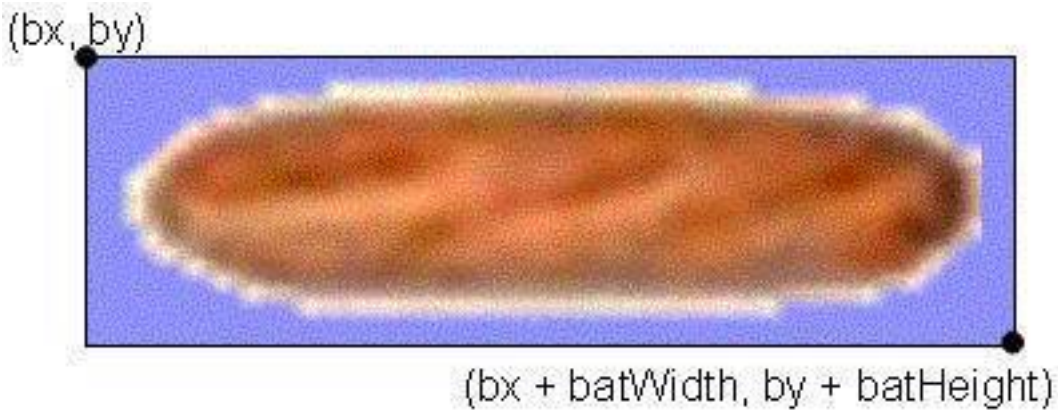


Рис. 5.1. Прямоугольник, описанный вокруг хлеба.

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:

```
//The rectangle, described around the first object:
Rectangle cheeseRectangle;
//The rectangle, described around the second object:
Rectangle breadRectangle;
```

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.

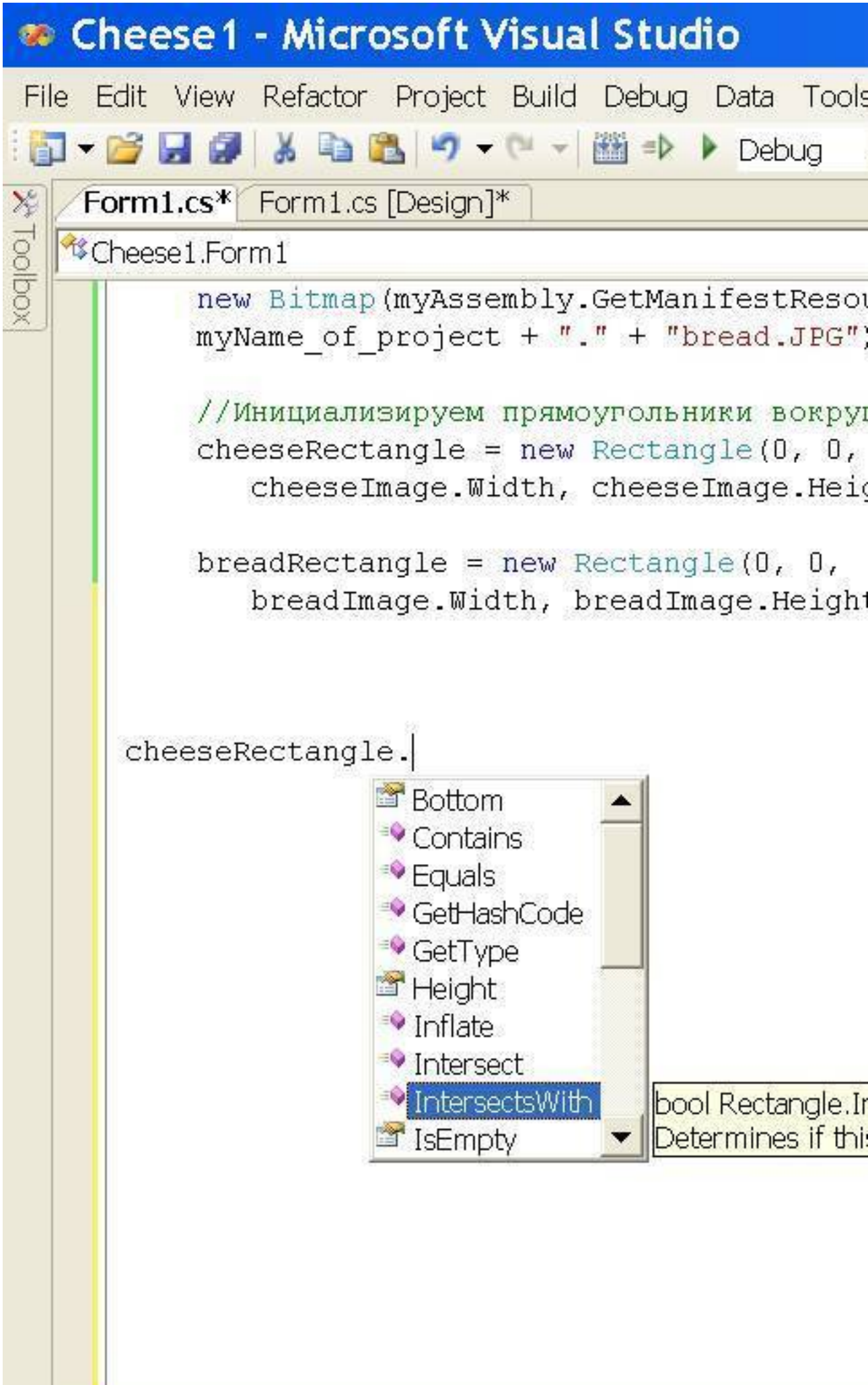

Cheese1 - Microsoft Visual Studio
File Edit View Refactor Project Build Debug Data Tools
Debug
Form1.cs* Form1.cs [Design]*
Toolbox
Cheese1.Form1
new Bitmap(myAssembly.GetManifestResou
myName_of_project + "." + "bread.JPG"
//Инициализируем прямоугольники вокру
cheeseRectangle = new Rectangle(0, 0,
cheeseImage.Width, 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 (Rectangle *rect*).

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

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

```
//We check the collision of objects:
if (cheeseRectangle.IntersectsWith(breadRectangle))
{
//We change the direction of the movement to opposite:
goingDown = !goingDown;
//At the time of collision, we give a sound signal Beep:
Microsoft.VisualBasic.Interaction.Beep();
}
```

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.

```
//The rectangle, described around the first object:
Rectangle cheeseRectangle;
//The rectangle, described around the second object:
Rectangle breadRectangle;
private void Form1_Paint(object sender, PaintEventArgs e)
{
//We load into objects of class System.Drawing.Image
//the image files of the set format, added to the project
//by means of ResourceStream:
cheeseImage =
new Bitmap(myAssembly.GetManifestResourceStream(
myName_of_project + "." + "cheese.JPG"));
breadImage =
new Bitmap(myAssembly.GetManifestResourceStream(
myName_of_project + "." + "bread.JPG"));
//We initialize the rectangles, described around objects:
cheeseRectangle = new Rectangle(cx, cy,
cheeseImage.Width, cheeseImage.Height);
breadRectangle = new Rectangle(bx, by,
breadImage.Width, breadImage.Height);
//If it is necessary, we create the new buffer:
if (backBuffer == null)
{
backBuffer = new Bitmap(this.ClientSize.Width,
this.ClientSize.Height);
}
//We createobject of the Graphics class from the buffer:
using (Graphics g = Graphics.FromImage(backBuffer))
{
//We clear the form:
g.Clear(Color.White);
//We draw the image in backBuffer:
g.DrawImage(cheeseImage, cx, cy);
g.DrawImage(breadImage, bx, by);
}
//We draw the image on Form1:
e.Graphics.DrawImage(backBuffer, 0, 0);
//We turn on the timer:
timer1.Enabled = true;
} //End of the method Form1_Paint.
```

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.

```
private void updatePositions()
{
if (goingRight)
{
cx += xSpeed;
}
else
{
cx -= xSpeed;
}
if ((cx + cheeseImage.Width) >= this.Width)
{
goingRight = false;
//At the time of collision,
//the sound signal Beep is given:
Microsoft.VisualBasic.Interaction.Beep();
}
if (cx <= 0)
{
goingRight = true;
//At the time of collision,
//the sound signal Beep is given:
Microsoft.VisualBasic.Interaction.Beep();
}
if (goingDown)
{
cy += ySpeed;
}
else
{
cy -= ySpeed;
}
//That cheese did not come for the button3.Location.Y:
if ((cy + cheeseImage.Height) >= button3.Location.Y)
{
goingDown = false;
//At the time of collision,
//the sound signal Beep is given:
Microsoft.VisualBasic.Interaction.Beep();
}
if (cy <= 0)
{
goingDown = true;
//At the time of collision,
//the sound signal Beep is given:
Microsoft.VisualBasic.Interaction.Beep();
```

```
}
//We set to rectangles of coordinates of objects:
cheeseRectangle.X = cx;
cheeseRectangle.Y = cy;
breadRectangle.X = bx;
breadRectangle.Y = by;
//We check the collision of objects:
if (cheeseRectangle.IntersectsWith(breadRectangle))
{
//We change the direction of the movement to opposite:
goingDown = !goingDown;
//At the time of collision,
//the sound signal Beep is given:
Microsoft.VisualBasic.Interaction.Beep();
}
} //End of the updatePositions method.
```

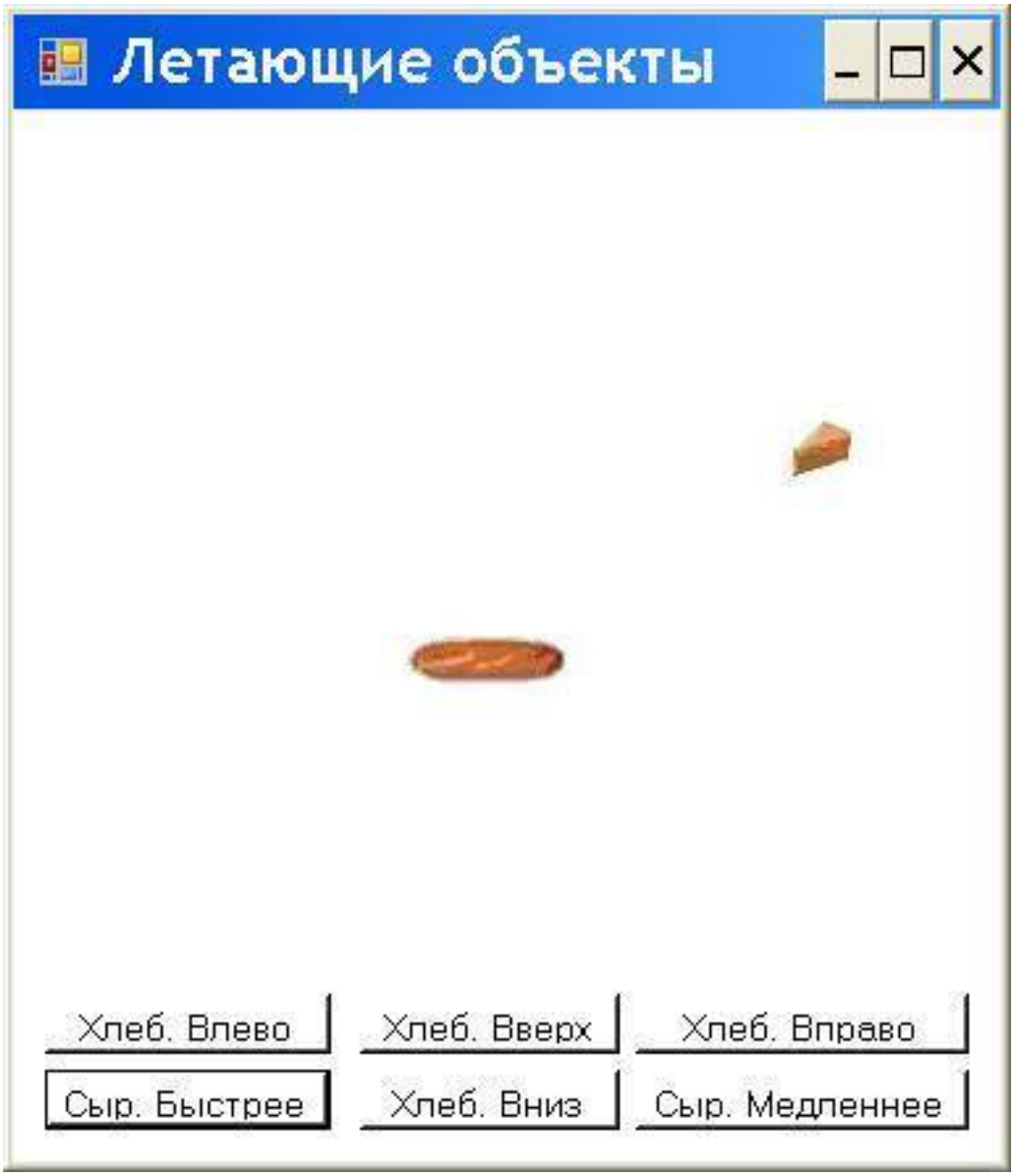
В режиме выполнения (Build, Build Selection; Debug, Start Without Debugging) при помощи кнопок и мыши мы можем перемещать хлеб и этим хлебом, как ракеткой, отбивать сыр или вверх, или вниз (рис. 5.3). Напомним, что, так как угол падения сыра на хлеб равен 45 градусам, то и угол отражения сыра от хлеба (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.


Летающие объекты
Хлеб. Влево
Хлеб. Вверх
Хлеб. Вправо
Сыр. Быстрее
Хлеб. Вниз
Сыр. Медленнее

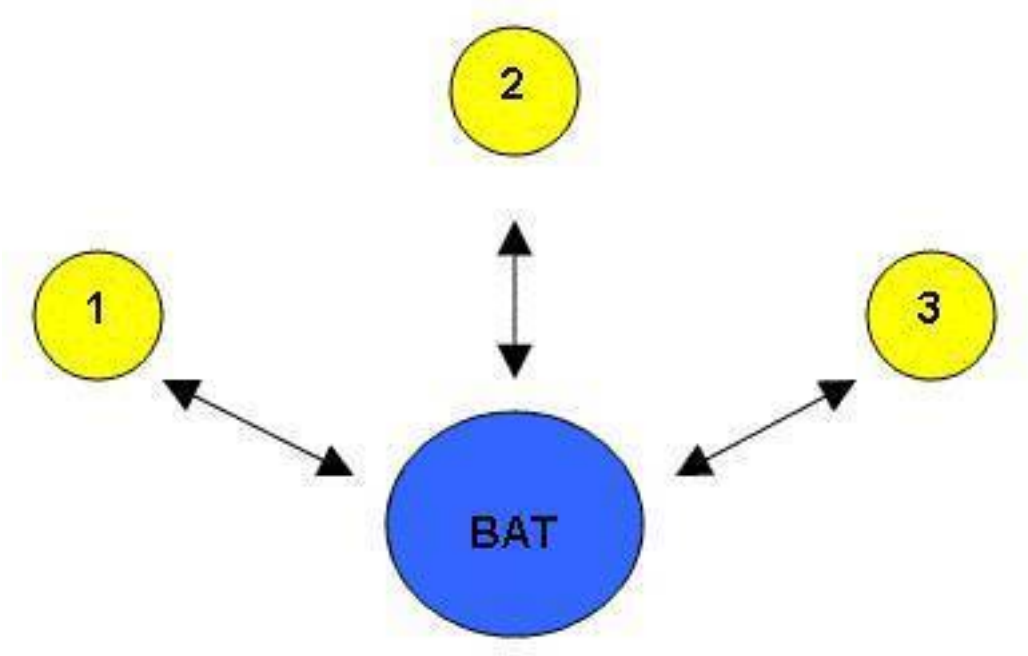


Рис. 5.3. Сыр отскочил от хлеба. **Рис. 5.4.** Три схемы столкновений.

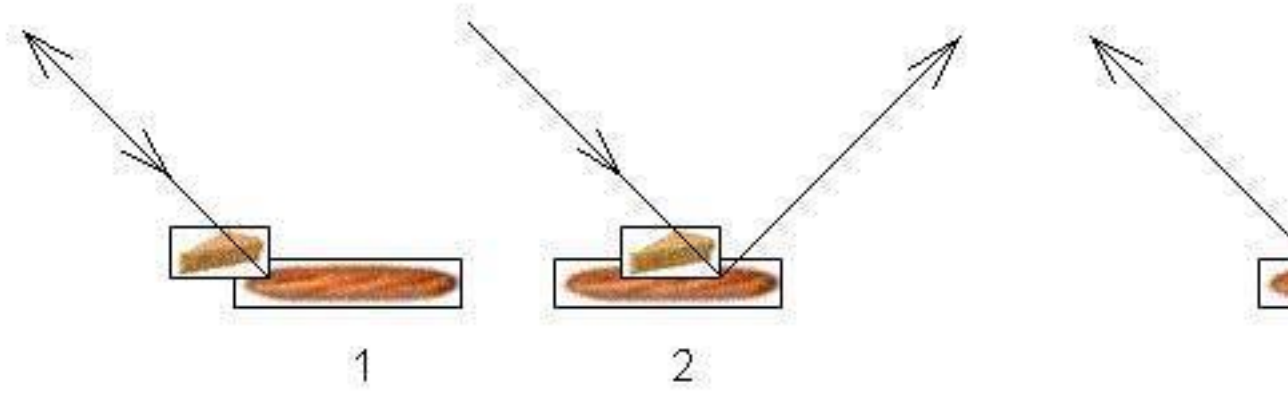


Рис. 5.5. Четыре схемы столкновений.

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.

```
private void updatePositions()
{
if (goingRight)
{
cx += xSpeed;
}
else
{
```

```
cx -= xSpeed;
}
if ((cx + cheeseImage.Width) >= this.Width)
{
goingRight = false;
//At the time of collision, the signal Beep is given:
Microsoft.VisualBasic.Interaction.Beep();
}
if (cx <= 0)
{
goingRight = true;
//At the time of collision, the signal Beep is given:
Microsoft.VisualBasic.Interaction.Beep();
}
if (goingDown)
{
cy += ySpeed;
}
else
{
cy -= ySpeed;
}
//That cheese did not come for the button3.Location.Y:
if ((cy + cheeseImage.Height) >= button3.Location.Y)
{
goingDown = false;
//At the time of collision, the signal Beep is given:
Microsoft.VisualBasic.Interaction.Beep();
}
if (cy <= 0)
{
goingDown = true;
//At the time of collision, the signal Beep is given:
Microsoft.VisualBasic.Interaction.Beep();
}
//We check the collision of objects:
if (goingDown)
{
//If cheese moves down and there is the collision:
if (cheeseRectangle.IntersectsWith(breadRectangle))
{
//At the time of collision, the signal Beep
//is given:
Microsoft.VisualBasic.Interaction.Beep();
//We have the collision:
bool rightIn = breadRectangle.Contains(
cheeseRectangle.Right,
cheeseRectangle.Bottom);
bool leftIn = breadRectangle.Contains(
```

```
cheeseRectangle.Left,
cheeseRectangle.Bottom);
//types of collisions:
if (rightIn & leftIn)
{
//bounce up:
goingDown = false;
}
else
{
//bounce up:
goingDown = false;
//the bounces in horizontal direction:
if (rightIn)
{
goingRight = false;
}
if (leftIn)
{
goingRight = true;
}
}
}
}
} //End of the method updatePositions.
```

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).

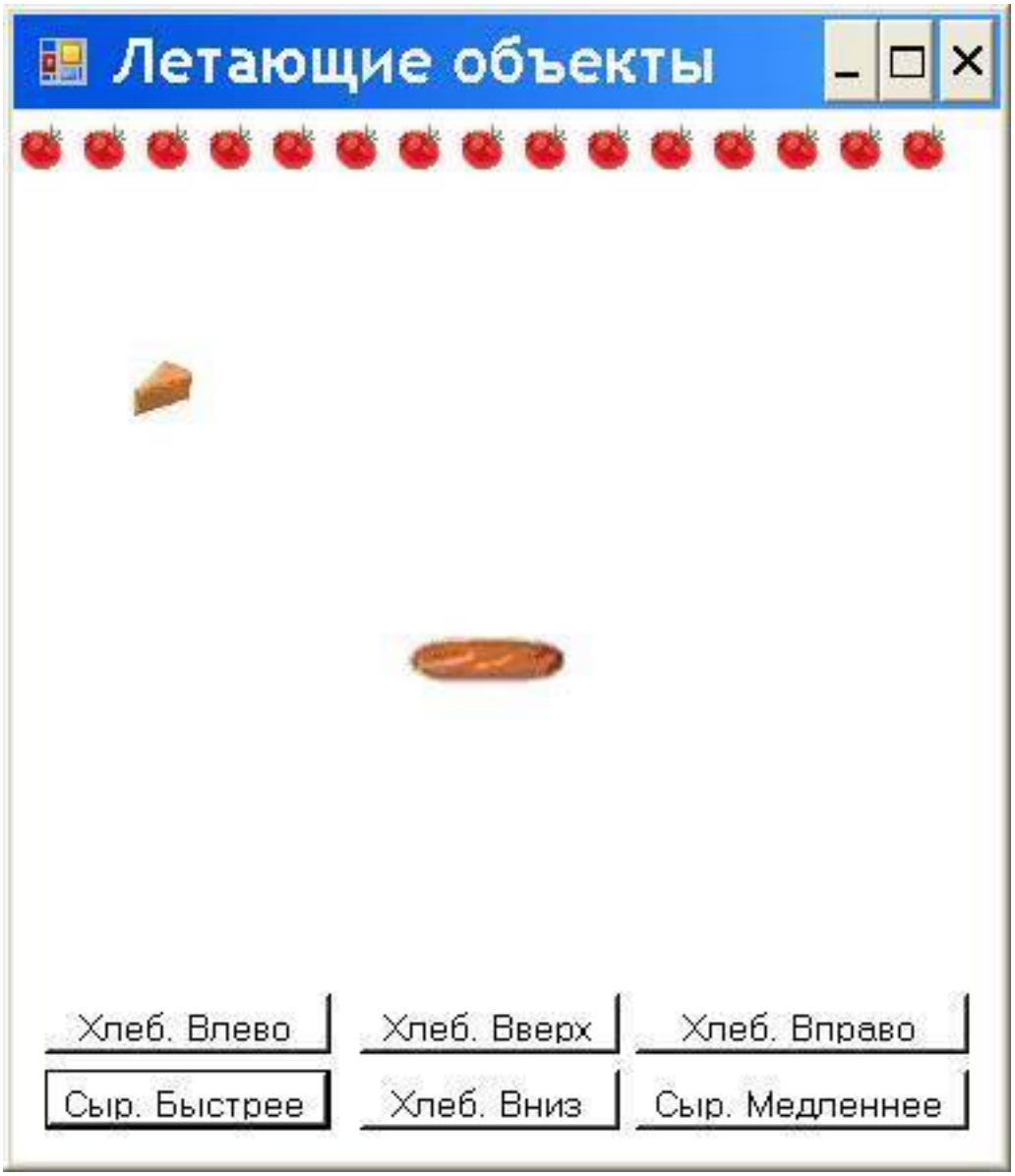
Попадание сыра в помидор определяется уже применяемым выше методом IntersectWith.

Исчезновение помидоров выполняется при помощи свойства visible, которому присваивается булево 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 построении) вместо заданного по умолчанию выбираем значение Embedded Resource (Встроенный ресурс).



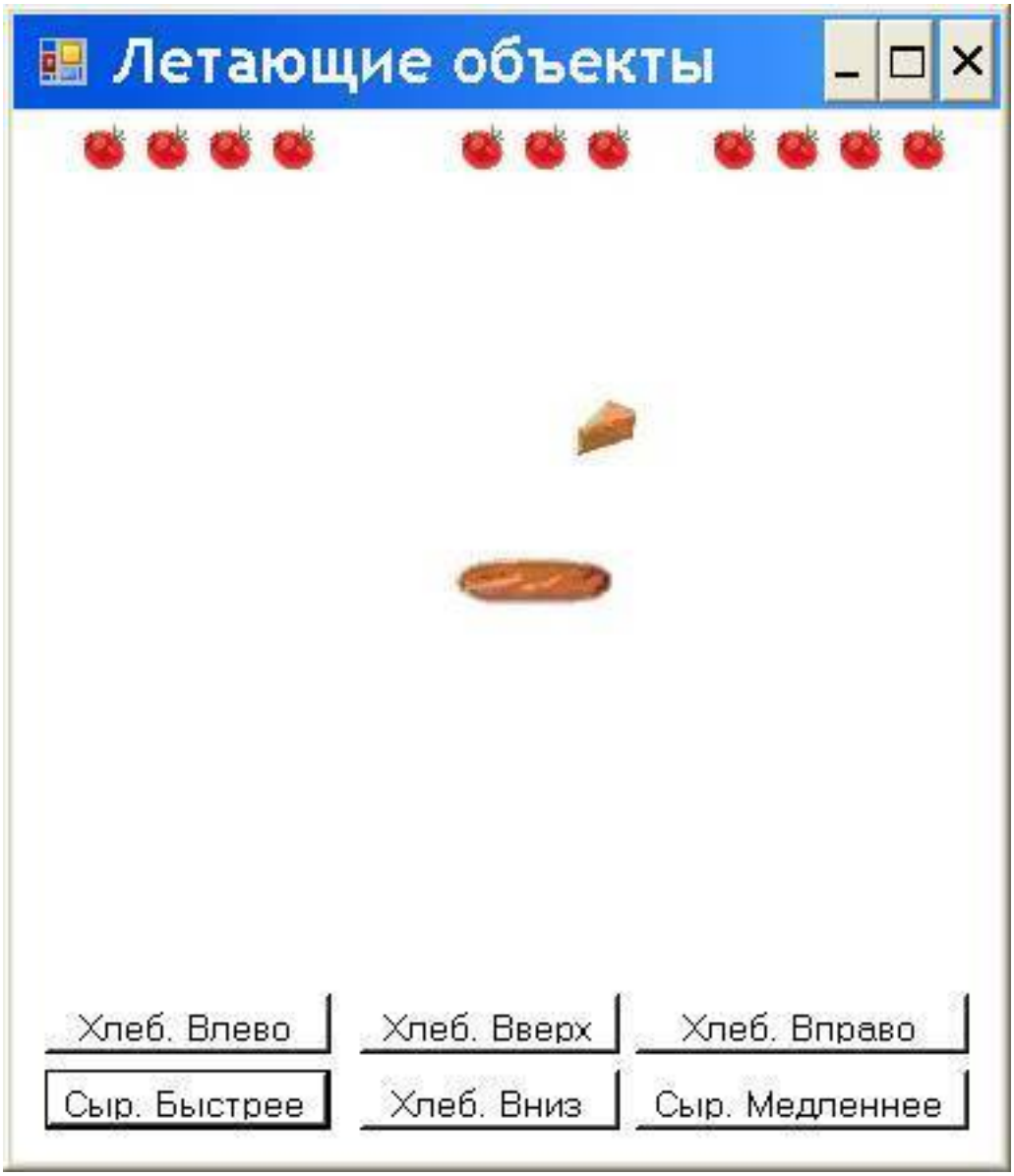


Рис. 5.7. Помидоры – мишени. **Рис. 5.8.** Помидоры исчезают после попадания в них сыра.

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).

```
//We declare the object of the System.Drawing.Image class
//for product:
Image tomatoImage;
//Position and state of tomato
struct tomato
{
public Rectangle rectangle;
public bool visible;
}
// Spacing between tomatoes. Set once for the game
int tomatoSpacing = 4;
```

```
// Height, at which the tomatoes are drawn. Will change
// as the game progresses. Starts at the top.
int tomatoDrawHeight = 4;
// The number of tomatoes on the screen. Set at the start
// of the game by initialiseTomatoes.
int noOfTomatoes;
// Positions of the tomato targets.
tomato[] tomatoes;
// called once to set up all the tomatoes.
void initialiseTomatoes()
{
noOfTomatoes = (this.ClientSize.Width – tomatoSpacing) /
(tomatoImage.Width + tomatoSpacing);
// create an array to hold the tomato positions
tomatoes = new tomato[noOfTomatoes];
// x coordinate of each potato
int tomatoX = tomatoSpacing / 2;
for (int i = 0; i < tomatoes.Length; i++)
{
tomatoes[i].rectangle =
new Rectangle(tomatoX, tomatoDrawHeight,
tomatoImage.Width, tomatoImage.Height);
tomatoX = tomatoX + tomatoImage.Width + tomatoSpacing;
}
}
// Called to place a row of tomatoes.
private void placeTomatoes()
{
for (int i = 0; i < tomatoes.Length; i++)
{
tomatoes[i].rectangle.Y = tomatoDrawHeight;
tomatoes[i].visible = true;
}
}
```

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

Листинг 5.5. Метод для рисования изображения.

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
//If it is necessary, we create the new buffer:
if (backBuffer == null)
{
backBuffer = new Bitmap(this.ClientSize.Width,
this.ClientSize.Height);
}
//We create a object of the Graphics class from the buffer:
using (Graphics g = Graphics.FromImage(backBuffer))
{
//We clear the form:
```

```
g.Clear(Color.White);
//We draw the image in the backBuffer:
g.DrawImage(cheeseImage, cx, cy);
g.DrawImage(breadImage, bx, by);
for (int i = 0; i < tomatoes.Length; i++)
{
if (tomatoes[i].visible)
{
g.DrawImage(tomatoImage,
tomatoes[i].rectangle.X,
tomatoes[i].rectangle.Y);
}
}
}
//We draw the image on the Form1:
e.Graphics.DrawImage(backBuffer, 0, 0);
} //End of the method Form1_Paint.
```

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 void Form1_Load(object sender, EventArgs e)
{
//We load into objects of class System.Drawing.Image
//the image files of the set format, added to the project,
//by means of ResourceStream:
cheeseImage =
new Bitmap(myAssembly.GetManifestResourceStream(
myName_of_project + "." + "cheese.JPG"));
breadImage =
new Bitmap(myAssembly.GetManifestResourceStream(
myName_of_project + "." + "bread.JPG"));
//We initialize the rectangles, described around objects:
cheeseRectangle = new Rectangle(cx, cy,
cheeseImage.Width, cheeseImage.Height);
breadRectangle = new Rectangle(bx, by,
breadImage.Width, breadImage.Height);
//We load the tomato:
tomatoImage =
new Bitmap(myAssembly.GetManifestResourceStream(
myName_of_project + "." + "tomato.gif"));
//We initialize an array of tomatoes and rectangles:
initialiseTomatoes();
//We place the tomatoes in an upper part of the screen:
placeTomatoes();
//We turn on the timer:
timer1.Enabled = true;
}
```

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.

```
private void updatePositions()
{
if (goingRight)
{
cx += xSpeed;
}
else
{
cx -= xSpeed;
}
if ((cx + cheeseImage.Width) >= this.Width)
{
goingRight = false;
//At the time of collision, the Beep signal is given:
Microsoft.VisualBasic.Interaction.Beep();
}
if (cx <= 0)
{
goingRight = true;
//At the time of collision, the Beep signal is given:
Microsoft.VisualBasic.Interaction.Beep();
}
if (goingDown)
{
cy += ySpeed;
}
else
{
cy -= ySpeed;
}
//That the cheese did not come for the button3.Location.Y:
if ((cy + cheeseImage.Height) >= button3.Location.Y)
{
goingDown = false;
//At the time of collision, the Beep signal is given:
Microsoft.VisualBasic.Interaction.Beep();
}
if (cy <= 0)
{
goingDown = true;
//At the time of collision, the Beep signal is given:
Microsoft.VisualBasic.Interaction.Beep();
}
//We set to rectangles of coordinate of objects:
cheeseRectangle.X = cx;
```

```
cheeseRectangle.Y = cy;
breadRectangle.X = bx;
breadRectangle.Y = by;
//We check the collision of objects
//taking into account the tomatoes:
if (goingDown)
{
// only bounce if the cheese is going down
if (cheeseRectangle.IntersectsWith(breadRectangle))
{
//At the time of collision,
//the Beep signal is given:
Microsoft.VisualBasic.Interaction.Beep();
// we have a collision
bool rightIn = breadRectangle.Contains(
cheeseRectangle.Right,
cheeseRectangle.Bottom);
bool leftIn = breadRectangle.Contains(
cheeseRectangle.Left,
cheeseRectangle.Bottom);
// now deal with the bounce
if (rightIn & leftIn)
{
// bounce up
goingDown = false;
}
else
{
// bounce up
goingDown = false;
// now sort out horizontal bounce
if (rightIn)
{
goingRight = false;
}
if (leftIn)
{
goingRight = true;
}
}
}
}
else
{
// only destroy tomatoes of the cheese is going up
for (int i = 0; i < tomatoes.Length; i++)
{
if (!tomatoes[i].visible)
{
```

```
continue;
}
if (cheeseRectangle.IntersectsWith(
tomatoes[i].rectangle))
{
//At the time of collision,
//the Beep signal is given:
Microsoft.VisualBasic.Interaction.Beep();
// hide the tomato
tomatoes[i].visible = false;
// bounce down
goingDown = true;
// only destroy one at a time
break;
}
}
}
} //End of the method 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 высокие (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:

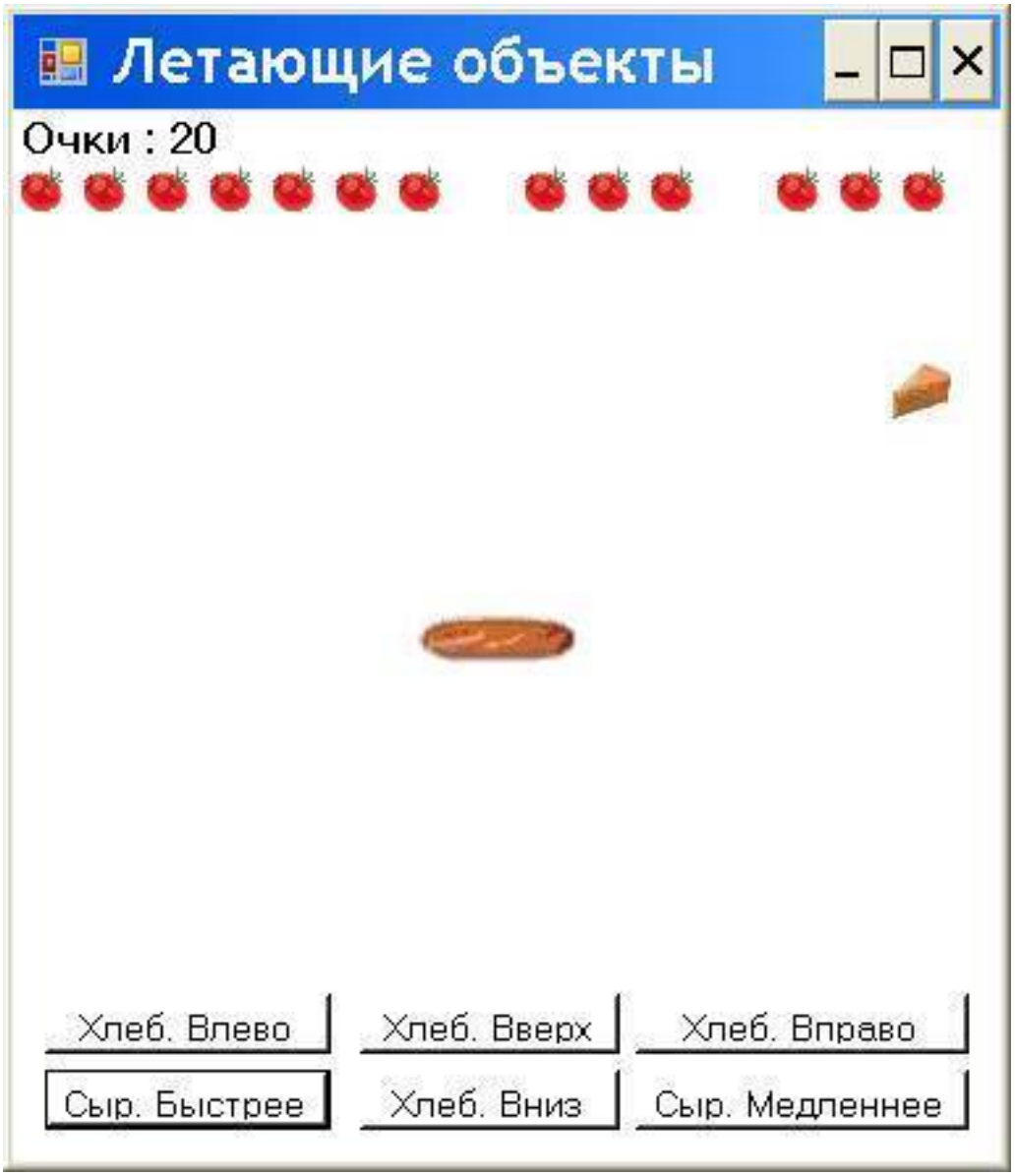
```
g.DrawString(messageString, messageFont, messageBrush,
messageRectangle);
```

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.


Летающие объекты
Очки : 20
Хлеб. Влево
Хлеб. Вверх
Хлеб. Вправо
Сыр. Быстрее
Хлеб. Вниз
Сыр. Медленнее

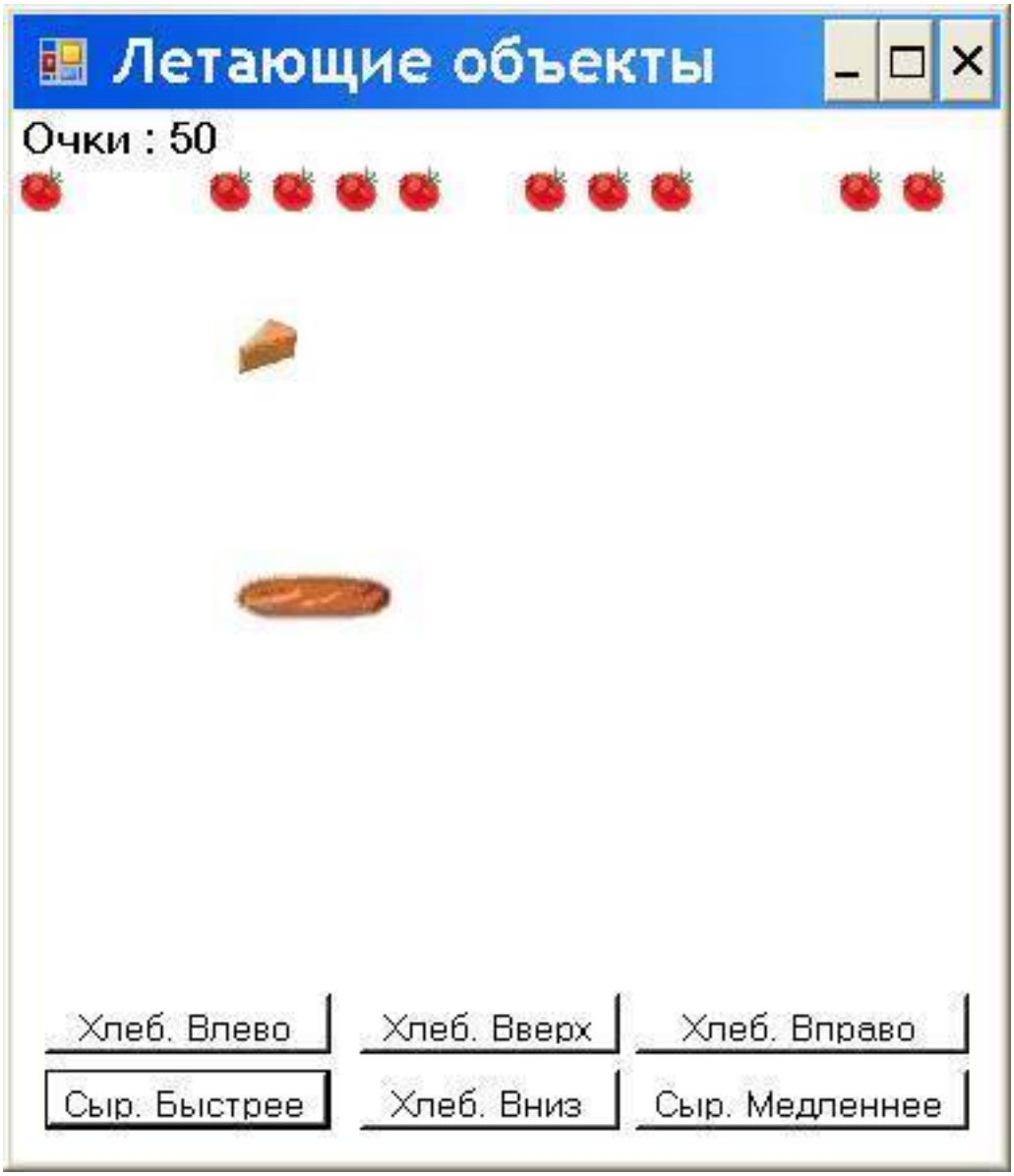


Рис. 5.9. Получили 20 очков за 2 сбитых помидора. **Рис. 5.10.** Получили 50 очков.

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:

```
int tomatoDrawHeight = 20;
```

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

Листинг 5.8. Новые переменные.

```
// Font for score messages.
Font messageFont = null;
// Rectangle for score display.
Rectangle messageRectangle;
// Height of the score panel.
int scoreHeight = 20;
// Brush used to draw the messages.
SolidBrush messageBrush;
```

```
// The string, which is drawn as the user message.
string messageString = "Score : 0";
// Score in a game.
int scoreValue = 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 void Form1_Paint(object sender, PaintEventArgs e)
{
//If the buffer empty, we create the new buffer:
if (backBuffer == null)
{
backBuffer = new Bitmap(this.ClientSize.Width,
this.ClientSize.Height);
}
//We create a object of class Graphics from the buffer:
using (Graphics g = Graphics.FromImage(backBuffer))
{
//We clear the form:
g.Clear(Color.White);
//We draw the images of objects in the backBuffer:
g.DrawImage(cheeseImage, cx, cy);
g.DrawImage(breadImage, bx, by);
for (int i = 0; i < tomatoes.Length; i++)
{
if (tomatoes[i].visible)
{
g.DrawImage(tomatoImage,
tomatoes[i].rectangle.X,
tomatoes[i].rectangle.Y);
}
}
//We write the player's points:
g.DrawString(messageString, messageFont, messageBrush,
messageRectangle);
}
//We draw the image on the Form1:
e.Graphics.DrawImage(backBuffer, 0, 0);
} //End of the method Form1_Paint.
```

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 void Form1_Load(object sender, EventArgs e)
{
//We load into objects of the System.Drawing.Image class
//the image files of the set format, added to the project,
//by means of ResourceStream:
cheeseImage =
new Bitmap(myAssembly.GetManifestResourceStream(
```

```
myName_of_project + "." + "cheese.JPG"));
breadImage =
new Bitmap(myAssembly.GetManifestResourceStream(
myName_of_project + "." + "bread.JPG"));
//We initialize the rectangles, described around objects:
cheeseRectangle = new Rectangle(cx, cy,
cheeseImage.Width, cheeseImage.Height);
breadRectangle = new Rectangle(bx, by,
breadImage.Width, breadImage.Height);
//We load the image file of a new object:
tomatoImage =
new Bitmap(myAssembly.GetManifestResourceStream(
myName_of_project + "." + "tomato.gif"));
//We initialize an array of new objects and rectangles,
//described around these objects:
initialiseTomatoes();
//We place new objects in an upper part of the screen:
placeTomatoes();
//We create and initialize a font for record of points:
messageFont = new Font(FontFamily.GenericSansSerif, 10,
FontStyle.Regular);
//We reserve a rectangle on the screen
//for record of points:
messageRectangle = new Rectangle(0, 0,
this.ClientSize.Width, scoreHeight);
//We set the color of a brush for record of points:
messageBrush = new SolidBrush(Color.Black);
//We turn on the timer:
timer1.Enabled = true;
} //End of the method Form1_Load.
```

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.

```
private void updatePositions()
{
if (goingRight)
{
cx += xSpeed;
}
else
{
cx -= xSpeed;
}
if ((cx + cheeseImage.Width) >= this.Width)
{
goingRight = false;
//At the time of collision, the Beep signal is given:
Microsoft.VisualBasic.Interaction.Beep();
```

```
}
if (cx <= 0)
{
goingRight = true;
//At the time of collision, the Beep signal is given:
Microsoft.VisualBasic.Interaction.Beep();
}
if (goingDown)
{
cy += ySpeed;
}
else
{
cy -= ySpeed;
}
//That cheese did not come for the button3.Location.Y:
if ((cy + cheeseImage.Height) >= button3.Location.Y)
{
goingDown = false;
//At the time of collision, the Beep signal is given:
Microsoft.VisualBasic.Interaction.Beep();
}
if (cy <= 0)
{
goingDown = true;
//At the time of collision, the Beep signal is given:
Microsoft.VisualBasic.Interaction.Beep();
}
//We set to rectangles of coordinate of objects:
cheeseRectangle.X = cx;
cheeseRectangle.Y = cy;
breadRectangle.X = bx;
breadRectangle.Y = by;
// check for collisions.
if (goingDown)
{
// only bounce if the cheese is going down
if (cheeseRectangle.IntersectsWith(breadRectangle))
{
//At the time of collision,
//the Beep signal is given:
Microsoft.VisualBasic.Interaction.Beep();
// we have a collision
bool rightIn = breadRectangle.Contains(
cheeseRectangle.Right,
cheeseRectangle.Bottom);
bool leftIn = breadRectangle.Contains(
cheeseRectangle.Left,
cheeseRectangle.Bottom);
```

```
// now deal with the bounce
if (rightIn & leftIn)
{
// bounce up
goingDown = false;
}
else
{
// bounce up
goingDown = false;
// now sort out horizontal bounce
if (rightIn)
{
goingRight = false;
}
if (leftIn)
{
goingRight = true;
}
}
}
}
else
{
// only destroy tomatoes of the cheese is going up
for (int i = 0; i < tomatoes.Length; i++)
{
if (!tomatoes[i].visible)
{
continue;
}
if (cheeseRectangle.IntersectsWith(
tomatoes[i].rectangle))
{
//At the time of collision,
//the Beep signal is given:
Microsoft.VisualBasic.Interaction.Beep();
// hide the tomato
tomatoes[i].visible = false;
// bounce down
goingDown = true;
// update the score
scoreValue = scoreValue + 10;
messageString = "Points : " + scoreValue;
// only destroy one at a time
```

Конец ознакомительного фрагмента.