Доктор технических наук
Валерий Алексеевич Жарков

Справочник Жаркова

по

проектированию и программированию
искусственного интеллекта

Том 3:

Программирование на Visual C# искусственного интеллекта (продолжение 2)

Валерий Алексеевич Жарков

Справочник Жаркова по проектированию и программированию искусственного интеллекта. Том 3: Программирование на Visual C# искусственного интеллекта (продолжение 2)

http://www.litres.ru/pages/biblio_book/?art=68341601
ISBN 9785005910844

Аннотация

В серии книг «Справочник Жаркова по проектированию и программированию искусственного интеллекта» собрано лучшее программирование искусственного интеллекта (ИИ) в двух- и трёхмерных играх и приложениях, разработанных как автором, так и накопленных за многие годы в Интернете, для настольных компьютеров, ноутбуков, планшетов, смартфонов и Xbox. Даны инструкции по программированию ИИ с текстами программ,

а также адреса в Интернете, где можно загрузить программы, графические и звуковые файлы.

Содержание

Справочник Жаркова по проектированию и программированию искусственного интеллекта Том 3: Программирование на Visual C# искусственного интеллекта (продолжение 2)

Валерий Алексеевич Жарков

ISBN 978-5-0059-1084-4 (т. 3)
ISBN 978-5-0056-6546-1
Создано в интеллектуальной издательской системе Ridero

Введение

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

Искусственный интеллект (ИИ, artificial intelligence, AI) – 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 коман-

ду на выполнение устройством определённого действия. При программировании ИИ важно правильно подобрать среду разработки ИИ и язык программирования.

Среда разработки (иначе, платформа, студия) Visual Studio (или коротко VS) для визуального объектно-ориентированного проектирования приложений даёт уникальную возможность быстро разрабатывать высокотехнологичные и наукоёмкие программные продукты с использованием ИИ, а также компьютерные игры с двухмерной и трёхмерной графикой также с использованием ИИ. Важно отметить, что на основе VS мы программируем не закрытые «чёрные ящики», как это делают другие известные компьютерные фирмы, а мы создаём открытые любому пользователю (для постоянного дополнения и совершенствования) программы на базе самых мощных в мире алгоритмических языков высокого уровня Visual C#, Visual Basic и Visual C++. В данном чрезвычайно насыщенном томе (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 и журналы с сайта ZharkovPress.ru приведены в нашем списке литературы) и там найдёт ответ, так как терминология по всем тематикам у нас общая. Мы будем давать лишь краткие пояснения в виде комментариев в программах, чтобы начинающий пользователь постепенно осваивал базовые дисциплины программирования ИИ на языке 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 и журналов из списка литературы и с сайта ZharkovPress.ru (а также используя справку Help из Visual Studio, как правило, заменяющей все учебники по всем языкам), читатель сможет в одиночку работать за конструктора, технолога, математика и программиста одновременно (при разработке практических приложений) или за сценариста, режиссёра, оператора, дизайнера, художника, музыкального редактора и программиста одновременно (при разработке игр) и сэкономить недели упорного труда. Если в начальных главах серии книг инструкции по разработке ИИ в играх и приложениях на базе VS подробны (в интересах новичков), то инструкции в каждой последующей главе мы даём всё короче и короче, чтобы не повторяться и экономить место в книге.

Приводим краткое содержание данного тома справочника.

Часть II. Методология программирования искусственного интеллекта в спортивных играх с ракетками и мячами.

Глава 2. Методика программирования искусственного интеллекта в игре в теннис для игрока с компьютером или двух игроков.

Глава 3. Вариант программирования искусственного интеллекта в игре в теннис для игрока с компьютером.

Глава 4. Методика программирования искусственного интеллекта в игре Minjie по зеркальному размещению шашек одного игрока по отношению к шашкам другого игрока.

Часть III. Методология программирования искусственного интеллекта в играх и приложениях, когда первый объект охотится за вторым, а второй – за третьим.

Глава 5. Методика программирования искусственного интеллекта на примере обнаружения одним объектом другого.

Глава 6. Методика программирования искусственного интеллекта в играх и приложениях, когда первый объект охотится за вторым, а второй – за третьим.

Глава 7. Методика программирования искусственного интеллекта в играх и приложениях, когда один большой объект охотится на одного большого объекта, составленного из нескольких маленьких объектов.

Глава 8. Методика программирования искусственного интеллекта в игре боя космического корабля с инопланетными

кораблями.

Глава 9. Статья: Программирование на C# искусственного интеллекта с несколькими агентами в игре в мини-баскетбол.

Часть IV. Развёртывание, публикация и распространение разработанной игры или приложения с искусственным интеллектом.

Глава 10. Методика распространения игры или приложения с искусственным интеллектом.

Заключение.

Список литературы.

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

мы именно им и характерны для их профессиональных или непрофессиональных интересов.

Книга предназначена для всех желающих быстро изучить основы проектирования и 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 и планшетов, на этих основах сразу же проектировать ИИ в сложных играх и приложениях и применять ИИ на практике или на отдыхе в разнообразных сферах профессиональной и непрофессиональной деятельности. Также адресована начинающим и опытным пользователям, программистам любой квалификации, а также учащимся и слушателям курсов, студентам, аспирантам, учителям, преподавателям и научным работникам.

В следующем томе автор (доктор технических наук Жарков Валерий Алексеевич) продолжит описывать программирование ИИ в следующих играх и приложениях.

Вопросы, замечания и предложения по тематике наших книг и журналов можно направлять по email: valeryzharkov@mtu-net.ru или с сайта: www.ZharkovPress.ru.

Автор: доктор технических наук Жарков Валерий Алексеевич (г. Москва).

Часть I. Методология программирования искусственного интеллекта в карточных играх

Глава 1. Методика разработки искусственного интеллекта в карточных играх на примере игры в 21 очко

1.1. Общие сведения

Разработаем методологию программирования искусственного интеллекта в карточных играх на примере типичной и широко распространённой игры в «очко» или в «21», следуя статье с сайта microsoft.com: Rob Miles. Pocket Jack: Writing a Card-Playing Application, но с нашими усовершенствованиями для современной версии Visual Studio.

Карточные игры широко распространены во всем мире, по многим из них официально проводятся спортивные соревнования различных уровней, вплоть до первенства мира. В США, России и других странах некоторые карточные игры также официально признаны как спортивные игры, и по ним также проводятся спортивные соревнования различных уровней, включая первенство мира.

Поэтому имеет смысл разработать ряд классов, методы которых обеспечивали бы управление широким диапазоном карточных игр. Эта глава описывает механизм сдачи карт (из колоды) случайным образом (на основе генератора случайных чисел), показ их на экране, показ их в «руках» для каждого игрока и управления карточной игрой. Мы разработаем полностью функциональную игру, которая в США и других странах называется как «Black Jack», «21» или pontoon при использовании 52-х карт, а в России и других странах обычно называется как «очко» или «21» при исполь-

зовании 36-ти карт. В разработанной далее в данной главе игре нетрудно будет вместо 52-х карт добавить в проект 36 карт (в случае необходимости).

Напомним смысл игры. В игре участвуют, как минимум, два игрока: игрок и банкомёт в виде компьютера, сдающего карты. Из колоды карт банкомёт (по-английски называется dealer – дилер) сдаёт по одной карте игроку, который должен набрать количество очков, как можно ближе к 21 или равное 21 (21 – это лучший вариант – очко), но не более 21. Если игрок набрал больше 21, то это обычно называется «перебор», и игрок считается проигравшим. Если же игрок набрал меньше 21, то он предлагает банкомёту набирать карты себе. Если банкомёт наберёт очков меньше, чем игрок, банкомёт считается проигравшим, а если больше, то победившим. При равном количестве очков в данной игре принимается, что победил тот, кто первым набрал эти очки, а именно, игрок (но можно счёт оставить прежним). Правила игры сформулируем далее.

1.2. Рисование карт на экране

Первая проблема, которую нужно решить, – показ на экране карт, которые два соревнующихся игрока, например, компьютер и игрок, держат в «руках». Чтобы это сделать, нам нужно: 52 изображения карт в виде файлов формата, например, (.gif), одно изображение фона игры 0.gif, одно изображения банка bank.jpg и одно изображение загрузки loading.gif. Напомним, что формат (.gif) расшифровывается как Graphics Interchange Format (Формат обмена графическими данными). Все изображения в уменьшенном масштабе показаны на рис. 1.1, а в увеличенном масштабе – на рис. 1.2 – 1.5. Видно, что файлы карт имеют имена 1, 2, 3, …,52 и содержат каждую последовательность из 13 карт в четырёх мастях (последовательно сверху вниз: трефы – club, бубны – diamond, черви – heart, пики – spade). Файл с именем «0» – фон игры. Эти изображения должны быть добавлены к проекту и использованы для вывода карт на экран. Каждое изображение карты имеет приблизительно только 1 Кбайт объёма, потому что содержит только четыре цвета, чтобы не использовать много памяти.

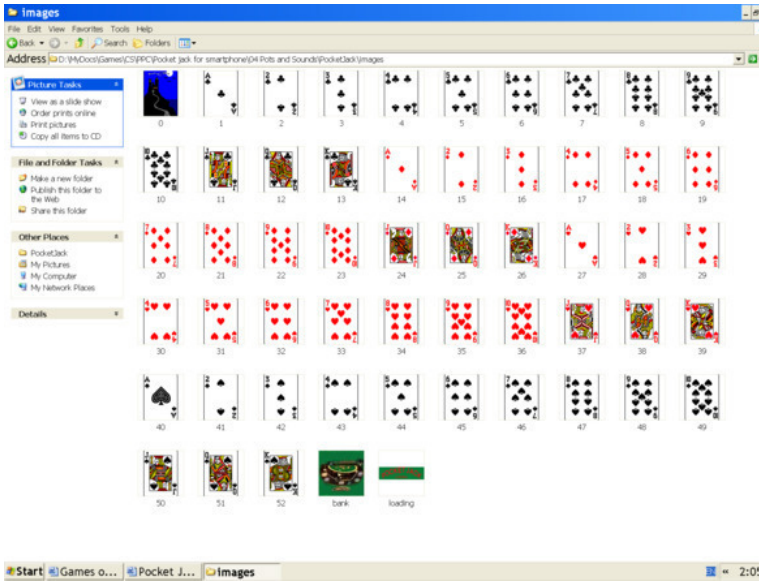


Рис. 1.1. Карты и другие рисунки игры в уменьшенном масштабе.

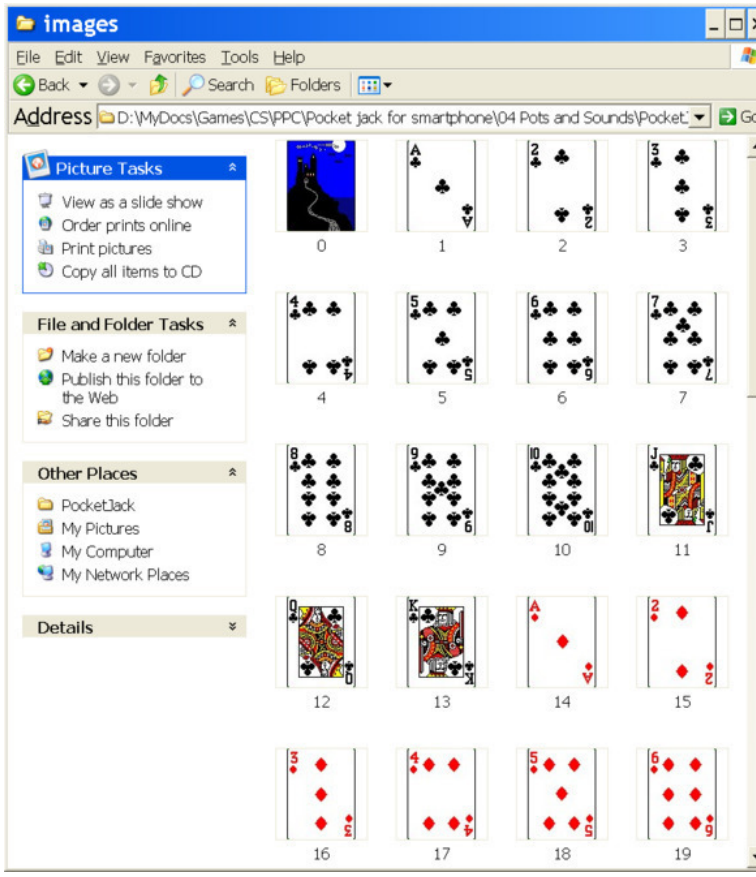


Рис. 1.2. Первые семь карт в увеличенном масштабе.

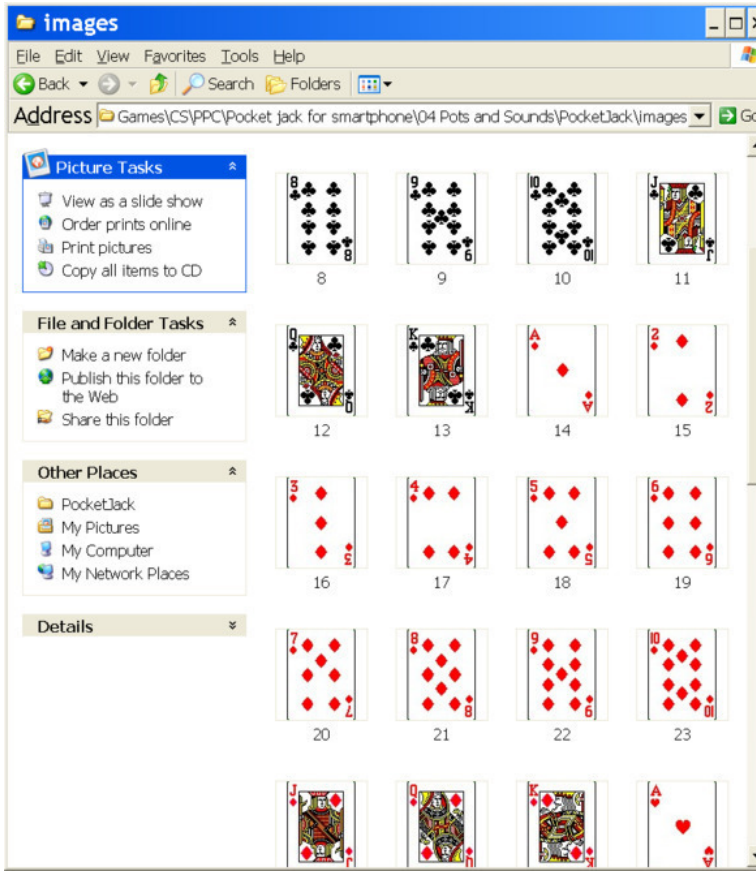


Рис. 1.3. Последующие шестнадцать карт в увеличенном масштабе.

На этом рисунке закончились 13 карт 1, 2, 3, …,13 первой масти трефы – club. Первая карта любой масти – Туз (Ace – A).

Рис. 1.4. Последующие шестнадцать карт в увеличенном масштабе.

На этом рисунке закончились 13 карт 14, 15, 16, …, 26 второй масти бубны – diamond и 13 карт 27, 28, 29, …, 39 третьей масти черви – heart. И здесь первая карта масти – Туз (Ace – A).

Ниже показаны 13 карт 40, 41, 42, …, 52 последней четвертой масти пики – spade.

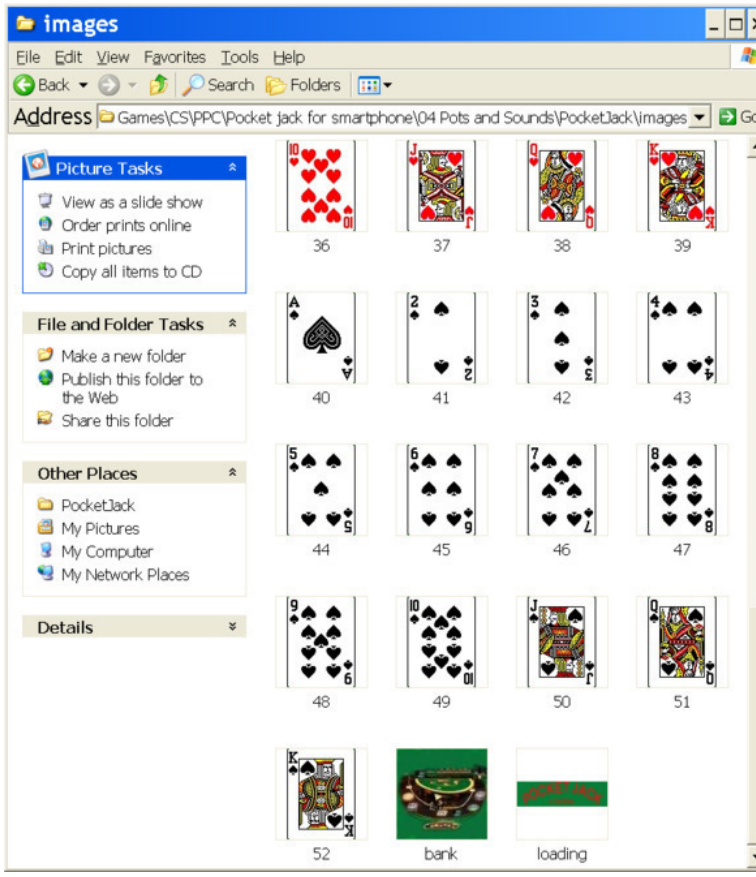


Рис. 1.5. Последние тринадцать карт в увеличенном масштабе.

Значения очков каждой карты следующие: Туз (Ace – A) = 1 или 11; как 1-я, 2-я или 3-я карта – Туз даёт 11 очков; с Валетом, Дамой и Королём, Туз даёт 11 очков и в сумме 10+11=21 эти две карты называются PocketJack, который бьёт карты соперника, даже набравшие 21; как 4-я и последующая карта – Туз даёт 1 очко; цифры на картах от 2 до 9 означают очки этой карты;

карта с числом 10 card with number 10, Валет (Jack – J), Дама (Queen – Q), Король (King – K) = по 10 очков.

1.3. Загрузка в проект изображений карт

Чтобы добавить имеющиеся у нас файлы карт в проект, необходимо сначала добавить в проект папку для этих файлов, затем скопировать в эту папку файлы, а затем свойства этих файлов задать как Embedded Resource, как подробно будет описано далее при создании проекта игры.

Лучший способ загрузить файлы карт в программу при её выполнении – создать массив этих карт в классе Image или Bitmap. После этого изображения могут тогда быть нарисованы на поле игры playfield, когда потребуется. Чтобы загрузить изображения карт в массив, возможны два варианта кода. По первому варианту, код имеет следующий вид:

```
static private Image [] cardImages = new Bitmap [53];

    System.Reflection.Assembly execAssem =
    System.Reflection.Assembly.GetExecutingAssembly ();
    for (int i=0; i <53; i++)
    {
    cardImages [i] =
    new Bitmap(execAssem.GetManifestResourceStream (
    @"PocketJack.cardImages."+i+@".gif»));
```

```
}
```

Видно, что в этом варианте проект имеет имя PocketJack, папка с файлами карт имеет имя cardImages, а в массиве cardImages все файлы карт с именами «i» должны иметь расширение (.gif).

По второму варианту, который мы применим далее в программе, код имеет следующий вид:

```
    public Image CardImage
    {
    get
    {
    int dispNo = CardNo;
    if (!FaceUp)
    {
    dispNo = 0;
    }
    if (cardImages [dispNo] == null)
    {
    cardImages [dispNo] = new Bitmap (
    execAssem.GetManifestResourceStream (
    @"PocketJack.images.» + dispNo + @".gif»));
    }
    return cardImages [dispNo];

}
}
```

Видно, что в этом варианте проект имеет имя PocketJack, папка с файлами карт имеет имя images, а в массиве cardImages все файлы карт с именами «dispNo» должны иметь расширение (.gif).

1.4. Рисование изображений карт

Следующий шаг в разработке игры – процесс рисования карт. Изображение каждой карты должно иметь скруглённые углы. Когда карты прорисовываются на фоне игры, карты со скруглёнными углами выглядят более реалистично. Это – маленькая деталь, но существенная, если мы хотим спроектировать хороший пользовательский интерфейс игры. А если мы пристально посмотрим на изображения карт на экране, то можно увидеть, что углы карт нарисованы зелёным цветом, как показано на рис. 1.6.

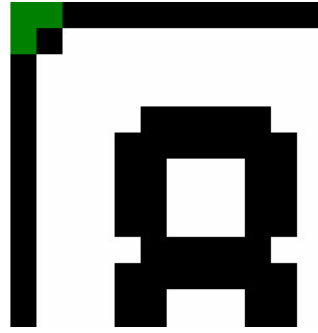


Рис. 1.6. Углы карт нарисованы зелёным цветом.

При рисовании карт мы должны назначить этот цвет как

прозрачный, чтобы был виден фон формы Form1 вокруг углов каждой карты. Мы должны использовать следующий код, чтобы создать объект класса ImageAttributes с целью задания зелёного цвета прозрачным:

```
static public System.Drawing.Imaging.ImageAttributes
cardAttributes;

    static Card ()
    {
    cardAttributes =
    new System.Drawing.Imaging.ImageAttributes ();
    cardAttributes.SetColorKey(Color.Green, Color.Green);
    execAssem =
    System.Reflection.Assembly.GetExecutingAssembly ();

}
```

В этом коде метод SetColorKey даёт начало и конец диапазона цветов, которые будут расценены как прозрачный. Для среды выполнения. NET Compact Framework эти два цвета должны иметь одно и то же значение, так как только один цвет может быть сделан прозрачным.

Целесообразно также после создания проекта в программе задать фон формы Form1 в свойстве BackColor темно-зелёного цвета (DarkGreen) из структуры Color.

Когда изображение карты нарисовало, метод DrawImage

используется следующим образом:

```
private static Rectangle drawRect;

    public void DrawHand (Graphics g, int startx, int starty,
    int gapx, int gapy)
    {
    drawRect. X = startx;
    drawRect. Y = starty;
    foreach (Card card in this)
    {
    drawRect. Width = card.CardImage. Width;
    drawRect. Height = card.CardImage. Height;
    g. DrawImage (
    card.CardImage, // Image
    drawRect, // destination rectange
    0, // srcX
    0, // srcY
    card.CardImage. Width, // srcWidth
    card.CardImage. Height, // srcHeight
    GraphicsUnit. Pixel, // srcUnit
    Card.cardAttributes); // ImageAttributes
    drawRect. X += gapx;
    drawRect. Y += gapy;

}
}
```

Этот код рисует все карты на экране в случайно определённой позиции (при помощи генератора случайных чисел класса Random).

1.5. Класс Card для загрузки карт в программу

В движке игры CardEngine. cs объект класса Card представляет каждую из карт в игре. Этот класс держит фактическое значение карты и рисует её на экране. Он также обеспечивает свойства, которые дают возможность пользователям класса найти координаты карты, получить название карты и другую полезную информацию. Класс Card может использоваться во многих других карточных играх, но есть некоторые особенности, которые характерны для игры в очко.

Первая версия класса Card выполняла загрузку всех изображений, когда приложение начинало выполняться. Каждая из 52 карт и фон игры загружались в самом начале игры. Это делало приложение замедленным. Способ ускорить процесс загрузки состоит в том, чтобы загружать изображения только по запросу при использовании следующего кода:

```
static private Image [] cardImages = new Bitmap [53];

    public Image CardImage
    {
    get
    {
    int dispNo = CardNo;
```

```
        if (!FaceUp)
        {
        dispNo = 0;
        }
        if (cardImages [dispNo] == null)
        {
        cardImages [dispNo] = new Bitmap (
        execAssem.GetManifestResourceStream (
        @"PocketJack.images.» + dispNo + @".gif»));
        }
        return cardImages [dispNo];

    }
    }
```

Переменная cardImages – массив изображений карт. Первоначально все изображения карт в этом массиве пусты. В этом коде переменная dispNo является индексом массива и именем файла карты. Если данный элемент массива – пустой указатель (null), изображение загружается и затем может быть нарисовано. В следующий раз, когда потребуется изображение данной карты, оно будет найдено немедленно. В результате приложение начинает выполняться намного быстрее, чем если бы все карты были загружены в начале игры; время, потраченное, чтобы загрузить только небольшое количество карт, необходимых для игроков, будет небольшим. Если наши приложения нуждаются в большем количе-

стве изображений карт, то это стоит выполнять постепенно по мере загрузки приложения, вместо того, чтобы выполнить это все сразу же в начале игры.

1.6. Класс CardHand для представления карт в руках игрока

В движке игры CardEngine. cs мы нуждаемся в контейнерном классе CardHand (Рука игрока или банкомёта с картами), чтобы держать все карты. Законченная игра будет требовать двух объектов этого контейнера: один – для управляемого компьютером дилера и другой – для игрока. Класс CardHand, который мы собираемся использовать, держит множество карт. Это основано на коллекции ArrayList, которая облегчит для пользователей класса CardHand возможность добавлять и перечислять карты в руке. Эта коллекция также содержит метод, который будет рисовать карты в руке, как показано в следующем коде:

```
public void DrawHand (Graphics g, int startx, int starty,
int gapx, int gapy)
```

Этот метод, приведённый выше, рисует карты, начиная с определённой позиции на экране. Каждая последующая карта рисуется на определённом расстоянии от предыдущей.

Класс CardHand также содержит следующий метод, который вычисляет счёт набранным картам второго игрока blackjack (компьютера):

```
    public int BlackJackScoreHand ()
    {
    int score = 0;
    int aces = 0;
    foreach (Card card in this)
    {
    score += card. BlackJackScore;
    if (card. BlackJackScore == 11)
    {
    aces++;
    }
    }
    while ((score> 21) && (aces> 0))
    {
    score -= 10;
    aces – ;
    }
    return score;

}
```

Метод работает для каждой карты в руке. Он следит за числом тузов (aces), и если пришел туз, то уменьшает счёт карт с учётом туза, чтобы гарантировать, что счёт – как можно ближе к 21, насколько это возможно без перебора.

1.7. Класс CardShoe для представления карт в колоде случайным образом и тестирования игры

Заключительный класс, который управляет картами в движке игры CardEngine. cs, – класс CardShoe. Мы используем этот класс, чтобы обеспечить вывод карт случайным образом (при помощи генератора случайных чисел – г.с.ч.). Игорное казино данного приложения имеет специальное устройство, названное shoe (колода) или deck (колода), которое содержит карты. В начале игры карты перетасованы (shuffle) много раз и помещены в колоду. В процессе многократной перетасовки приложение использует г. с. ч. для размещения карт в виде элементов массива. Класс CardShoe содержит этот массив и заполняет его в начале игры. Все карты вводятся в массив от первой до последней, а затем массив перетасовывается снова, и так несколько раз.

Когда мы проектируем любую систему, мы должны также думать, как мы собираемся её проверять (тестировать). Было бы трудным для нас проверить игру, если бы мы должны были запустить игру 50 раз только для того, чтобы удостовериться, что игра работает правильно, когда игрок получает счёт карт, равный 21. Поэтому класс CardShoe снабжен до-

полнительной особенностью. В дополнение к конструктору этого класса, который позволяет разработчику использовать класс, чтобы выбрать число перетасовок в колоде, имеется ещё перегрузка конструктора, который принимает массив числовых значений типа byte и представляет «расположенную в стеке» колоду. Такая колода не перетасована, и вместо этого располагает карты в специфической заранее предопределённой последовательности. Расположенная в стеке колода даёт возможность разработчику проверить поведение карт в различных ситуациях игры, предоставляя приложению специфическую последовательность значений карт.

Чтобы гарантировать, что расположенная в стеке колода карт не может использоваться в низменных целях (для подтасовок в игре), флажок сообщает пользователю объекта класса CardShoe, действительно ли в данный момент используется расположенная в стеке колода карт. Разработчик, который использует объект этого класса, может проверить этот флажок и гарантировать, что игра не использует расположенную заранее определённым образом в стеке колоду карт.

Проверяя приложение, мы можем создавать сравнительно редкие сочетания карт, типа blackjacks, просто обеспечивая расположенную в стеке колоду, как показано в следующем коде:

```
public CardShoe (byte [] stackedDeck)
{
```

```
        decks = stackedDeck;
        testShoe = true;

    }
```

Расположенная в стеке колода даст в игре Блэк Джека обоим игрокам. Последовательность карты 1 представляет первую карту первой масти в колоде, которая является тузом. Поскольку каждая масть содержит 13 карт, 14-я карта представляет первую карту второй масти, которая является также тузом. В полной игре между игроком (player) и банкомётом (banker) наличие с начала колоды последовательностей карт 1 и 14 приводит и к игроку, и к банкомёту, первоначально начавшему игру, туза. Последовательности карт 11 и 25 представляют Джек (валета) от первых и вторых мастей соответственно, таким образом, каждый игрок получает Джек (валета) как их вторая карта. В игре и игрок (player), и банкомёт (banker) получают по тузу, комбинация Джека (валета) приходит к обоим игрокам, имеющим Блэк Джеки.

1.8. Схема запуска игры

Когда начинается игра, приложение очищает (Clear) «руку игрока» (player's hand) и добавляет две колоды карт следующим образом:

```
CardHand playerHand = new CardHand ();

    playerHand.Clear ();
    dealerHand.Clear ();
    // deal the face down hole card
    dealerHoleCard = shoe.DealCard ();
    dealerHoleCard. FaceUp = false;
    dealerHand.Add (dealerHoleCard);
    // deal the first player card
    playerHand.Add(shoe.DealCard ());
    // deal the second dealer card (face up)
    dealerHand.Add(shoe.DealCard ());
    // deal the second player card

    playerHand.Add(shoe.DealCard ());
```

В этом коде, для простоты, не принимается во внимание чередование раздачи карт между игроком и банкомётом. Это будет учтено далее в полной программе игры.

Далее в проекте имеется команда «Hit Me» меню maneMenu1. После выбора этой команды, компьютер выда-

ёт игроку дополнительную карту, если набранное им количество очков меньше 21, как показано в следующем коде:

```
    void playerHits ()
    {
    if (playerHand. BlackJackScoreHand () <21)
    {
    playerHand.Add(shoe.DealCard ());
    if (playerHand. BlackJackScoreHand ()> 21)
    {
    //We write in the original:
    pot.DoPlaceBet ();
    pot. HouseWins ();
    showPot ();
    mode = GameMode.PlayerBust;
    }
    this.Invalidate ();

}
}
```

Отметим, что метод BlackJackScoreHand каждый раз возвращает счёт «руки игрока» (playerHand). После этого метод Invalidate перерисовывает форму Form1, точнее, перерисовывает карты и обновляет счёт обоих игроков. Аналогично в меню maneMenu1 имеется команда «Себе», по которой банкомёт набирает карты себе (после игрока).

1.9. Рисование очков игроков

Методы Windows для рисования текстов хороши для простых сообщений, но для игры пользователь ожидает что-то более красивое. Например, мы можем рисовать текст на фоне какого-либо рисунка, чтобы выделить текст. Мы можем осуществить это, неоднократно рисуя текст в множестве позиций вокруг его желательного местоположения, перед размещением реального текста на самом верху фона. Чтобы сделать это, был написан ряд утилит, как показано в следующем коде:

```
static private SolidBrush messageBrush =
new SolidBrush (Color. Black);
public static void BigText (string message, int x, int y,
Color back, Color fore,
Font messageFont, Graphics g)
{
int i;
messageBrush.Color = back;
for (i = 1; i <3; i++)
{
g. DrawString (message, messageFont, messageBrush,
x – i, y – i);
g. DrawString (message, messageFont, messageBrush,
x – i, y + i);
g. DrawString (message, messageFont, messageBrush,
```

```
        x + i, y – i);
        g. DrawString (message, messageFont, messageBrush,
        x + i, y + i);
        }
        messageBrush.Color = fore;
        g. DrawString (message, messageFont, messageBrush, x, y);

    }
```

Этот метод BigText снабжен ссылкой на объект графики, чтобы использовать её для рисования текста (message) соответствующим шрифтом (messageFont) в соответствующей позиции. Задаётся также цвет для приоритетных и фоновых версий текста. Метод рисует множество фоновых версий текста перед помещением приоритетной версии на вершине. Метод является статическим, поэтому для вызова не нуждается в объекте класса Utilities, а вызывается напрямую после имени класса, как показано в следующем коде:

```
Utilities.BigText («Dealer Bust»,
20, 80, Color. Black, Color. Yellow, messageFont, g);
```

В этом коде сообщение «Dealer Bust!» означает «Банкомёт перебрал карты».

Объект messageFont класса Font создан в начале приложения и используется для всего рисунка сообщения.

Далее при разработке программы игры мы сначала стан-

дартно создадим шаблон метода Paint (после двойного щелчка по имени события Paint в панели Properties для формы Form1), затем в тело этого шаблона запишем наш код и будем вызывать этот метод каждый раз, когда экран должен быть перерисован. С точки зрения проектирования, считается не очень хорошей практикой выполнять прикладные функции непосредственно в обработчике события Paint. Поэтому для рисования изображений, наше приложение в шаблоне метода Form1_Paint будет вызывать специальный метод paintForm, как показано в следующем коде:

```
private void Form1_Paint (object sender, PaintEventArgs e)
{
paintForm(e.Graphics);
}
```

1.10. Управление игрой

Теперь мы можем использовать вышеупомянутые классы, чтобы осуществить большинство видов игры в карты. Рассмотрим, как в целом осуществляется игра, давая возможность сначала игроку (player) сделать первые ходы (набрать карты), а затем – банкомёту (или дилеру – dealer) сделать ответные ходы (набрать свои карты).

Игра в очко может иметь одно из следующих состояний в течение всей игры:

игрок выдаёт карты себе или Компьютеру (the player is making his or her moves);

игрок перебрал карты (the player busted);

игрок выиграл (the player has won);

банкомёт (dealer) осуществляет набор карт (the dealer is making his or her moves.);

банкомёт перебрал карты (the dealer busted);

банкомёт выиграл (the dealer has won);

счёт равный (the score is tied, known as a push).

В каждом из этих состояний рисунок экрана будет различным, как ответ игры на события. Например, запрос банкомёта следующей карты допускается только тогда, когда игрок закончил набор своих карт. Мы можем представить эти состояния или режимы игры (mode) посредством перечисления enum следующим образом:

```
    public enum GameMode
    {
    LoadingDisplay,
    PlacingBets,
    PlayerActive,
    PlayerWon,
    PlayerBust,
    PocketJack,
    DealerActive,
    DealerWon,
    DealerBust,
    Push

}
```

Переменная типа GameMode сохраняет состояние игры. Эта переменная управляет перерисовкой экрана. Когда состояние игры изменяется, должен произойти ряд действий. Лучший способ получить это поведение состоит в том, чтобы осуществить управление посредством свойства следующим образом:

```
    GameMode modeValue;
    GameMode mode
    {
    get
    {
```

```
    return modeValue;
    }
    set
    {
    switch (value)
    {
    case GameMode. LoadingDisplay:
    BetMinusToolStripMenuItem1.Enabled = false;
    BetPlusToolStripMenuItem. Enabled = false;
    HitMeToolStripMenuItem. Enabled = false;
    StayToolStripMenuItem. Text = «Point»;
    StayToolStripMenuItem. Enabled = false;
    MenuToolStripMenuItem. Text = «21»;
    MenuToolStripMenuItem. Enabled = false;
    break;
    …
    modeValue = value;
    this.Invalidate ();

}
}
```

Когда свойству задано значение value, выполняется часть набора свойства после ключевого слова set. Когда код набора set выполнен, переключатель switch устанавливает приложение нужным способом. Например, когда состояние свойства изменено в состояние PlayerActive, активизированы команды меню Hit и Stay. Везде, где происходит изменение состо-

яния в основном приложении, пользовательский интерфейс находится всегда в нужном состоянии. Это также означает, что мы должны изменить конфигурацию игры только в одном месте кода. Отметим, что, когда состояние игры изменено, вызывается метод Invalidate для обновления экрана.

Состояние игры (game state) также управляет рисованием, когда вызывается событие Paint, как показано в следующем коде:

```
void paintForm (Graphics g)
{
switch (mode)
{
…
case GameMode.PlayerActive:
dealerHand. DrawHand (g, 10, 30, 80, 25);
playerHand. DrawHand (g, 10, 135, 20, 25);
Utilities.BigText (playerHand. BlackJackScoreHand ().
ToString (), 140, 150, Color. Black,
Color. Yellow, messageFont, g);

break;
…
```

Этот код с методом Paint показывает, как приложение управляет выводом сообщений и карт. Очки банкомёта не появляются, когда игрок набирает карты.

1.11. Набор карт банкомётом

Банкомёт (dealer) должен получить две карты, одна из которых появляется лицевой стороной вниз (face down). Мы достигаем этого при использовании следующего кода:

```
// clear the hands
playerHand.Clear ();
dealerHand.Clear ();
// deal the face down hole card
dealerHoleCard = shoe.DealCard ();
dealerHoleCard. FaceUp = false;
dealerHand.Add (dealerHoleCard);
// deal the first player card
playerHand.Add(shoe.DealCard ());
// deal the second dealer card (face up)
dealerHand.Add(shoe.DealCard ());
// deal the second player card
playerHand.Add(shoe.DealCard ());
…
mode = GameMode.PlayerActive;
```

Приложение сохраняет ссылку на «тайную – hole» карту банкомёта, которая инициализируется лицевой стороной вниз, когда начинается набор карт. Это достигается заданием свойству FaceUp (Лицевая сторона вверх) значения, равное False. Когда экран будет перерисовываться, будет на-

рисована также и обратная сторона «тайной – hole» карты банкомёта. Когда банкомёт начинает набор карт, свойству FaceUp (Лицевая сторона вверх) задаётся значение True, и изображение на лицевой стороне карты становится видимым. Отметим, что изменение режима (mode) внизу приведённого кода переводит игру в активное состояние, когда игрок готов принять участие в игре.

Приложение сдаёт карты в том же самом порядке, как в реальной игре с банкомётом и игроком, по очереди берущими карты. Нет никакой программируемой причины для замены между банкомётом и игроком, но в реальной игре проще «расположить в стеке» колоду, если сдача карт не чередуется.

1.12. Набор карт игроком

Приложение содержит метод, который вызывается, когда игрок хочет набрать себе карты. Он получает дополнительную карту, только если счёт – меньше 21, как показано в следующем коде:

```
void playerHits ()
{
if (playerHand. BlackJackScoreHand () <21)
{
playerHand.Add(shoe.DealCard ());
if (playerHand. BlackJackScoreHand ()> 21)
{
//We write in the original:
pot.DoPlaceBet ();
pot. HouseWins ();
showPot ();
mode = GameMode.PlayerBust;
}
this.Invalidate ();

}
}
```

Если счёт игрока превышает 21, игрок совершил перебор карт (the player busts), и состояние игры изменяется, чтобы

отобразить это. Иначе, экран обновляется, что вызывает перерисовку и добавление на экран новой карты.

Когда игрок достиг счёта, которым он доволен, игрок может приостановить (stay) набор новой карты, как показано в следующем коде:

```
void playerStays ()
{
dealerHoleCard. FaceUp = true;
mode = GameMode.DealerActive;
this.Refresh ();
System.Threading.Thread.Sleep (750);
while (dealerHand. BlackJackScoreHand () <17)
{
dealerHand.Add(shoe.DealCard ());
this.Refresh ();
System.Threading.Thread.Sleep (750);
}
if (dealerHand. BlackJackScoreHand ()> 21)
{
mode = GameMode.DealerBust;
pot.PlayerWins ();
showPot ();
return;
}
if (playerHand. BlackJackScoreHand ()>
dealerHand. BlackJackScoreHand ())
{
```

```
    mode = GameMode.PlayerWon;
    pot.PlayerWins ();
    showPot ();
    return;
    }
    if (playerHand. BlackJackScoreHand () <
    dealerHand. BlackJackScoreHand ())
    {
    mode = GameMode.DealerWon;
    //Мы дописываем в оригинале:
    pot.DoPlaceBet ();
    pot. HouseWins ();
    showPot ();
    return;
    }
    if (playerHand. BlackJackScoreHand () ==
    dealerHand. BlackJackScoreHand ())
    {
    mode = GameMode. Push;
    pot.DoPushBet ();
    showPot ();
    return;

}
}
```

Этот метод должен изменить игровое состояние на DealerActive и затем закончить набор карт банкомётом.

Он также переворачивает тайную карту банкомёта лицом вверх, чтобы эту карту можно было увидеть. Игра банкомёта организована по циклу, который неоднократно даёт новые карты банкомёту, если счёт банкомёта меньше 17. Банкомёт обязан запускать игру этим механистическим способом. Приложение содержит паузу на 750 миллисекунд между каждой картой банкомёта, чтобы добавить волнение в игру (в этой паузе также звучит тревожная музыка). Вызов метода Refresh гарантирует, что игрок информирован относительно каждой последующей карты банкомёта.

Если банкомёт получает счёт, больше 21, состояние изменено на DealerBust (Банкомёт перебрал карты), и выполнение метода заканчивается. Иначе, метод решает, кто выиграл и устанавливает соответствующее состояние игры. Тогда игрок может выбрать команду «new game» для начала новой игры, которая снова устанавливает «руки» игрока и банкомёта и соответствующее состояние игры.

1.13. Контекстно-зависимый пользовательский интерфейс

Даже настольный компьютер ограничен доступными средствами управления пользовательским интерфейсом. Он спроектирован так, чтобы во многих случаях можно было пользоваться только одной рукой, что означает, что управления должны быть удобными. В данном приложении есть ограниченное число действий, которые игрок может выполнить. Чтобы сделать игру проще, необходимо отобразить эти действия на клавишах клавиатуры и кнопках мыши, которыми игрок наиболее часто пользуется, как показано в следующем коде:

```
void doEnter ()
{
switch (mode)
{
case GameMode. LoadingDisplay:
break;
case GameMode.PlacingBets:
startPlay ();
break;
case GameMode.PlayerActive:
playerHits ();
break;
```

```
        case GameMode. PocketJack:
        case GameMode.PlayerWon:
        case GameMode.PlayerBust:
        case GameMode.DealerActive:
        case GameMode.DealerWon:
        case GameMode.DealerBust:
        case GameMode. Push:
        startHand ();
        break;

    }
    }
```

Этот код означает программируемые команды Меню на основе элемента управления MenuStrip.

Когда игрок нажимает клавиши на клавиатуре, предыдущий код решает важные для приложения задачи. В режиме PlayerActive, нажимая клавишу Enter, программа сдаёт игроку следующую карту. В любом другом режиме программа сдаёт карты для новой игры. Используя команду Stay (Приостановка) элемента управления MenuStrip, пользователь может приостановить и запустить игру.

Мы должны рассмотреть эту проблему, проектируя наши карточные и другие игры. Мы должна попробовать использовать игру на мобильном устройстве (о чем описано в наших предыдущих книгах), которое мы держим в одной, например, левой руке, в то время как в другой руке мы дер-

жим чашку кофе. Это должно быть возможным. Отметим, что команды элемента управления MenuStrip могут также выполнять все эти контекстно-зависимые действия; например, по команде Hit программа сдаёт игроку следующую карту. Игрок никогда не обязан предполагать, а должен точно знать, как выполнить специфическое действие.

1.14. Банк и ставка

Большинство игроков хочет иметь возможность взять банк (Pot). Чтобы сделать это, игра должна иметь ставку (Bet), которую игрок может выиграть или проиграть на каждой сдаче карт. В начале игры игроку показывают размеры доступного банка и ставки, которые он может увеличить или уменьшить. Если капитал игрока накапливается плохо, приложение должно дать игроку опцию сброса размера банка назад к оригинальным значениям начала игры. Банк осуществлен как отдельный класс Pot, чтобы он мог использоваться в других играх (если требуется). Перед каждой сдачей карт игрок может решить, сколько держать пари (какую назначить ставку), до количества денег в банке. Если игрок попробует держать пари больше, чем размер банка, приложение предложит сбросить банк до начального значения.

Игрок легко управляет размером ставки, нажимая клавиши со стрелками вверх или вниз. Эти команды также доступны на элементе управления MenuStrip.

После выбора размера ставки игрок может запустить игру при помощи команды на элементе управления MenuStrip и мыши или клавиш. Режим банка и ставки фактически управляется посредством дополнительного игрового состояния, которое также рисует размеры банка и ставки на фоне игры, как будет показано далее.

1.15. Добавление справочной формы

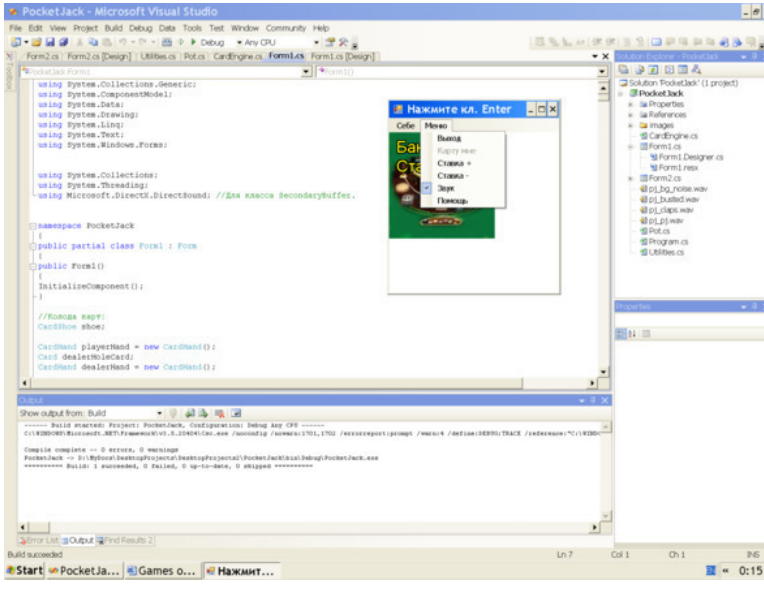
Приложение теперь имеет все основные составляющие, позволяющие запустить игру. Однако несколько добавлений придадут приложению более современный вид. Эти добавления включают:

справочную форму,

рисунок с сообщением Loading, появляющийся на экране во время длительной загрузки и вывода на экран какого-либо изображения

и звуковое сопровождение.

Справочная форма – это вторая форма Form2 (вслед за главной формой Form1 с интерфейсом игры), которая содержит один или несколько элементов управления с панели инструментов Toolbox, например, окно текста TextBox со справочной информацией для игрока. Когда при помощи мыши игрок раскрывает Меню элемента управления MenuStrip, а затем в этом Меню выбирает команду Помощь (рис. 1.7), рядом с первой формой Form1 появляется эта справочная форма Ащкь2 (рис. 1.8). Напомним, что Меню элемента управления MenuStrip закрывается после его повторного выбора мышью. Эта форма Form2 закрывается щелчком по значку Close, и на экране остаётся одна игровая форма Form1.


PocketJack - Microsoft Visual Studio
Нажмите кл. Enter
Себе
Меню
Выход
Ставка +
Ставка -
Звук
Помощь

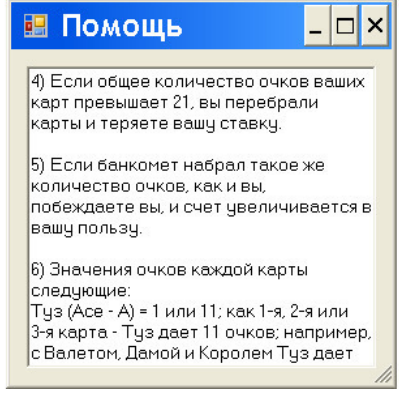

Помощь
4) Если общее количество очков ваших карт превышает 21, вы перебрали карты и теряете вашу ставку.
5) Если банкомет набрал такое же количество очков, как и вы, побеждаете вы, и счет увеличивается в вашу пользу.
6) Значения очков каждой карты следующие:
Туз (Ace - A) = 1 или 11; как 1-я, 2-я или 3-я карта - Туз дает 11 очков; например, с Валетом, Дамой и Королем Туз дает

Рис. 1.7. Команда Помощь в Меню. **Рис. 1.8.** Справочная форма.

1.16. Добавление рисунка загрузки

Одним из заключительных этапов разработки игры является добавление в проект и вывод на экран рисунка загрузки (a loading screen) в виде файла loading.gif. Этот рисунок появляется на экране в режиме (mode = GameMode. LoadingDisplay;) при помощи метода Paint в начале загрузки в программу игровых компонентов (например, в виде графических и звуковых файлов) и построения формы, а затем исчезает. Здесь основная проблема – обеспечение видимости фона формы в процессе загрузки. Чтобы сделать игровую фон формы видимым, мы устанавливаем для формы значение свойства Visible, как показано в следующем коде:

```
System.Reflection.Assembly asm =
System.Reflection.Assembly.GetExecutingAssembly ();
loadingImage = new
Bitmap(asm.GetManifestResourceStream (
"PocketJack.images.loading.gif»));
bankImage = new
Bitmap(asm.GetManifestResourceStream (
"PocketJack.images.bank.jpg»));
mode = GameMode. LoadingDisplay;
this.Visible = true;
```

В этом коде в строке

PocketJack.images.loading.gif

переменная PocketJack означает имя проекта (или пространства имён),

а переменная images – имя папки, в которой имеется графический файл loading.gif.

После запуска игры, рисунок загрузки loading.gif появляется в середине экрана и находится на экране все время загрузки игровых компонентов, как определено в следующем коде:

```
void paintForm (Graphics g)
{
switch (mode)
{
case GameMode. LoadingDisplay:
//We draw all images below the menu:
g. DrawImage (
bankImage, 0, StayToolStripMenuItem. Height);
g. DrawImage (
loadingImage, 0, StayToolStripMenuItem. Height +60);

break;

case GameMode.PlacingBets:
g. DrawImage (bankImage, 0, StayToolStripMenuItem.
Height);
```

```
        Utilities.BigText («Bank: " + pot.PotValue.ToString (),
        10, 40, Color. Black,
        Color. Yellow, messageFont, g);
        Utilities.BigText («Bet: " +
        pot.BetValue.ToString (), 10, 80, Color. Black,
        Color. Yellow, messageFont, g);

        break;
...
```

На рис. 1.9 показан узкий слева – направо рисунок загрузки loading.gif с надписью «POCKET JACK LOADING» поверх фона игры в виде файла bank.jpg. Этот рисунок держится на экране всего несколько секунд, пока не загрузятся все графические и звуковые файлы, а затем исчезает, уступая место информации о размерах Банка и Ставки, показанных на следующем рисунке.

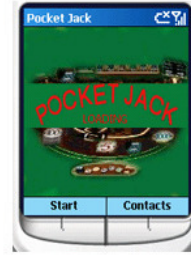


Рис. 1.9. Рисунок загрузки loading.gif

Разработку звукового сопровождения игры мы опишем далее при написании программы, а сейчас перейдем к правилам игры.

1.17. Правила игры

Сформулируем правила данной компьютерной карточной (из 52 карт) игры в очко, которые далее будут также записаны в справочную форму Form2, выводимую после выбора команды Помощь из меню элемента управления MenuStrip на главной форме Form1.

1. Мы (пользователи) считаемся как один игрок и играем один на один с компьютером-банкомётом.

2. Мы устанавливаем в Банке начальную сумму денег, например, 500 долларов и определяем Ставку в игре, равную, например, 5 долларам. В каждой сдаче карт наш выигрыш будет увеличивать Банк, а проигрыш – уменьшать Банк на величину Ставки.

По окончании игры, если Банк будет больше 500, то мы выиграли у Банкомёта разницу между итоговым значением Банка и начальным значением Банка в 500 долларов.

Если же по окончании игры Банк будет меньше 500, то мы проиграли Банкомёту разницу между начальным значением Банка в 500 долларов и итоговым значением Банка.

Естественно, по методике данной главы в приводимом далее проекте мы можем запрограммировать другие варианты Банка, Ставки и условий игры.

3. В данном варианте игры, наша цель состоит в том, чтобы на экране набрать «в руку» карты с очками, как можно

ближе к 21, но не превышая 21, и, к тому же, больше, чем у банкомёта. На экране сначала мы видим вверху карту банкомёта, а внизу – две наши карты с очками этих карт.

4. После запуска игры на форме появляются команда Себе и команда общего Меню (рис. 1.10), приведённый выше рисунок loading.gif, а затем фон игры в виде приведённого выше изображения bank.jpg и начальный счёт игры:

Банк: 500 Ставка: 5

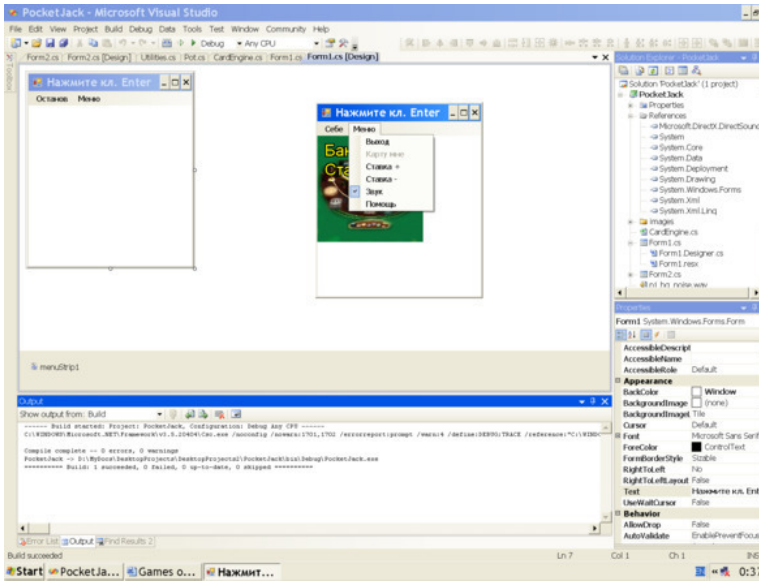


Рис. 1.10. Исходное состояние (заставка) игры. **Рис. 1.11.** Команда «Ставка +” в Меню.

Если мы желаем изменить значение Ставки, то при помощи мыши раскрываем Меню элемента управления MenuStrip, в котором выбираем команду «Ставка +” или «Ставка -” (рис. 1.11). Это меню закрывается после его повторного выбора. Значения Банка и Ставки можно также изменить в приведённой далее программе. Для примера, оставляем заданные по умолчанию значения Банка и Ставки. Ни-

каких игровых действий пока не производится.

5. Для начала игры нажимаем клавишу Enter.

В верхней части экрана (как свойство Text формы Form1) появляются начальные значения Банка и Ставки (рис. 1.12): Банк: 500 Ставка: 5

В верхней половине экрана появляется одна закрытая и одна открытая карта банкомёта, а в нижней – две карты игрока с очками этих карт (рис. 1.12). Звучит сопровождение в виде файла pj_bg_noise. wav тревожного шума окружающих игрока и банкомёта групп поддержки.

6. У нас два варианта действий. Видя наши исходные очки и очки банкомёта, либо,

по первому варианту, при помощи клавиши Enter (или команды «Карту мне» из Меню, см. предыдущий рисунок) нам взять следующую карту,

либо, по второму варианту, при помощи мыши выбрать в Меню команду Себе (попросить банкомёт брать карты себе).


Банк: 500 Ставка: 5
Себе
Меню
14

Рис. 1.12. Карта банкомёта и две карты игрока. **Рис. 1.13.** Счёт игры изменился.

Мы помним (из предыдущего описания), что банкомёт будет набирать карты себе до тех пор, пока у него не наберётся ровно 17 или более очков. А у нас уже 14 очков. Конечно, можно дать команду Банкомёту брать карты себе (в надежде на перебор карт у него). Но мы решаемся, и при помощи клавиши Enter берём следующую карту. Но, увы, к нам пришла не та карта, и мы видим грустное сообщение «Вы перебрали» с итоговыми нашими очками 22 (рис. 1.13). Звучит недо-

вольство с помощью звукового файла pj_busted. wav. В верхней части экрана Банк уменьшается на величину Ставки, и счёт игры становится не в нашу пользу таким:

Банк: 495 Ставка: 5

7. Для осуществления нового набора карт мы можем выбрать команду «Сдача карт» (при помощи мыши) или нажать клавишу Enter.

Нажимаем клавишу Enter.

Появляется фон игры в виде приведённого выше изображения bank.jpg и новый счёт игры не только в верхней части экрана (как свойство Text формы Form1), но и в центральной части экрана в виде крупного изображения, нарисованного методом DrawString (рис. 1.14):

Банк: 495 Ставка: 5

Так продолжается игра. Например, в середине игры нажимаем клавишу Enter. Появляются очередные начальные наборы карт у нас и Банкомёта (рис. 1.15).

8. У нас те же два варианта действий. Видя наши исходные очки и очки банкомёта, либо,

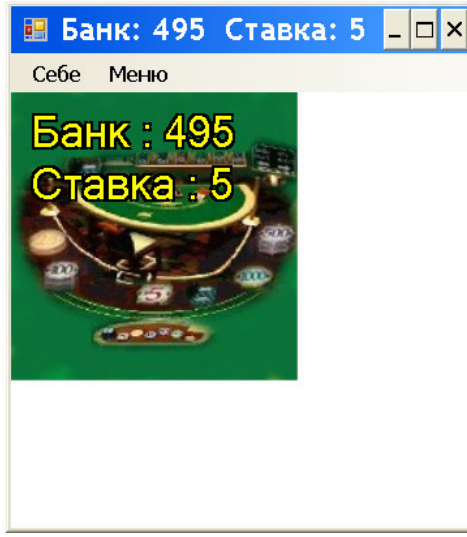
по первому варианту, при помощи клавиши Enter нам взять следующую карту,

либо, по второму варианту, при помощи мыши выбрать в Меню команду «Take a card to yourself» (попросить банкомёт брать карты себе).

Мы помним (из предыдущего описания), что банкомёт будет набирать карты себе до тех пор, пока у него не наберётся

ровно 17 или более очков.

А у нас теперь всего 9 очков. Поэтому при помощи клавиши Enter смело берём следующую карту. К нам пришла пятёрка, появилось количество очков 14 наших карт (рис. 1.16).



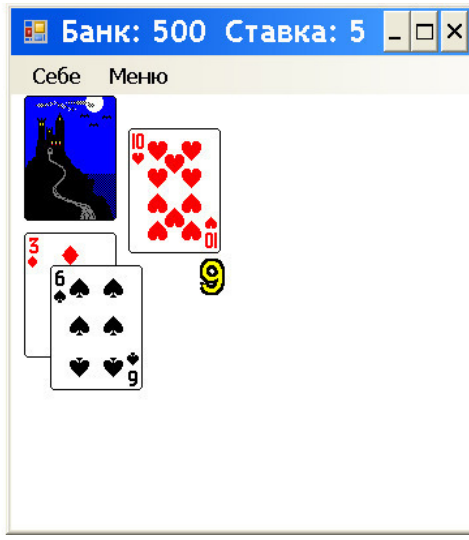
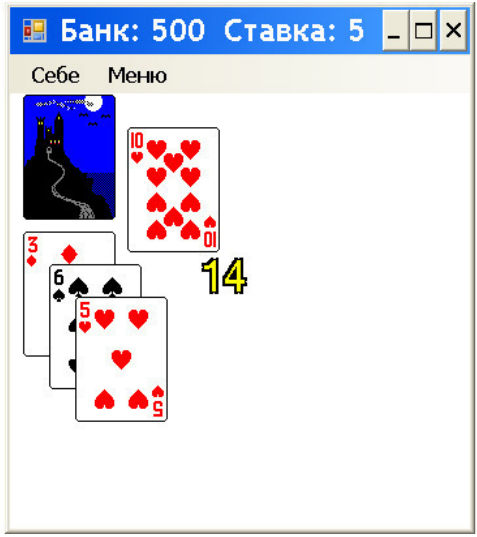


Рис. 1.14. Заставка игры с новым счётом. **Рис. 1.15.** Очередные начальные наборы карт.


Банк: 500 Ставка: 5
Себе
Меню
14

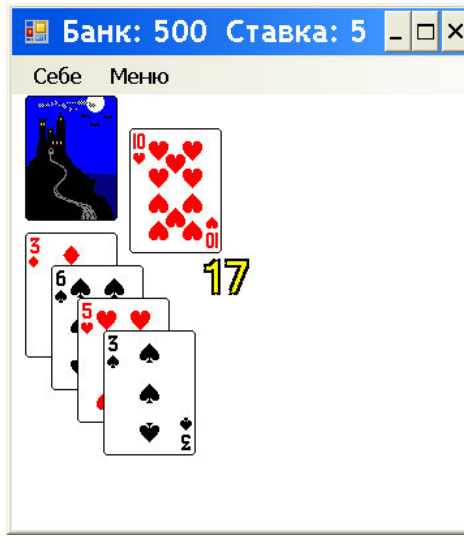


Рис. 1.16. Мы набрали 14 очков. **Рис. 1.17.** Мы набрали 17 очков.

9. При помощи клавиши Enter берём следующую карту. К нам пришла тройка, появилось количество очков 17 наших карт (рис. 1.17).

10. Ситуация непростая, т.к. набранное нами число 17 равно числу 17, на котором Банкомёт останавливает набор карт, и близко к числу 21, за которым начинается перебор карт.

У нас те же два варианта действий. Видя наши очки и очки

банкомёта, либо,

по первому варианту, при помощи клавиши Enter нам взять следующую карту, но велика вероятность перебора карт (больше 21),

либо, по второму варианту, при помощи мыши в Меню выбрать команду «Take a card to yourself» (попросить банкомёт брать карты себе), с надеждой на перебор у него.

Решаемся, и при помощи клавиши Enter берём следующую карту. Но опять, увы, к нам пришла не та карта, и мы видим грустное сообщение «Вы перебрали» с итоговыми нашими очками (рис. 1.18).

Звучит недовольство с помощью звукового файла pj_busted. wav.

В верхней части экрана Банк уменьшается на величину Ставки, например, с 505 до 500 и счёт игры становится таким: Банк: 500 Ставка: 5


Банк: 495 Ставка: 5
Сдача карт
Меню
Вы перебрали.
29

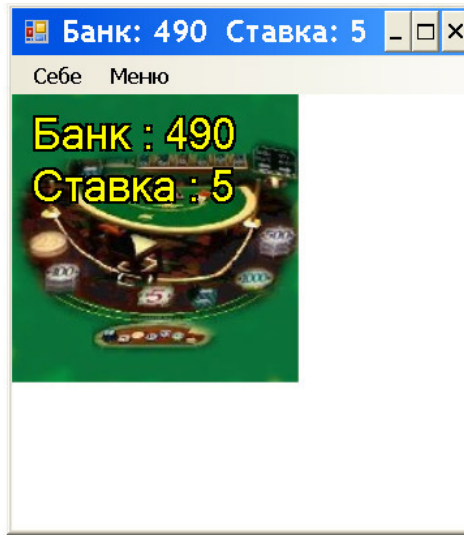


Рис. 1.18. У нас опять превышение очков. **Рис. 1.19.** Заставка игры с новым счётом.

11. Продолжаем игру, например, на каком-то этапе игры нажимаем клавишу Enter.

Появляется фон игры в виде приведённого выше изображения bank.jpg и новый счёт игры не только в верхней части экрана (как свойство Text формы Form1), но и в центральной части экрана в виде крупного изображения, нарисованного методом DrawString (рис. 1.19):

Банк: 490 Ставка: 5

12. Для осуществления нового набора карт мы можем выбрать команду «Сдача карт» (при помощи мыши) или нажать клавишу Enter.

Нажимаем клавишу Enter. Появляется новый начальный набор карт у нас и Банкомёта.

Дальше поступаем аналогично, сначала сами набираем (или не набираем, оставляя начальный вариант) карты до удовлетворяющего нас результата, а затем даём команду Банкомёту набирать карты себе.

Теперь кратко опишем возможные типичные варианты игры.

13. При помощи клавиши Enter набираем (или не набираем, оставляя начальный счёт) карты до удовлетворяющего нас результата, например, до 16 (рис. 1.20) и при помощи мыши даём команду Банкомёту набирать карты себе.

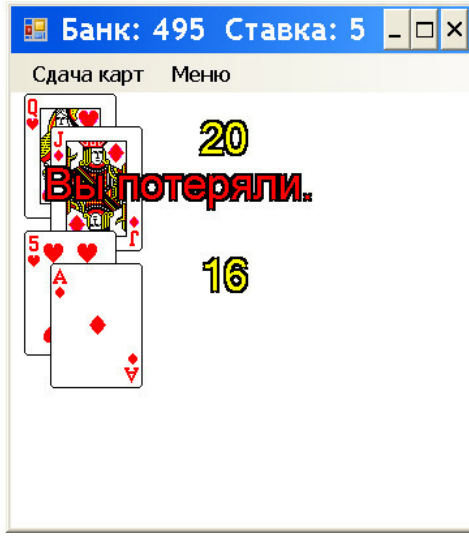
Банкомёт, к нашему неудовольствию, набирает 20 очков, и мы видим неприятное для нас сообщение «Вы потеряли».

В верхней части экрана Банк уменьшается на величину Ставки не в нашу пользу.

14. При помощи клавиши Enter получаем новую сдачу карт, однако Банкомёт, к нашему неудовольствию, сразу же набирает 21 очко (или коротко, «очко»), и мы видим неприятное для нас сообщение «Вы потеряли» (рис. 1.21).

В верхней части экрана Банк уменьшается на величину Ставки не в нашу пользу.

15. При помощи клавиши Enter набираем (или не набираем, оставляя начальный счёт) карты до удовлетворяющего нас результата, например, до 18 (рис. 1.22) и при помощи мыши даём команду Банкомёту набирать карты себе.



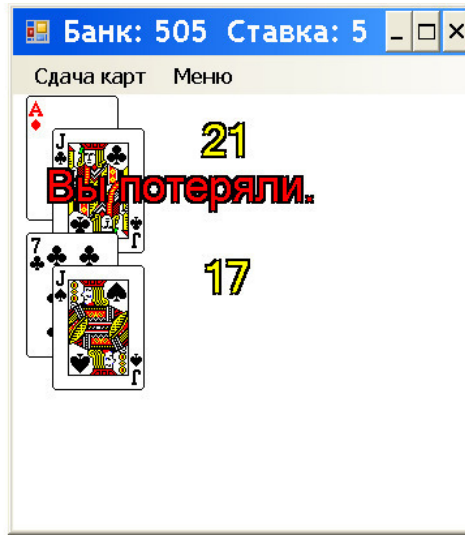


Рис. 1.20. Мы проиграли. **Рис. 1.21.** У Банкомёта – очко.


Банк: 470 Ставка: 5
Сдача карт
Меню
17
Вы выиграли!
18

Рис. 1.22. Мы выиграли. **Рис. 1.23.** Превышение очков у Банкомёта.

Банкомёт, к нашей радости, набирает всего 17 очков, и мы видим приятное для нас сообщение «Вы выиграли».

В верхней части экрана Банк увеличивается на величину Ставки в нашу пользу.

Звучат аплодисменты в наш адрес в виде файла pj_claps. wav.

16. При помощи клавиши Enter набираем (или не набираем, оставляя начальный счёт) карты до удовлетворяюще-

го нас результата, например, до 20 (рис. 1.23) и при помощи мыши даём команду Банкомёту набирать карты себе.

Банкомёт, к нашей радости, набирает 22 очка, и мы видим приятное для нас сообщение «Превышение очков (у Банкомёта)».

В верхней части экрана Банк увеличивается на величину Ставки в нашу пользу.

17. При помощи клавиши Enter набираем (или не набираем, оставляя начальный счёт) карты до удовлетворяющего нас результата, например, до 17 (рис. 1.24) и при помощи мыши даём команду Банкомёту набирать карты себе.

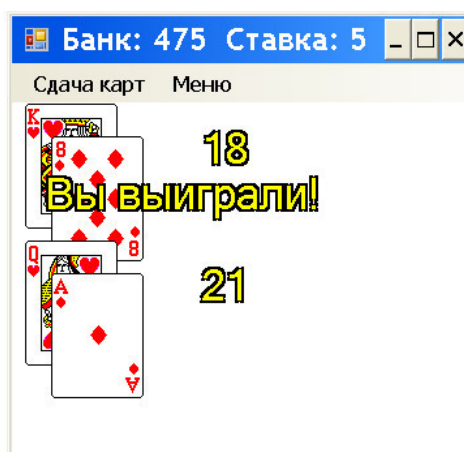


Рис. 1.24. Равное количество очков. **Рис. 1.25.** Туз с Дамой – это PocketJack.

Банкомёт, к нашей радости, набирает также 17 очков. Согласно правилам, при равенстве очков победителем считается тот, кто первым набрал это количество очков, т.е. игрок. И мы видим приятное для нас сообщение «Вы выиграли». Это правило можно перепрограммировать в приведённой далее программе, установив, например, неизменность Банка при ничьей. В верхней части экрана Банк увеличива-

ется на величину Ставки в нашу пользу.

18. При помощи клавиши Enter получаем новую сдачу карт, и, к нашей радости, мы сразу же набираем 21 очко (или коротко, «очко»), причём не просто «очко». Просто «очко» – это Туз с десяткой. А Туз с Валетом, Дамой, Королём или десяткой даёт 21 очко и называется PocketJack, который бьёт даже 21 очко у соперника. И мы видим приятное для нас сообщение «Вы выиграли» (рис. 1.25).

В верхней части экрана Банк увеличивается на величину Ставки в нашу пользу.

Звучит радостная мелодия со словами PocketJack в виде файла pj_pj. wav (в этом имени pj и есть сокращение слова PocketJack).

19. Напомним, что

13 карт 1, 2, 3, …,13 первой масти – это трефы (club),

13 карт 14, 15, 16, …, 26 второй масти – это бубны (diamond),

13 карт 27, 28, 29, …, 39 третьей масти – это черви (heart),

13 карт 40, 41, 42, …, 52 последней четвертой масти – это пики (spade).

Первая карта любой масти – Туз (Ace – A).

Значения очков каждой карты следующие:

Туз (Ace – A) = 1 или 11;

как 1-я, 2-я или 3-я карта – Туз даёт 11 очков;

с Валетом, Дамой и Королём, Туз даёт 11 очков и в сумме 10+11=21 эти две карты называются PocketJack, который

бьёт карты соперника (игрока или Банкомёта), даже набравшие 21;

как 4-я и последующая карта – Туз даёт 1 очко;

цифры на картах от 2 до 9 означают очки этой карты;

карта с числом 10, Валет (Jack – J), Дама (Queen – Q), Король (King – K) = по 10 очков.

20. Банкомёт сдаёт карты с единственной колоды карт.

21. Банкомёт будет сдавать себе карты, пока не достигнет 17 или больше.

22. Первую карту банкомёта можно запрограммировать так, что она будет сдаваться лицевой стороной вниз и невидимой.

23. Мы должны или оставить Ставку по умолчанию, или установить новую вашу Ставку до сдачи карт (в последнем случае используем команды «Ставка +" и «Ставка -" в Меню для элемента управления MenuStrip).

24. Наше значение Банка все время показывается на экране. Если значение Банка станет ниже нашей Ставки, нам предложат начать новую игру.

25. Когда мы набрали карты, мы можем приостановить игру, выбрав в Меню команду Останов. Банкомёт покажет свою карту (если до этого она была невидима).

26. Схема оплаты:

проигравший платит победителю по договорённости, например, 1:1.

27. Чтобы начать новую игру, необходимо при помощи

мыши в Меню выбрать команду Выход, а затем ещё раз нажать клавишу Enter.

28. Если в игре по очереди участвуют несколько игроков, то победителем считается игрок, который, например, за одно и то же для всех игроков время набрал больший Банк. Здесь возможны различные варианты, в зависимости от наших желаний.

29. Игра в очко желает вам успехов во всех играх.

На основании этих правил игры можно разработать другие правила этой или другой подобной игры с внесением соответствующих изменений в приведённую далее программу (если будет необходимость в этих изменениях).

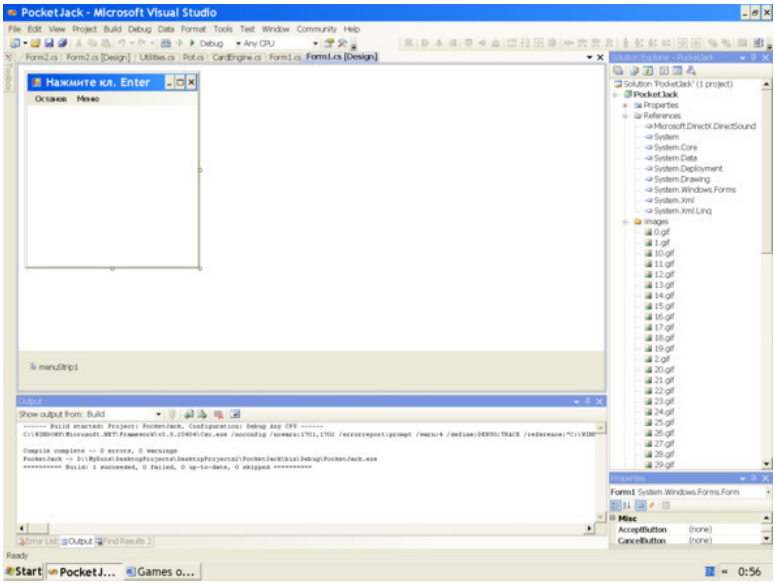
1.18. Создание проекта

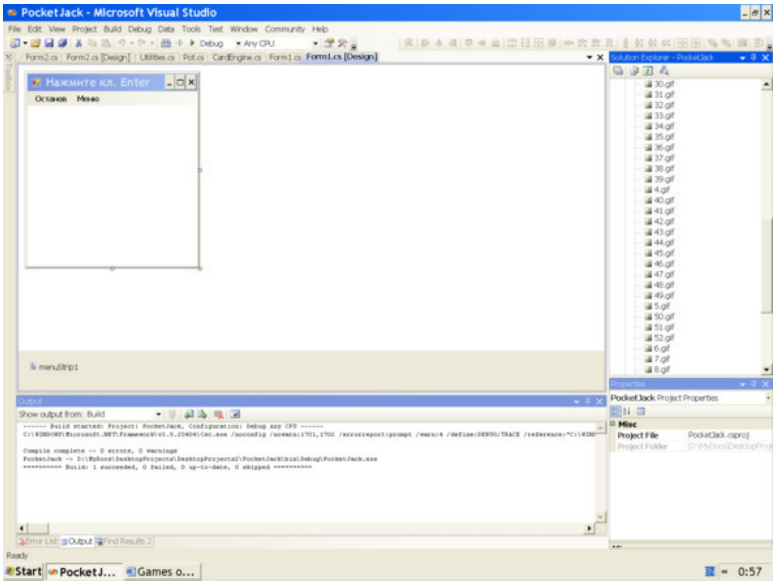
Создаём проект по обычной схеме: в VS в панели New Project мы выбираем Templates, Visual C#, Windows Classic Desktop, Windows Forms App (.NET Framework), в окне Name записываем имя проекта PocketJack и щёлкаем OK. Это имя проекта далее будет использоваться в программе. Поэтому, если в окне Name мы запишем другое имя проекта, отличное от PocketJack, то в приведённой далее программе в строках кода по загрузке графических и звуковых файлов, а также имя пространства имён во всех наших файлах (кроме Form1.cs) необходимо будет заменить PocketJack на новое имя.

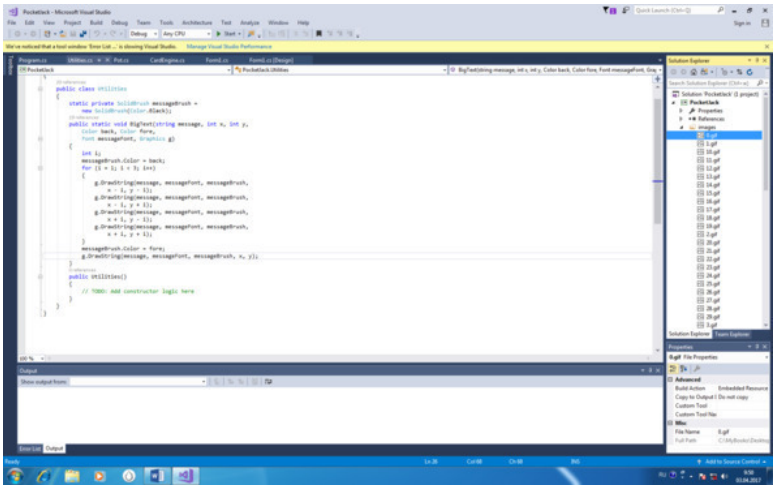
Создаётся проект, появляется форма Form1 (см. приведённые в правилах игры рисунки) в режиме проектирования. Проектируем (или оставляем по умолчанию) форму, как подробно описано выше с размерами формы, например, 361; 408. Чтобы изображения были лучше видны на форме, в панели Properties (для Form1) в свойстве BackColor вместо заданного по умолчанию фона Control выбираем белый фон Window.

Если в игре применяется много графических файлов, то их целесообразно разместить в одной папке с именем, например, images. Для добавления в проект первой папки, в панели Solution Explorer (рис. 1.26 – 1.28) выполняем правый

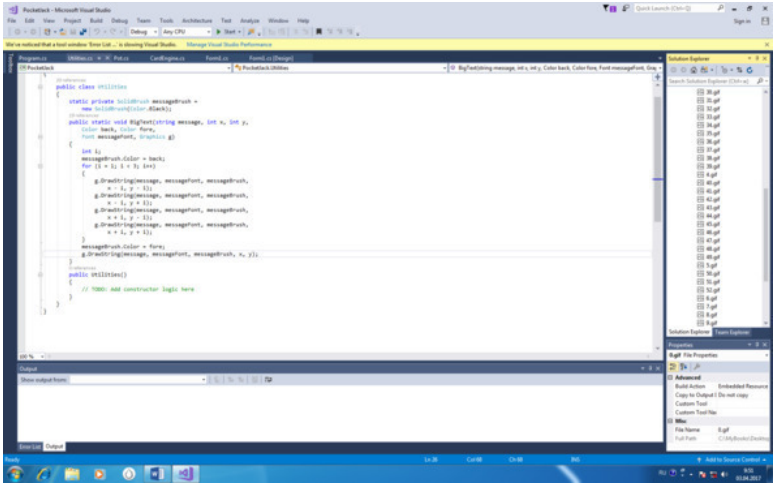
щелчок по имени проекта, в контекстном меню выбираем Add, New Folder, в поле появившегося значка папки записываем это имя images и нажимаем клавишу Enter. Добавляем в эту папку images файл 0.gif (для скрытой карты Банкомёта с изображением средневековой башни, показанной выше на самом первом рисунке) по стандартной схеме, а именно: выполняем правый щелчок по имени этой папки, в контекстном меню выбираем Add, Existing Item, в панели Add Existing Item в окне «Files of type» выбираем «All Files», в центральном окне находим и выделяем имя файла (загруженного, например, из Интернета или из указанной в списке литературы ссылки) и щёлкаем кнопку Add (или дважды щёлкаем по имени файла). В панели Solution Explorer мы увидим этот файл.


PocketJack - Microsoft Visual Studio
File Edit View Project Build Debug Data Format Tools Test Window Community Help
Debug
Any CPU
Form2.cs | Form2.cs [Design] | Utilities.cs | Pot.cs | Card|Engine.cs | Form1.cs | Form1.cs [Design]
Нажмите кл. Enter
Останов
Меню
menuStrip1
Output
Show output from: Build
Build started: Project: PocketJack, Configuration: Debug Any CPU
Compile complete -- 0 errors, 0 warnings
PocketJack -> D:\MyDocs\Desktop\Projects\Desktop\Projects\PocketJack\bin\Debug\PocketJack.exe
Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped
Error List
Output
Find Results 2
Solution Explorer - PocketJack
Solution 'PocketJack' (1 project)
PocketJack
Properties
References
Microsoft.DirectX.DirectSound
System
System.Core
System.Data
System.Deployment
System.Drawing
System.Windows.Forms
System.Xml
System.Xml.Linq
images
0.gif
1.gif
10.gif
11.gif
12.gif
13.gif
14.gif
15.gif
16.gif
17.gif
18.gif
19.gif
2.gif
20.gif
21.gif
22.gif
23.gif
24.gif
25.gif
26.gif
27.gif
28.gif
29.gif
Properties
Form1 System.Windows.Forms.Form
Misc
AcceptButton (none)
CancelButton (none)
Ready
Start
PocketJ...
Games o...
0:56


PocketJack - Microsoft Visual Studio
File Edit View Project Build Debug Data Tools Test Window Community Help
Debug
Any CPU
Form2.cs | Form2.cs [Design] | Utilities.cs | Pot.cs | Card[Engine.cs | Form1.cs | Form1.cs [Design]
Нажмите кн. Enter
Останов
Меню
menuStrip1
Solution Explorer - PocketJack
30.gif
31.gif
32.gif
33.gif
34.gif
35.gif
36.gif
37.gif
38.gif
39.gif
4.gif
40.gif
41.gif
42.gif
43.gif
44.gif
45.gif
46.gif
47.gif
48.gif
49.gif
5.gif
50.gif
51.gif
52.gif
6.gif
7.gif
8.gif
Properties
PocketJack Project Properties
Misc
Project File
PocketJack.csproj
Project Folder
Output
Show output from: Build
Compile complete -- 0 errors, 0 warnings
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
Error List
Output
Find Results 2
Ready
Start
PocketJ...
Games o...
0:57

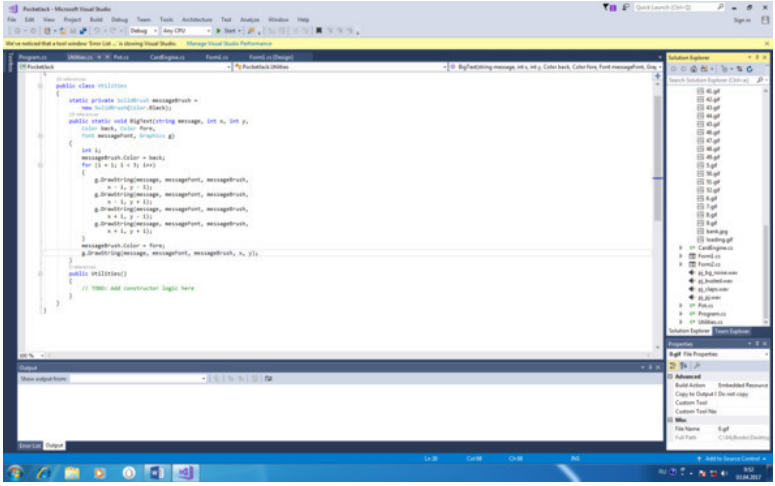


```
public class Utilities
{
    static private SolidBrush messageBrush =
        new SolidBrush(Color.Black);
public static void BigText(string message, int x, int y,
    Color back, Color fore,
    Font messageFont, Graphics g)
{
    int i;
    messageBrush.Color = back;
    for (i = 1; i < 3; i++)
    {
        g.DrawString(message, messageFont, messageBrush,
            x - i, y - i);
        g.DrawString(message, messageFont, messageBrush,
            x - i, y + i);
        g.DrawString(message, messageFont, messageBrush,
            x + i, y - i);
        g.DrawString(message, messageFont, messageBrush,
            x + i, y + i);
    }
    messageBrush.Color = fore;
    g.DrawString(message, messageFont, messageBrush, x, y);
}
public Utilities()
{
    // TODO: Add constructor logic here
}
}
```



```
public class Utilities
{
    static private SolidBrush messageBrush =
        new SolidBrush(Color.Black);
public static void BigText(string message, int x, int y,
    Color back, Color fore,
    Font messageFont, Graphics g)
{
    int i;
    messageBrush.Color = back;
    for (i = 1; i < 3; i++)
    {
        g.DrawString(message, messageFont, messageBrush,
            x - i, y - i);
        g.DrawString(message, messageFont, messageBrush,
            x - i, y + i);
        g.DrawString(message, messageFont, messageBrush,
            x + i, y - i);
        g.DrawString(message, messageFont, messageBrush,
            x + i, y + i);
    }
    messageBrush.Color = fore;
    g.DrawString(message, messageFont, messageBrush, x, y);
}
public Utilities()
{
    // TODO: Add constructor logic here
}
}
```

Рис. 1.26. Папка «images» (верхняя часть). **Рис. 1.27.** Папка «images» (средняя часть).



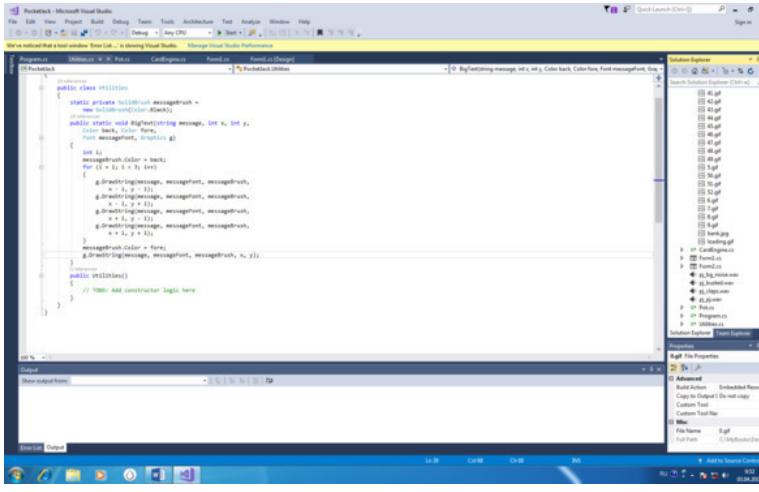


Рис. 1.28. Файлы проекта. **Рис. 1.29.** Выбираем Embedded Resource.

Теперь этот же файл 0.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 вместо заданного по умолчанию выбираем значение Embedded Resource, рис. 1.29.

Аналогично добавляем в эту папку и встраиваем в виде ресурса остальные графические файлы:

файлы для 52 карт, показанных выше

1.gif, 10.gif, 11.gif, 12.gif, …, 52.gif,

файл bank.jpg фона экрана, также показанного выше,

файл рисунка загрузки loading.gif, который появляется в начале и исчезает после окончания загрузки в программу всех графических и звуковых файлов

Добавляем в проект звуковой файл pj_bg_noise. wav (для имитации нервного шума групп поддержки игрока и Банкомёта) по стандартной схеме, а именно: выполняем правый щелчок по имени этой папки, в контекстном меню выбираем Add, Existing Item, в панели Add Existing Item в окне «Files of type» выбираем «All Files», в центральном окне находим и выделяем имя файла (загруженного, например, из Интернета или из указанной в списке литературы ссылки) и щёлкаем кнопку Add (или дважды щёлкаем по имени файла). В панели Solution Explorer мы увидим этот файл.

Аналогично добавляем в эту папку остальные звуковые файлы:

pj_busted. wav – звук неудовольствия группы поддержки при переборе карт игроком;

pj_claps. wav – звук аплодисментов группы поддержки при выигрыше игрока на данной сдаче карт;

pj_pj. wav – радостный шум группы поддержки и голос за кадром, произносящий PocketJack при выигрыше игрока на данной сдаче карт.

Напомним, что добавлять в проект указанные выше файлы можно как по одному, так и все сразу, и выбирать значение Embedded Resource можно как для одного файла, так

и для всех файлов сразу (после их выделения или только одной мышью, или мышью с нажатой клавишей Shift – для выделения всех соседних файлов после щелчка только первого и последнего файлов, или мышью с нажатой клавишей Ctrl – для выделения всех файлов в различных местах).

В данном проекте, DirectX не применяется.

1.19. Код программы

Открываем файл Form1.cs (например, по схеме: File, Open, File) и в самом верху импортируем пространства имён для управления соответствующими классами:

```
using System.Reflection; //Namespace for class Assembly.
using System.Media; //Namespace for class SoundPlayer.
using System.IO; //Namespace for class Stream.
```

В классе Form1 нашего проекта записываем следующие переменные и методы.

Листинг 1.1. Переменные и методы.

```
//Shoe of cards:
CardShoe shoe;
CardHand playerHand = new CardHand ();
Card dealerHoleCard;
CardHand dealerHand = new CardHand ();
//Bank of a game:
Pot pot;
//We declare an object for a help form:
Form2 helpForm;
Image loadingImage = null;
Image bankImage = null;
```

```
public enum GameMode
{
LoadingDisplay,
PlacingBets,
PlayerActive,
PlayerWon,
PlayerBust,
PocketJack,
DealerActive,
DealerWon,
DealerBust,
Push
}
GameMode modeValue;
GameMode mode
{
get
{
return modeValue;
}
set
{
switch (value)
{
case GameMode. LoadingDisplay:
BetMinusToolStripMenuItem1.Enabled = false;
BetPlusToolStripMenuItem. Enabled = false;
HitMeToolStripMenuItem. Enabled = false;
StayToolStripMenuItem. Text = «Point»;
```

```
StayToolStripMenuItem. Enabled = false;
MenuToolStripMenuItem. Text = «21»;
MenuToolStripMenuItem. Enabled = false;
break;
case GameMode.PlacingBets:
BetMinusToolStripMenuItem1.Enabled = true;
BetPlusToolStripMenuItem. Enabled = true;
HitMeToolStripMenuItem. Enabled = false;
StayToolStripMenuItem. Text = «Себе»;
StayToolStripMenuItem. Enabled = true;
MenuToolStripMenuItem. Text = «Menu»;
MenuToolStripMenuItem. Enabled = true;
break;
case GameMode.PlayerActive:
BetMinusToolStripMenuItem1.Enabled = false;
BetPlusToolStripMenuItem. Enabled = false;
HitMeToolStripMenuItem. Enabled = true;
StayToolStripMenuItem. Text = «Take a card to yourself»;
StayToolStripMenuItem. Enabled = true;
MenuToolStripMenuItem. Text = «Menu»;
MenuToolStripMenuItem. Enabled = true;
//Disturbing noise of the support groups
//surrounding the player:
//not single, but continuous Looping;
Assembly a = Assembly.GetExecutingAssembly ();
Stream s =
a.GetManifestResourceStream (
«PocketJack. pj_bg_noise. wav»);
SoundPlayer player = new SoundPlayer (s);
```

```
player.PlayLooping ();
break;
case GameMode.PlayerWon:
BetMinusToolStripMenuItem1.Enabled = false;
BetPlusToolStripMenuItem. Enabled = false;
HitMeToolStripMenuItem. Enabled = false;
StayToolStripMenuItem. Text = «Distribution of cards»;
StayToolStripMenuItem. Enabled = true;
MenuToolStripMenuItem. Text = «Menu»;
MenuToolStripMenuItem. Enabled = true;
//An applause in our address for a prize in cards:
Assembly a1 = Assembly.GetExecutingAssembly ();
Stream s1 =
a1.GetManifestResourceStream (
«PocketJack. pj_claps. wav»);
SoundPlayer player1 = new SoundPlayer (s1);
player1.Play ();
break;
case GameMode.PlayerBust:
BetMinusToolStripMenuItem1.Enabled = false;
BetPlusToolStripMenuItem. Enabled = false;
HitMeToolStripMenuItem. Enabled = false;
StayToolStripMenuItem. Text = «Distribution of cards»;
StayToolStripMenuItem. Enabled = true;
MenuToolStripMenuItem. Text = «Menu»;
MenuToolStripMenuItem. Enabled = true;
//Discontent of support group with our exceeding of
//cards:
Assembly a2 = Assembly.GetExecutingAssembly ();
```

```
Stream s2 =
a2.GetManifestResourceStream (
«PocketJack. pj_busted. wav»);
SoundPlayer player2 = new SoundPlayer (s2);
player2.Play ();
break;
case GameMode. PocketJack:
BetMinusToolStripMenuItem1.Enabled = false;
BetPlusToolStripMenuItem. Enabled = false;
HitMeToolStripMenuItem. Enabled = false;
StayToolStripMenuItem. Text = «Distribution of cards»;
StayToolStripMenuItem. Enabled = true;
MenuToolStripMenuItem. Text = «Menu»;
MenuToolStripMenuItem. Enabled = true;
//A victorious tune after Pocket Jack with 21 points:
Assembly a3 = Assembly.GetExecutingAssembly ();
Stream s3 =
a3.GetManifestResourceStream (
«PocketJack. pj_pj. wav»);
SoundPlayer player3 = new SoundPlayer (s3);
player3.Play ();
break;
case GameMode.DealerActive:
BetMinusToolStripMenuItem1.Enabled = false;
BetPlusToolStripMenuItem. Enabled = false;
HitMeToolStripMenuItem. Enabled = false;
StayToolStripMenuItem. Text = «Distribution of cards»;
StayToolStripMenuItem. Enabled = false;
MenuToolStripMenuItem. Text = «Menu»;
```

```
MenuToolStripMenuItem. Enabled = true;
break;
case GameMode.DealerWon:
BetMinusToolStripMenuItem1.Enabled = false;
BetPlusToolStripMenuItem. Enabled = false;
HitMeToolStripMenuItem. Enabled = false;
StayToolStripMenuItem. Text = «Distribution of cards»;
StayToolStripMenuItem. Enabled = true;
MenuToolStripMenuItem. Text = «Menu»;
MenuToolStripMenuItem. Enabled = true;
//Discontent of support group with our exceeding of
//cards:
Assembly a4 = Assembly.GetExecutingAssembly ();
Stream s4 =
a4.GetManifestResourceStream (
«PocketJack. pj_busted. wav»);
SoundPlayer player4 = new SoundPlayer (s4);
player4.Play ();
break;
case GameMode.DealerBust:
BetMinusToolStripMenuItem1.Enabled = false;
BetPlusToolStripMenuItem. Enabled = false;
HitMeToolStripMenuItem. Enabled = false;
StayToolStripMenuItem. Text = «Distribution of cards»;
StayToolStripMenuItem. Enabled = true;
MenuToolStripMenuItem. Text = «Menu»;
MenuToolStripMenuItem. Enabled = true;
//An applause in our address for a prize in cards:
Assembly a5 = Assembly.GetExecutingAssembly ();
```

```
Stream s5 =
a5.GetManifestResourceStream (
«PocketJack. pj_claps. wav»);
SoundPlayer player5 = new SoundPlayer (s5);
player5.Play ();
break;
case GameMode. Push:
BetMinusToolStripMenuItem1.Enabled = false;
BetPlusToolStripMenuItem. Enabled = false;
HitMeToolStripMenuItem. Enabled = false;
StayToolStripMenuItem. Text = «Distribution of cards»;
StayToolStripMenuItem. Enabled = true;
MenuToolStripMenuItem. Text = «Menu»;
MenuToolStripMenuItem. Enabled = true;
break;
}
modeValue = value;
this.Invalidate ();
}
}
//We load the game objects:
public void init ()
{
System.Reflection.Assembly asm =
System.Reflection.Assembly.GetExecutingAssembly ();
loadingImage = new
Bitmap(asm.GetManifestResourceStream (
"PocketJack.images.loading.gif»));
bankImage = new
```

```
Bitmap(asm.GetManifestResourceStream (
"PocketJack.images.bank.jpg»));
mode = GameMode. LoadingDisplay;
this.Visible = true;
this.Refresh ();
pot = new Pot ();
//We create also initsializuy the help Form2 form:
helpForm = new Form2 ();
}
void startGame ()
{
shoe = new CardShoe ();
//We comment in the original,
//since this line – only for testing of a game:
//shoe =
//new CardShoe (new byte [] {2, 14, 11, 25, 10, 7, 6, 5});
pot.ResetPot ();
mode = GameMode.PlacingBets;
}
void startHand ()
{
mode = GameMode.PlacingBets;
}
void showPot ()
{
this. Text =
«Bank: " + pot.PotValue.ToString () + " Bet: " +
pot.BetValue.ToString ();
}
```

```
void startPlay ()
{
//We commented out in the original:
//pot.DoPlaceBet ();
//We write in the original:
if (mode == GameMode.PlayerBust && mode ==
GameMode.DealerWon)
pot.DoPlaceBet ();
showPot ();
// clear the hands
playerHand.Clear ();
dealerHand.Clear ();
// deal the face down hole card
dealerHoleCard = shoe.DealCard ();
dealerHoleCard. FaceUp = false;
dealerHand.Add (dealerHoleCard);
// deal the first player card
playerHand.Add(shoe.DealCard ());
// deal the second dealer card (face up)
dealerHand.Add(shoe.DealCard ());
// deal the second player card
playerHand.Add(shoe.DealCard ());
if ((dealerHand. BlackJackScoreHand () == 21) &&
(playerHand. BlackJackScoreHand ()!= 21))
{
//We write in the original:
pot.DoPlaceBet ();
pot. HouseWins ();
showPot ();
```

```
//Discontent of support group with our exceeding of
//cards:
Assembly a2 = Assembly.GetExecutingAssembly ();
Stream s2 =
a2.GetManifestResourceStream («PocketJack. pj_busted.
wav»);
SoundPlayer player2 = new SoundPlayer (s2);
player2.Play ();
dealerHoleCard. FaceUp = true;
mode = GameMode.DealerWon;
return;
}
if ((playerHand. BlackJackScoreHand () == 21) &&
(dealerHand. BlackJackScoreHand ()!= 21))
{
pot.PlayerWins ();
showPot ();
dealerHoleCard. FaceUp = true;
mode = GameMode. PocketJack;
return;
}
if ((playerHand. BlackJackScoreHand () == 21) &&
(dealerHand. BlackJackScoreHand () == 21))
{
pot.DoPushBet ();
showPot ();
dealerHoleCard. FaceUp = true;
mode = GameMode. Push;
return;
```

```
}
mode = GameMode.PlayerActive;
}
Font messageFont = new
Font(FontFamily.GenericSansSerif, 20,
FontStyle.Regular);
void paintForm (Graphics g)
{
switch (mode)
{
case GameMode. LoadingDisplay:
//We draw all images below the menu:
g. DrawImage (
bankImage, 0, StayToolStripMenuItem. Height);
g. DrawImage (
loadingImage, 0, StayToolStripMenuItem. Height +60);
break;
case GameMode.PlacingBets:
g. DrawImage (bankImage, 0, StayToolStripMenuItem.
Height);
Utilities.BigText («Bank: " + pot.PotValue.ToString (),
10, 40, Color. Black,
Color. Yellow, messageFont, g);
Utilities.BigText («Bet: " +
pot.BetValue.ToString (), 10, 80, Color. Black,
Color. Yellow, messageFont, g);
break;
case GameMode.PlayerActive:
dealerHand. DrawHand (g, 10, 30, 80, 25);
```

```
playerHand. DrawHand (g, 10, 135, 20, 25);
Utilities.BigText (playerHand. BlackJackScoreHand ().
ToString (), 140, 150, Color. Black,
Color. Yellow, messageFont, g);
break;
case GameMode.PlayerWon:
case GameMode. PocketJack:
dealerHand. DrawHand (g, 10, 30, 20, 25);
playerHand. DrawHand (g, 10, 135, 20, 25);
Utilities.BigText (dealerHand. BlackJackScoreHand ().
ToString (), 140, 45, Color. Black,
Color. Yellow, messageFont, g);
Utilities.BigText (playerHand. BlackJackScoreHand ().
ToString (), 140, 150, Color. Black,
Color. Yellow, messageFont, g);
Utilities.BigText («Вы выиграли!»,
20, 80, Color. Black, Color. Yellow, messageFont, g);
break;
case GameMode.PlayerBust:
dealerHand. DrawHand (g, 10, 30, 80, 25);
playerHand. DrawHand (g, 10, 135, 20, 25);
Utilities.BigText (playerHand. BlackJackScoreHand ().
ToString (), 140, 150, Color. Black,
Color. Yellow, messageFont, g);
Utilities.BigText («Вы перебрали.»,
20, 80, Color. Black, Color.Red, messageFont, g);
break;
case GameMode.DealerActive:
dealerHand. DrawHand (g, 10, 30, 20, 25);
```

```
playerHand. DrawHand (g, 10, 135, 20, 25);
Utilities.BigText (dealerHand. BlackJackScoreHand ().
ToString (), 140, 45, Color. Black,
Color. Yellow, messageFont, g);
Utilities.BigText (playerHand. BlackJackScoreHand ().
ToString (), 140, 150, Color. Black,
Color. Yellow, messageFont, g);
break;
case GameMode.DealerWon:
dealerHand. DrawHand (g, 10, 30, 20, 25);
playerHand. DrawHand (g, 10, 135, 20, 25);
Utilities.BigText (dealerHand. BlackJackScoreHand ().
ToString (), 140, 45, Color. Black,
Color. Yellow, messageFont, g);
Utilities.BigText (playerHand. BlackJackScoreHand ().
ToString (), 140, 150, Color. Black,
Color. Yellow, messageFont, g);
Utilities.BigText («Вы потеряли.»,
20, 80, Color. Black, Color.Red, messageFont, g);
break;
case GameMode.DealerBust:
dealerHand. DrawHand (g, 10, 30, 20, 25);
playerHand. DrawHand (g, 10, 135, 20, 25);
Utilities.BigText (dealerHand. BlackJackScoreHand ().
ToString (), 140, 45, Color. Black,
Color. Yellow, messageFont, g);
Utilities.BigText (playerHand. BlackJackScoreHand ().
ToString (), 140, 150, Color. Black,
Color. Yellow, messageFont, g);
```

```
Utilities.BigText («Dealer Bust»,
20, 80, Color. Black, Color. Yellow, messageFont, g);
break;
case GameMode. Push:
dealerHand. DrawHand (g, 10, 30, 20, 25);
playerHand. DrawHand (g, 10, 135, 20, 25);
Utilities.BigText (dealerHand. BlackJackScoreHand ().
ToString (), 140, 45, Color. Black,
Color. Yellow, messageFont, g);
Utilities.BigText (playerHand. BlackJackScoreHand ().
ToString (), 140, 150, Color. Black,
Color. Yellow, messageFont, g);
Utilities.BigText («Вы выиграли.»,
20, 80, Color. Black, Color. Yellow, messageFont, g);
break;
}
}
void playerHits ()
{
if (playerHand. BlackJackScoreHand () <21)
{
playerHand.Add(shoe.DealCard ());
if (playerHand. BlackJackScoreHand ()> 21)
{
//We write in the original:
pot.DoPlaceBet ();
pot. HouseWins ();
showPot ();
mode = GameMode.PlayerBust;
```

```
}
this.Invalidate ();
}
}
void playerStays ()
{
dealerHoleCard. FaceUp = true;
mode = GameMode.DealerActive;
this.Refresh ();
System.Threading.Thread.Sleep (750);
while (dealerHand. BlackJackScoreHand () <17)
{
dealerHand.Add(shoe.DealCard ());
this.Refresh ();
System.Threading.Thread.Sleep (750);
}
if (dealerHand. BlackJackScoreHand ()> 21)
{
mode = GameMode.DealerBust;
pot.PlayerWins ();
showPot ();
return;
}
if (playerHand. BlackJackScoreHand ()>
dealerHand. BlackJackScoreHand ())
{
mode = GameMode.PlayerWon;
pot.PlayerWins ();
showPot ();
```

```
return;
}
if (playerHand. BlackJackScoreHand () <
dealerHand. BlackJackScoreHand ())
{
mode = GameMode.DealerWon;
//We write in the original:
pot.DoPlaceBet ();
pot. HouseWins ();
showPot ();
return;
}
if (playerHand. BlackJackScoreHand () ==
dealerHand. BlackJackScoreHand ())
{
mode = GameMode. Push;
pot.DoPushBet ();
showPot ();
return;
}
}
void doLeftMenuKey ()
{
switch (mode)
{
case GameMode. LoadingDisplay:
break;
case GameMode.PlacingBets:
startPlay ();
```

```
break;
case GameMode.PlayerActive:
playerStays ();
break;
case GameMode. PocketJack:
case GameMode.PlayerWon:
case GameMode.PlayerBust:
case GameMode.DealerActive:
case GameMode.DealerWon:
case GameMode.DealerBust:
case GameMode. Push:
startHand ();
break;
}
}
void doEnter ()
{
switch (mode)
{
case GameMode. LoadingDisplay:
break;
case GameMode.PlacingBets:
startPlay ();
break;
case GameMode.PlayerActive:
playerHits ();
break;
case GameMode. PocketJack:
case GameMode.PlayerWon:
```

```
case GameMode.PlayerBust:
case GameMode.DealerActive:
case GameMode.DealerWon:
case GameMode.DealerBust:
case GameMode. Push:
startHand ();
break;
}
}
void doUp ()
{
switch (mode)
{
case GameMode.PlacingBets:
pot.DoIncreaseBet ();
showPot ();
this.Invalidate ();
break;
}
}

void doDown ()
{
switch (mode)
{
case GameMode.PlacingBets:
pot.DoDecreaseBet ();
showPot ();
this.Invalidate ();
```

```
    break;
    }
    }
    void showHelp ()
    {
    helpForm.ShowDialog ();

}
```

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 1.2. Метод для загрузки файлов объектов.

```
    private void Form1_Load (object sender, EventArgs e)
    {
    //We load the game objects:
    init ();
    startGame ();

}
```

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. Напомним, что другие варианты вывода изоб-

ражения, например, на элемент управления PictureBox и после щелчка по какому-либо элементу управления уже приводились ранее.

Листинг 1.3. Метод для рисования изображений.

```
private void Form1_Paint (object sender, PaintEventArgs e)
{
paintForm(e.Graphics);
}
```

Для задания режимов и управления игрой воспользуемся каким-либо элементом управления или компонентом. Как и выше, с панели инструментов Toolbox переносим на форму компонент MenuStrip и щёлкаем по нему (ниже формы в режиме проектирования). На форме Form1 появляются окна с надписью Type Here, в которые записываем команды (по второму варианту, можно записывать в панели Properties в свойстве Text): Stay, Menu, Exit, HitMe, BetPlus, BetMinus, Sound, Help, рис. 1.30. Для команды, слева от которой поставлен флажок, в панели Properties для этой команды значение Checked следует задать как True.

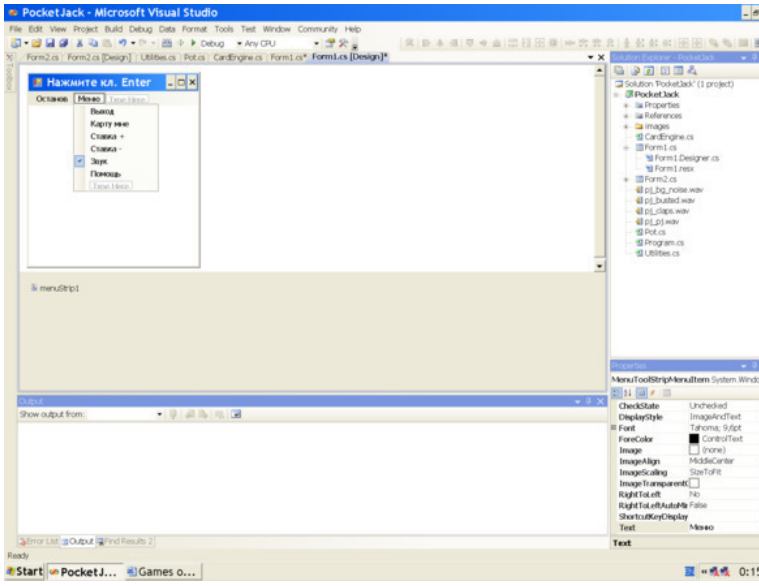


Рис. 1.30. Команды элемента управления MenuStrip.

В режиме редактирования дважды щёлкаем по левой команде Stay (для приостановки игры). Появившийся шаблон метода после записи строки принимает следующий вид.

Листинг 1.4. Метод для приостановки игры.

```
private void StayToolStripMenuItem_Click (object sender, EventArgs e)
```

```
{
doLeftMenuKey ();
}
```

Аналогично дважды щёлкаем по команде Exit. Появившийся шаблон метода после записи нашего кода принимает следующий вид.

Листинг 1.5. Метод для окончания игры.

```
private void ExitToolStripMenuItem_Click (object sender,
EventArgs e)
{
Application. Exit ();
}
```

Аналогично дважды щёлкаем по команде «Hit Me». Появившийся шаблон метода после записи нашего кода принимает следующий вид.

Листинг 1.6. Метод для выдачи карты игроку.

```
private void HitMeToolStripMenuItem_Click (object sender,
EventArgs e)
{
playerHits ();
}
```

Аналогично дважды щёлкаем по команде «BetPlus». Появившийся шаблон метода после записи нашего кода принимает следующий вид.

Листинг 1.7. Метод для увеличения Ставки.

```
private void BetPlusToolStripMenuItem_Click (object sender, EventArgs e)
{
doUp ();
}
```

Аналогично дважды щёлкаем по команде «Ставка -" (Bet -). Появившийся шаблон метода после записи нашего кода принимает следующий вид.

Листинг 1.8. Метод для уменьшения Ставки.

```
private void BetMinusToolStripMenuItem1_Click (object sender, EventArgs e)
{
doDown ();
}
```

Методика приостановки и возобновления звука при по-

мощи установки и удаления флажка в команде Звук (Sound) была описана выше. В данной игре, звук включён постоянно.

Аналогично дважды щёлкаем по команде Help. Появившийся шаблон метода после записи нашего кода принимает следующий вид.

Листинг 1.9. Метод для вывода справочной формы.

```
private void HelpToolStripMenuItem_Click (object sender,
EventArgs e)
{
showHelp ();
}
```

Теперь программа должна управлять карточной игрой, используя любые клавиши, например, верхнюю (Up) и нижнюю (Down), левую (Left) и правую (Right) клавиши со стрелками, а также клавишу Enter (для начальной сдачи карт игроку и Банкомёту и последующей сдачи по одной карте игроку). В панели Properties (для формы Form1) на вкладке Events дважды щёлкаем по имени события KeyDown. Появившийся шаблон обработки нажатий всех клавиш после записи нашего кода для нажатий только трех клавиш (относящимся к тематике данной главы) принимает следующий вид.

Листинг 1.10. Метод для обработки нажатий клавиш.

```
private void Form1_KeyDown (object sender,
KeyEventArgs e)
{
if ((e.KeyCode == System.Windows.Forms.Keys. Up))
{
doUp ();
e. Handled = true;
}
if ((e.KeyCode == System.Windows.Forms.Keys.Down))
{
doDown ();
e. Handled = true;

    }
    if ((e.KeyCode == System.Windows.Forms.Keys. Enter))
    {
    //Набираем себе карты:
    doEnter ();

}
}
```

Мы закончили написание программы в главный класс Form1 (для формы Form1 с пользовательским интерфейсом игры). В этом проекте движок игры (Engine Game) находится не в файле Form1.cs (как обычно бывает), а в следующем

файле CardEngine. cs.

Теперь в наш проект добавляем новые файлы (для программирования соответствующих игровых действий) по следующей схеме.

В панели Solution Explorer выполняем правый щелчок по имени проекта и в контекстном меню выбираем Add, New Item. В панели Add New Item выделяем шаблон Code File, в окне Name записываем имя нового файла с расширением *.cs и щёлкаем кнопку Add. В проект (и в панель Solution Explorer) добавляется этот файл, открывается пустое окно редактирования кода, в которое записываем следующий код.

Листинг 1.11. Новый файл CardEngine. cs.

```
using System;
using System.Collections;
using System. Drawing;
namespace PocketJack
{
/// <summary>
/// Provides the behaviours required to manage and draw
cards
/// </summary>
public class Card
{
/// <summary>
/// The number of the card, in the range 1 to 52
/// </summary>
```

```
public byte CardNo;
/// <summary>
/// Indicates if the card is to be drawn face up.
/// True by default.
/// </summary>
public bool FaceUp = true;
/// <summary>
/// The images of the cards. Stored for all the cards.
/// The image with number 0 is the
/// back pattern of the card
/// </summary>
static private Image [] cardImages = new Bitmap [53];
/// <summary>
/// The attribute to be used when drawing the card
/// to implement transpancy
/// </summary>
static public System.Drawing.Imaging.ImageAttributes
cardAttributes;
/// <summary>
/// Used when loading card images prior to drawing
/// </summary>
static private System.Reflection.Assembly execAssem;
/// <summary>
/// Sets up the color and attribute values.
/// </summary>
static Card ()
{
cardAttributes =
new System.Drawing.Imaging.ImageAttributes ();
```

```
cardAttributes.SetColorKey(Color.Green, Color.Green);
execAssem =
System.Reflection.Assembly.GetExecutingAssembly ();
}
/// <summary>
/// Scores for each of the cards in a suit
/// </summary>
static private byte [] scores =
new byte [] {11, //ace
2,3,4,5,6,7,8,9,10, //spot cards
10,10,10}; //jack, queen, king
/// <summary>
/// Picture information for each card in a suit
/// </summary>
static private bool [] isPicture =
new bool [] {false, //ace
false, false, false, false, false, false,
false, false, false, //spot cards
true, true, true}; //jack, queen, king
/// <summary>
/// Names of the suits, in the order that of the suits
/// in the number sequence
/// </summary>
static private string [] suitNames =
new string [] {«club», «diamond», «heart», «spade»};
/// <summary>
/// Names of individual cards, in the order of the cards
/// in a suit
/// </summary>
```

```
static private string [] valueNames =
new string [] {«Ace», «Deuce», «Three», «Four», «Five»,
«Six»,
«Seven», «Eight», «Nine», «Ten», «Jack», «Queen»,
«King»};
/// <summary>
/// Returns the value in points of a given card,
/// according to BlackJack rules
/// </summary>
public int BlackJackScore
{
get
{
return scores [(CardNo – 1) % 13];
}
}
/// <summary>
/// Returns true if the card is a picture
/// (i.e. jack, queen or king)
/// </summary>
public bool IsPicture
{
get
{
return isPicture [(CardNo – 1) % 13];
}
}
/// <summary>
/// Returns text of the suit of this card
```

```
/// </summary>
public string Suit
{
get
{
return suitNames [(CardNo – 1) / 13];
}
}
/// <summary>
/// Returns the text of the value of this card
/// </summary>
public string ValueName
{
get
{
return valueNames [(CardNo – 1) % 13];
}
}
/// <summary>
/// Returns true if this is a red card
/// </summary>
public bool Red
{
get
{
int suit = (CardNo – 1) / 13;
return ((suit == 1) || (suit == 2));
}
}
```

```
/// <summary>
/// Returns true if this is a black card
/// </summary>
public bool Black
{
get
{
return! Red;
}
}
/// <summary>
/// Returns an image which can be used to draw this card
/// </summary>
public Image CardImage
{
get
{
int dispNo = CardNo;
if (!FaceUp)
{
dispNo = 0;
}
if (cardImages [dispNo] == null)
{
cardImages [dispNo] = new Bitmap (
execAssem.GetManifestResourceStream (
@"PocketJack.images.» + dispNo + @".gif»));
}
return cardImages [dispNo];
```

```
}
}
/// <summary>
/// Constructs a card with a partiuclar number
/// </summary>
/// <param name=«cardNo»> number of the card
/// in the range 1 to 52 </param>
/// <param name=«faceUp»> true if the card
/// is to be drawn face up </param>
public Card (byte cardNo, bool faceUp)
{
CardNo = cardNo;
FaceUp = faceUp;
}
/// <summary>
/// Constructs a face up card with that number
/// </summary>
/// <param name=«cardNo»> </param>
public Card (byte cardNo)
: this (cardNo, true)
{
}
/// <summary>
/// String description of the card
/// </summary>
/// <returns> the name and suit of the card </returns>
public override string ToString ()
{
return ValueName + " of " + Suit;
```

```
}
}
/// <summary>
/// Provides a container for a number of cards.
/// May be used to draw the cards and compute their score.
/// </summary>
public class CardHand: ArrayList
{
/// <summary>
/// Used as a destination of teh draw action
/// </summary>
private static Rectangle drawRect;
/// <summary>
/// Draws the hand on the graphics.
/// </summary>
/// <param name=«g»> graphics to draw with </param>
/// <param name=«startx»> left edge of first card </param>
/// <param name=«starty»> top of first card </param>
/// <param name=«gapx»> x gap between each card </
param>
/// <param name=«gapy»> y gap between each card </
param>
public void DrawHand (Graphics g, int startx, int starty,
int gapx, int gapy)
{
drawRect. X = startx;
drawRect. Y = starty;
foreach (Card card in this)
{
```

```
drawRect. Width = card.CardImage. Width;
drawRect. Height = card.CardImage. Height;
g. DrawImage (
card.CardImage, // Image
drawRect, // destination rectange
0, // srcX
0, // srcY
card.CardImage. Width, // srcWidth
card.CardImage. Height, // srcHeight
GraphicsUnit. Pixel, // srcUnit
Card.cardAttributes); // ImageAttributes
drawRect. X += gapx;
drawRect. Y += gapy;
}
}
/// <summary>
/// Computes the score of the hand
/// </summary>
/// <returns> the value of the score </returns>
public int BlackJackScoreHand ()
{
int score = 0;
int aces = 0;
foreach (Card card in this)
{
score += card. BlackJackScore;
if (card. BlackJackScore == 11)
{
aces++;
```

```
}
}
while ((score> 21) && (aces> 0))
{
score -= 10;
aces – ;
}
return score;
}
}
/// <summary>
/// Contains a number of card decks
/// which can be dealt one at a time.
/// </summary>
public class CardShoe
{
private int noOfDecks = 1;
private byte [] decks;
private int nextCard;
private bool testShoe = false;
/// <summary>
/// True if the deck is «stacked»,
/// i.e. was created from a byte array
/// </summary>
public bool TestShoe
{
get
{
return testShoe;
```

```
}
}
private void makeShoe ()
{
decks = new byte [noOfDecks * 52];
int cardPos = 0;
for (int i = 0; i <noOfDecks; i++)
{
for (byte j = 1; j <53; j++)
{
decks [cardPos] = j;
cardPos++;
}
}
nextCard = 0;
}
private void shuffleShoe ()
{
if (!testShoe)
{
System. Random rand = new Random ();
byte swap;
int p1, p2;
for (int i = 0; i <decks. Length; i++)
{
p1 = rand.Next (decks. Length);
p2 = rand.Next (decks. Length);
swap = decks [p1];
decks [p1] = decks [p2];
```

```
decks [p2] = swap;
}
}
nextCard = 0;
}
/// <summary>
/// Gets the next card number from the deck
/// </summary>
/// <returns> The number of the next card </returns>
public byte NextCardNo ()
{
if (nextCard == decks. Length)
{
shuffleShoe ();
}
return decks [nextCard++];
}
/// <summary>
/// Gets the next card from the deck.
/// </summary>
/// <returns> A new instance of the card </returns>
public Card DealCard ()
{
return new Card (NextCardNo ());
}
/// <summary>
/// Constructs a shoe containing a number of decks
/// </summary>
/// <param name=«noOfDecks»> </param>
```

```
    public CardShoe (int noOfDecks)
    {
    this.noOfDecks = noOfDecks;
    makeShoe ();
    shuffleShoe ();
    testShoe = false;
    }
    /// <summary>
    /// Constructs a shoe containing a single deck
    /// </summary>
    public CardShoe ()
    : this (1)
    {
    }

    /// <summary>
    /// Creates a stacked deck for test purposes.
    /// </summary>
    /// <param name=«stackedDeck»> array of bytes </param>
    public CardShoe (byte [] stackedDeck)
    {
    decks = stackedDeck;
    testShoe = true;

}
}
}
```

В панели Solution Explorer выполняем правый щелчок по имени проекта и в контекстном меню выбираем Add, New Item. В панели Add New Item выделяем шаблон Code File, в окне Name записываем имя нового файла с расширением *.cs и щёлкаем кнопку Add. В проект (и в панель Solution Explorer) добавляется этот файл, открывается пустое окно редактирования кода, в которое записываем следующий код.

Листинг 1.12. Новый файл Pot. cs.

```
using System;
namespace PocketJack
{
/// <summary>
/// Summary description for Betting.
/// </summary>
public class Pot
{
private int betValueChangeValue;
private int betValue;
private int potValue;
private const int INITIAL_POT_VALUE = 500;
private const int INITIAL_BET_CHANGE_VALUE = 5;
public int BetValue
{
get
{
return betValue;
```

```
}
}
public int PotValue
{
get
{
return potValue;
}
}
public void ResetPot ()
{
betValueChangeValue                                    =
INITIAL_BET_CHANGE_VALUE;
betValue = INITIAL_BET_CHANGE_VALUE;
potValue = INITIAL_POT_VALUE;
}
public void CheckPot ()
{
if (betValue> potValue)
{
if (System.Windows.Forms.MessageBox.Show (
«Insufficient funds for the bet.» +
«Do you want to reload the pot?»,
«Bank»,
System.Windows.Forms.MessageBoxButtons. YesNo,
System.Windows.Forms.MessageBoxIcon. Question,
System.Windows.Forms.
MessageBoxDefaultButton. Button1) ==
System.Windows.Forms. DialogResult. Yes)
```

```
{
ResetPot ();
}
else
{
betValue = potValue;
}
}
}
public void DoIncreaseBet ()
{
betValue = betValue + betValueChangeValue;
CheckPot ();
}
public void DoDecreaseBet ()
{
if (betValue> = betValueChangeValue)
{
betValue = betValue – betValueChangeValue;
}
}
public void PlayerWins ()
{
// win back 2 * our stake
potValue = potValue + betValue;
//potValue = potValue + betValue; //We commented out.
}
public void HouseWins ()
{
```

```
    CheckPot ();
    }
    public void DoPushBet ()
    {
    // put the betValue back in the potValue
    potValue = potValue + betValue;
    }
    public void DoPlaceBet ()
    {
    potValue = potValue – betValue;
    }
    public Pot ()
    {
    ResetPot ();

}
}
}
```

После этого добавления в панели Solution Explorer должны быть файлы, показанные на рисунке выше. Дважды щёлкая по имени файла, любой файл можно открыть, изучить и редактировать.

В этих файлах использованы XML-комментарии (XML comment), где XML – Extensible Markup Language – расширяемый язык разметки, типа:

```
/// <summary>
/// Description of a variable:
```

/// </summary>
который состоит из начального тэга (start tag):
/// <summary>
и конечного тэга (end tag):
/// </summary>
между которыми записывается сам комментарий:
/// Описание переменной:
/// Description of a variable:

В отличие от обычных комментариев после двойного слеша // или между двух символов /*…*/, в любом месте программы при наведении указателя мыши на переменную с XML-комментарием появляется подсказка, в которой имеется не только тип и класс переменной (как для переменной с обычным комментарием или вообще без комментария), но также имеется и её описание на любом языке, в том числе на русском языке, которое мы записали между начальным и конечным тэгами. XML-комментарии имеют и другие преимущества, описанные в специальной литературе.

К недостатку XML-комментария, относятся две дополнительные строки начального и конечного тэгов, увеличивающие (и без них) большое количество строк в программе.

1.20. Методика рисования текстов на основе класса

Для рисования текстов на экране при помощи универ-

сального (для многих других игр) класса Utilities,

В панели Solution Explorer выполняем правый щелчок по имени проекта и в контекстном меню выбираем Add, New Item. В панели Add New Item выделяем шаблон Code File, в окне Name записываем имя нового файла с расширением *.cs и щёлкаем кнопку Add. В проект (и в панель Solution Explorer) добавляется этот файл, открывается пустое окно редактирования кода, в которое записываем следующий код.

Листинг 1.13. Файл Utilities. cs.

```
using System. Drawing;
namespace PocketJack
{
public class Utilities
{
static private SolidBrush messageBrush =
new SolidBrush (Color. Black);
public static void BigText (string message, int x, int y,
Color back, Color fore, Font messageFont, Graphics g)
{
int i;
messageBrush.Color = back;
for (i = 1; i <3; i++)
{
g. DrawString (message, messageFont, messageBrush,
x – i, y – i);
g. DrawString (message, messageFont, messageBrush,
```

```
    x – i, y + i);
    g. DrawString (message, messageFont, messageBrush,
    x + i, y – i);
    g. DrawString (message, messageFont, messageBrush,
    x + i, y + i);
    }
    messageBrush.Color = fore;
    g. DrawString (message, messageFont, messageBrush, x, y);
    }
    public Utilities ()
    {
    // TODO: Add constructor logic here

}
}
}
```

Этот файл Utilities. cs мы будем использовать в нескольких приведённых далее играх для рисования текстов на экране, но там мы не будем приводить этот файл, для экономии места в книге, а будем давать только ссылку на этот параграф.

Сразу же здесь отметим, что для использования данного файла Utilities. cs в проекте с другим именем (отличным от имени данного проекта PocketJack), необходимо:

или в данном файле Utilities. cs вместо имени пространства имён (с именем проекта PocketJack) в строке:

namespace PocketJack

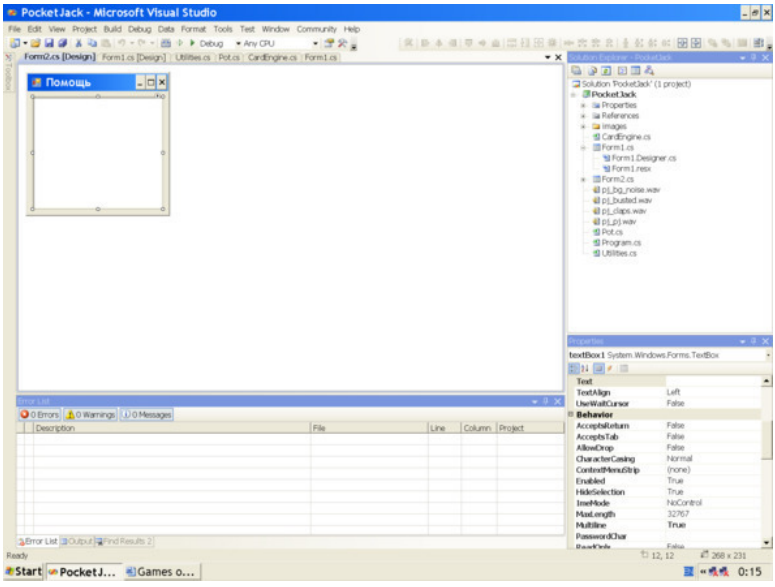
записать имя нового проекта,

или в новом проекте в файле, где используется ссылка на данный файл Utilities. cs, в верхней части импортировать (записать директиву) пространства имён PocketJack:

using PocketJack;

1.21. Методика добавления информации в справочные формы

Для ввода в проект новой (справочной) формы, по которой игрок будет изучать, например, правила игры, в меню Project выбираем Add Windows Form, в панели Add New Item оставляем заданные по умолчанию параметры и щёлкаем кнопку Add. В ответ Visual Studio выводит новую форму Form2 (рис. 1.31) и добавляет в панель Solution Explorer новый пункт Form2.cs.


PocketJack - Microsoft Visual Studio
File Edit View Project Build Debug Data Format Tools Test Window Community Help
Debug
Any CPU
Form2.cs [Design]
Form1.cs [Design]
Utilities.cs
Pot.cs
CardEngine.cs
Form1.cs
Помощь
Solution Explorer - PocketJack
Solution 'PocketJack' (1 project)
PocketJack
Properties
References
images
CardEngine.cs
Form1.cs
Form1.Designer.cs
Form1.resx
Form2.cs
pj_bg_noise.wav
pj_busted.wav
pj_claps.wav
pj_pj.wav
Pot.cs
Program.cs
Utilities.cs
Properties
textBox1 System.Windows.Forms.TextBox
Text
TextAlign Left
UseWaitCursor False
Behavior
AcceptsReturn False
AcceptsTab False
AllowDrop False
CharacterCasing Normal
ContextMenuStrip (none)
Enabled True
HideSelection True
ImeMode NoControl
MaxLength 32767
Multiline True
PasswordChar
Error List
0 Errors
0 Warnings
0 Messages
Description
File
Line
Column
Project
Error List
Output
Find Results 2
Ready
12, 12
268 x 231
Start
PocketJ...
Games o...
0:15

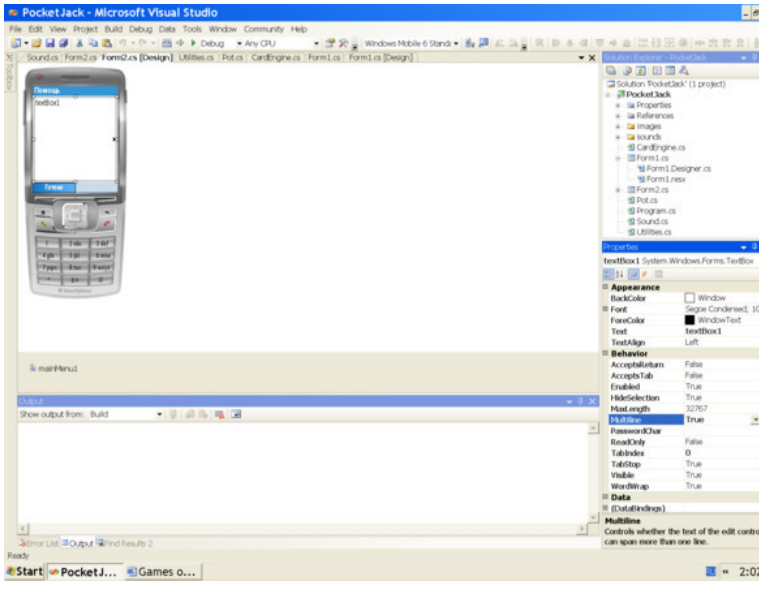


Рис. 1.31. Проектируем справочную форму. **Рис. 1.32** В свойстве Multiline выбираем True.

Аналогично, как первую, проектируем вторую форму и вводим элемент управления в виде окна TextBox. Чтобы в это окно можно было записать многострочный текст, в панели Properties (для этого элемента) в свойстве Multiline выбираем значение True (рис. 1.32).

По этой схеме можно добавлять и большее количество форм, сколько необходимо для каждого конкретного при-

ложения. Для закрытия второй формы можно воспользоваться каким-либо элементом управления или компонентом. А можно использовать уже автоматически размещенный на форме крестик Close, которым мы и будем пользоваться.

Теперь мы должны написать программу для второй формы Form2. Открываем (например, по схеме: File, Open, File) файл Form2.cs и в методе-конструкторе класса Form2 ниже следующих строк:

```
public Form2 ()
{
InitializeComponent ();
```

записываем следующий код для выдачи на экран на элемент управления TextBox справочной информации.

Листинг 1.14. Код для выдачи на экран справочной информации.

```
StringBuilder sbl;
sbl = new StringBuilder ();
sbl. Append («Правила игры в очко:\r\n\r\n\r\n»);
sbl. Append («Rules of the game in „point“: \r\n\r\n\r\n»);
sbl. Append («1) Вы являетесь игроком (player) и играете
" +
```

```
    sbl. Append («1) You are a player and play» +
    «один на один с банкомётом (dealer).\r\n\r\n»);
    «in private with a dealer.\r\n\r\n»);
    sbl. Append («2) Ваша цель состоит в том, чтобы иметь
в руке " +
    sbl. Append («2) Your purpose consists in having in a hand» +
    «карты с очками, как можно ближе к 21, но не превышая
21, " +
    «a card with points as it is possible closer to 21, " +
    «but without exceeding 21,» +
    «и больше, чем у банкомёта.\r\n\r\n»);
    «and more, than at dealer.\r\n\r\n»);

    sbl. Append («3) Первоначально у вас имеются две карты,
" +
    sbl. Append («3) Originally are available for you two cards,» +
    «вы видите сумму очков этих двух карт, и вы можете взять
" +
    «you see the sum of points of these two cards, " +
    «and you can take» +
    «дополнительные карты, нажимая кнопку Enter " +
    «the additional cards, pressing the Enter key» +
    «или выбирая команду «Карту мне» в Меню " +
    «or choosing the command „HitMe“ in the Menu» +
    «для элемента управления mainMenu1.\r\n\r\n»);
    «for the control mainMenu1.\r\n\r\n control»);
```

```
sbl. Append («4) Если общее количество очков " +
sbl. Append («4) If total quantity of points» +
«ваших карт превышает 21, " +
«of your cards exceeds 21,» +
«вы взяли лишние карты и теряете вашу ставку.\r\n\r\n»);
«you took excess cards and lose yours bet.\r\n\r\n»);
sbl. Append («5) Если банкомёт набрал такое же количество очков, " +
sbl. Append («5) If a dealer scored the same quantity of points,» +
«как и вы, побеждаете вы, " +
«as well as you, win you,» +
«и счёт увеличивается в вашу пользу.\r\n\r\n»);
«and the account increases in yours advantage.\r\n\r\n»);
sbl. Append («6) Значения очков каждой карты следующие:\r\n»);
sbl. Append («6) Values of points of each card following:\r\n»);
sbl. Append («Ace – A = 1 or 11; " +
«как 1-я, 2-я или 3-я карта – Туз даёт 11 очков; " +
«as the 1st, 2nd or 3rd card – Ace gives 11 points;" +
«например, с Валетом, Дамой и Королём Туз даёт 11 очков " +
«for example, with Jack, Gueen and King, Ace gives 11 points» +
«и в сумме 10+11=21 эти две карты называются
```

```
PocketJack, " +
    «and in the sum 10+11=21 these two cards are called " +
    «PocketJack,» +
    «который бьёт карты соперника, даже набравшие 21; " +
    «who covers the rival's cards, even gathered 21;" +
    «как 4-я и последующая карта – Туз даёт 1 очко;\r\n»);
    «as the 4th and subsequent card – Ace gives 1 point; \r\n»);
    sbl. Append («цифры на картах от 2 до 9 " +
    sbl. Append («Digits on cards from 2 to 9» +
    «означают очки этой карты;\r\n»);
    «mean the points of this card; \r\n»);
    sbl. Append («карта с числом 10, " +
    sbl. Append («a card with number 10,» +
    «Jack – J, " +
    «Queen – Q, " +
    «King – K = on 10 points.» +
    «\r\n\r\n»);
    sbl. Append («7) Если первые две карты у игрока или бан-
комёта " +
    sbl. Append («7) If the first two cards at player or dealer» +
    «набрали 21 очко, то они также " +
    «gathered 21 points, they also» +
    «бьют карты соперника, даже набравшие 21.\r\n\r\n»);
    «cover the rival's cards, even gathered 21.\r\n\r\n»);
    sbl. Append («8) Банкомёт сдаёт карты " +
    sbl. Append («8) Dealer hands over cards» +
```

```
    «с единственной колоды карт.\r\n\r\n»);
    «from the only shoe of cards.\r\n\r\n»);
    sbl. Append («9) Банкомёт будет сдавать себе карты, " +
    sbl. Append («9) Dealer will hand over itself cards,» +
    «пока не достигнет 17 или больше.\r\n\r\n»);
    «will not reach 17 or it is more.\r\n\r\n»);
    sbl. Append («10) Первая карта банкомёта может сдавать-
ся " +
    sbl. Append («10) The first card of a dealer can be given» +
    «лицевой стороной вниз и быть невидимой.\r\n\r\n»);
    «the face down and to be nevidimoy.\r\n\r\n»);
    sbl. Append («11) Вы должны или оставить ставку по умол-
чанию, " +
    sbl. Append («11) You should or to leave a bet by default,» +
    «или установить новую вашу ставку до раздачи карт " +
    «or to set your new bet before distribution of cards» +
    «(в последнем случае используйте команды " +
    «(in the latter case use the commands» +
    «<BetPlus -> и <BetMinus -> " +
    «в Меню для элемента управления mainMenu1).\r\n\r\n»);
    «in the Menu for the mainMenu1 control).\r\n\r\n»);
    sbl. Append («12) Ваше значение банка " +
    sbl. Append («12) Your value of bank» +
    «все время показывают на экране. " +
    «all the time show on the screen.» +
    «Если значение банка станет ниже вашей ставки, " +
```

```
    «If value of bank becomes below your bet,» +
    «вам предложат начать новую игру.\r\n\r\n»);
    «to you will suggest to begin the new game.\r\n\r\n»);
    sbl. Append («13) Когда вы набрали карты, вы можете " +
    sbl. Append («13) When you gathered cards, you can» +
    «приостановить игру, выбрав в Меню команду Останов. " +
    «suspend a game, having chosen in Menu the command Stop.» +
    «Банкомёт покажет свою карту.\r\n\r\n»);
    «Dealer will show the card.\r\n\r\n»);
    sbl. Append («14) Схема оплаты:\r\n»);
    sbl. Append («14) Scheme of payment:\r\n»);
    sbl. Append («проигравший платит победителю по договорённости, " +
    sbl. Append («the loser pays the winner by agreement,» +
    «например, 1:1;\r\n»);
    «for example, 1:1; \r\n»);
    sbl. Append («игра в очко желает вам всего наилучшего.\r \n»);
    sbl. Append («The game in a point wishes you all the best.\r \n»);
    textBox1.Text = sbl.ToString ();
```

Естественно, текст в этом листинге мы можем редактировать, как пожелаем.

В режиме выполнения, после выбора команды Помощь на форме Form1, поверх этой формы Form1 появляется справочная форма Form2 (рис. 1.33). Внутри элемента управления TextBox мигает курсор, который мы можем перемещать клавишами, одновременно перемещая текст, чтобы он стал видимым. На рис. 1.33 видны первые три правила игры, а следующие правила 4, 5 и 6 на форме уже были показаны ранее.

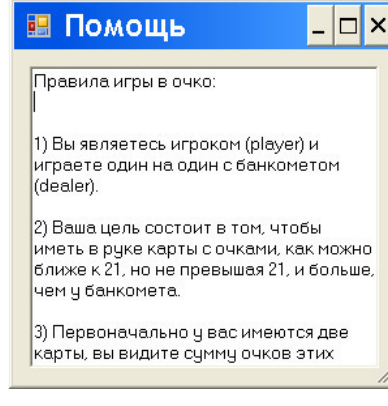


Рис. 1.33. Справочная форма Form2.

Напомним, что выше мы записали код для вывода справочной формы Form2 методом ShowDialog как модальной формы (рис. 1.34), а именно, мы не сможем продолжить игру на первой форме Form1 (и на любой другой форме), пока

не закроем форму Form2.

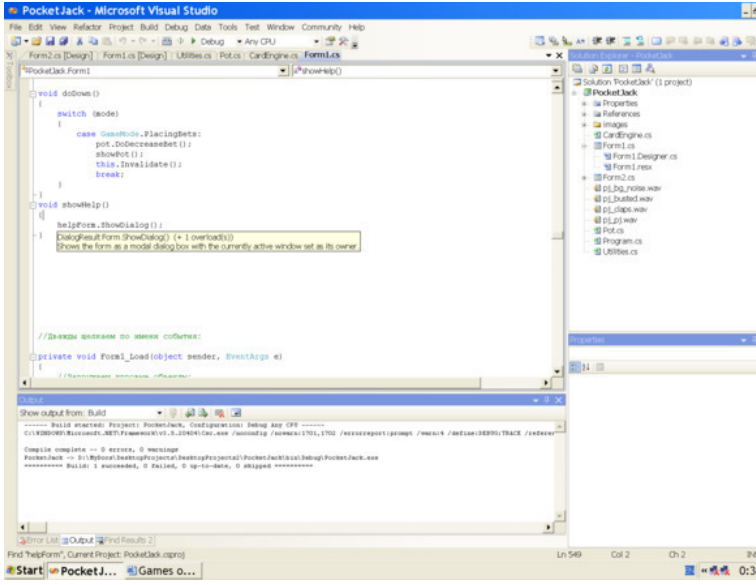


Рис. 1.34. Код для вывода Form2 методом ShowDialog как модальной формы.

Для вывода справочной формы Form2 методом Show как немодальной формы надо записать:

```
helpForm.Show ();
```

Теперь, активируя (щёлкая) форму Form1, мы сможем продолжить игру на первой форме Form1 (и на любой другой форме любого приложения), видя на экране справочную форму Form2 (которую можно передвинуть в любое удобное место экрана).

1.22. Запуск игры

Строим и запускаем программу на выполнение обычным образом:

Build, Build Selection; Debug, Start Without Debugging.

В ответ, Visual C# выводит форму Form1 с показанным выше фоном данной игры.

Отметим, что все графические файлы этой игры применимы как для смартфонов и планшетов (описанных в наших предыдущих книгах), так и для настольных компьютеров. Для настольного компьютера с большим (чем у смартфона и планшета) экраном размеры рисунков фона и карт можно увеличить (в случае необходимости) по описанной ранее схеме.

Для начала игры нажимаем клавишу Enter и играем согласно приведённым выше правилам.

Таким образом, эта глава описывает механизм раздачи карт (из колоды) случайным образом (на основе генератора случайных чисел класса Random), показ их на экране, показ их в «руках» для каждого игрока и управления карточной игрой.

Мы разработали методику программирования полностью функциональной игры, которая в США и других странах называется как «Black Jack», «21» или понтон (pontoon) при использовании 52 карт, а в России и других странах обычно

называется как «очко» или «21» при использовании 36 карт.

В разработанной в данной главе игре можно вместо 52 карт добавить в проект новые 36 карт (в случае необходимости) или из 52 карт удалить лишние карты, чтобы оставить только 36 карт.

По методике данной главы можно разрабатывать самые разнообразные карточные игры.

Часть II. Методология программирования искусственного интеллекта в спортивных играх с ракетками и мячами

Глава 2. Методика программирования искусственного интеллекта в игре в теннис для игрока с компьютером или двух игроков

2.1. Общие сведения

Разработаем методику проектирования и программирования такой типичной и широко распространённой игры, как игра в теннис с мячом и двумя ракетками.

Кратко, сюжет игры заключается в следующем. После старта игры появляется форма, справа на форме (точнее, в клиенткой области формы) находится ракетка Игрока 1, слева – ракетка Игрока 2 (по первому режиму игры) или Компьютера в роли Игрока 2 (по второму режиму игры). Обе ракетки могут перемещаться только вертикально. Мяч произвольным образом прыгает в пределах 2-х границ формы, отскакивая от верхней и нижней границ формы и двух ракеток. Правая граница формы – это ворота Игрока 1, а левая граница формы – это ворота Компьютера в роли Игрока 2 или самого Игрока 2. Если мяч коснётся левой или правой границ формы, считается, что один игрок забил мяч в ворота другого игрока, у пропустившего мяч игрока количество жизней (lives) с трёх уменьшается на единицу, а мяч снова появляется в произвольной точке формы и летит в произвольном направлении.

По первому режиму игры Singleplayer (Игрока 1 с компьютером), заданному по умолчанию, Игрок 1 при помощи клавиш со стрелками перемещает правую ракетку, старается отбить ею мяч и не дать мячу коснуться правой границы

формы, так как после каждого такого касания (пропущенного в свои ворота мяча) Игрок 1 теряет одну жизнь. Более того, Игрок 1 должен стараться (по возможности) ракеткой отбить мяч таким образом, чтобы забить его в противоположные ворота. Аналогично поступает Компьютер с левой ракеткой.

По второму режиму игры Multiplayer (Игрока 1 с Игроком 2), Игрок 1 действует так же, как по первому режиму, а вот Игрок 2 при помощи клавиш W и S перемещает левую ракетку, старается отбить ею мяч и не дать мячу коснуться левой границы формы, так как после каждого такого касания (пропущенного в свои ворота мяча) Игрок 2 теряет одну жизнь.

В программе в методе GetInputStates в строках:

```
if (gamePadUp ||
keyboard.IsKeyDown (Keys. Up))
rightPaddlePosition -= moveFactorPerSecond;
if (gamePadDown ||

keyboard.IsKeyDown(Keys.Down))
```

мы видим, что Игрок 1 управляет правой ракеткой при помощи клавиш со стрелками Up и Down.

В методе GetInputStates в строках:

```
if (gamePad2Up ||
keyboard.IsKeyDown (Keys. W))
leftPaddlePosition -= moveFactorPerSecond;
```

```
if (gamePad2Down ||
keyboard.IsKeyDown (Keys. S) ||
keyboard.IsKeyDown (Keys. O))

    leftPaddlePosition += moveFactorPerSecond;
```

мы видим, что Игрок 2 управляет левой ракеткой при помощи клавиш W, S и O.

По мере игры скорость перемещения мяча увеличивается, а угол отскока мяча от границ формы и ракеток изменяется, что приводит к достаточно быстрому окончанию игры.

Игра прекращается, когда один из игроков потеряет все жизни. Победителем считается игрок, у которого сохранилась хотя бы одна жизнь.

2.2. Правила игры

1. После запуска игры, на экране появляется форма с меню Singleplayer, Multiplayer и Exit (рис. 2.1). Клавишами со стрелками выбираем нужную команду и нажимаем клавишу Enter.

2. Появляется поле игры, на котором справа находится синяя (Blue) ракетка Игрока 1, а слева – красная (Red) ракетка Компьютера в роли Игрока 2 (по первому режиму игры Singleplayer) или Игрока 2 (по второму режиму игры Multiplayer).

Вверху слева и справа находятся два табло с тремя жизнями Lives в виде начальных трёх шаров для двух игроков (рис. 2.2).

Мяч произвольным образом прыгает в пределах 4-х границ формы, отскакивая от границ формы и двух ракеток. После каждого такого отскока мы слышим звук добавленного в проект звукового файла. Правая граница формы – это ворота Игрока 1, а левая граница формы – это ворота Игрока 2 (или Компьютера в роли Игрока 2).

Рис. 2.1. Выбираем режим игры Singleplayer или Multiplayer.

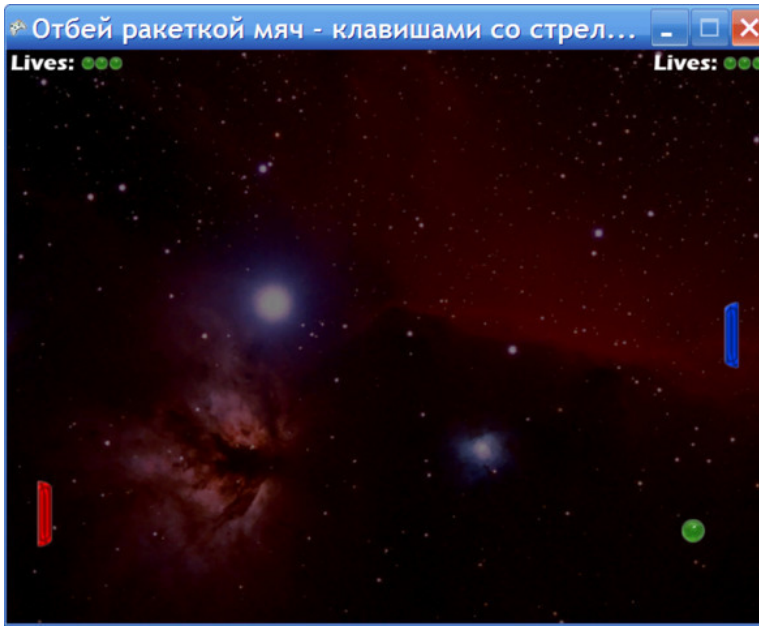


Рис. 2.2. Справа – ракетка Игрока 1, слева – ракетка Игрока 2, мяч – в нижнем правом углу.

3. Предположим, что мы выбрали заданный по умолчанию первый режим игры Singleplayer (Игрока 1 с компьютером) и нажали клавишу Enter. Игрок 1 при помощи клавиш со стрелками Up и Down перемещает правую ракетку, старается отбить ею мяч и не дать мячу коснуться правой границы формы, так как после каждого такого касания (пропущенно-

го в свои ворота мяча) Игрок 1 теряет одну жизнь. Аналогично поступает Компьютер с левой ракеткой.

4. Предположим, что клавишами со стрелками мы выбрали второй режим игры Multiplayer (Игрока 1 с Игроком 2) и нажили клавишу Enter. Игрок 1 действует так же, как по первому режиму, а вот Игрок 2 при помощи клавиш W и S перемещает левую ракетку, старается отбить ею мяч и не дать мячу коснуться левой границы формы, так как после каждого такого касания (пропущенного в свои ворота мяча) Игрок 2 теряет одну жизнь.

5. Игра прекращается, когда один из игроков потеряет все жизни.

6. Победителем считается игрок, у которого сохранилась хотя бы одна жизнь (на рис. 2.3 – это Игрок 2 с красной ракеткой Red). Мы видим сообщение Red Won. Чаще всего, Компьютер (в роли Игрока 2) побеждает.

4. Если в игре участвуют несколько человек, то победителем считается тот, у кого после окончания игры сохранилось больше жизней.

5. Форму с игрой закрываем стандартно, щёлкая на ней значок Close.

На основании этих правил можно сформулировать другие правила.

В игре можно использовать различные графические изображения и звуковые файлы различных форматов (с внесением соответствующих изменений в приведённую далее про-

грамму).

Рис. 2.3. Сообщение о победителе.

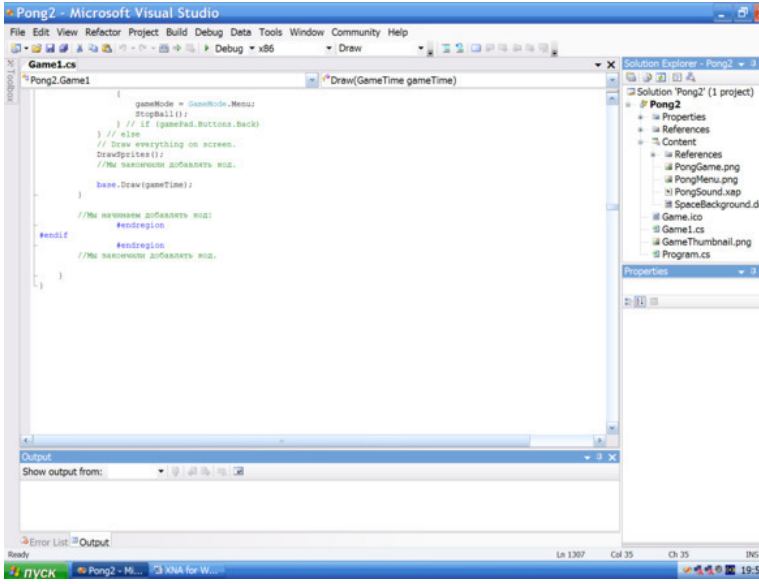
2.3. Создание проекта Visual Studio

В VS щёлкаем значок New Project (или File, New, Project), в панели New Project в окне Project types выбираем тип проекта Visual C#, XNA Game Studio в окне Templates выделяем шаблон Windows Game, в окне Name записываем имя Pong2 и щёлкаем OK. VS создаёт проект и выводит файл Game1.cs с показанным выше шаблоном кода.

В панели Solution Explorer выполняем правый щелчок по имени проекта Content, в контекстном меню выбираем Add, Existing Item, в панели Add Existing Item в окне «Files of type» выбираем «All Files», в центральном окне находим (например, в Интернете) и с нажатой клавишей Shift или Ctrl выделяем три не звуковых файла PongGame.png, PongMenu.png и SpaceBackground. dds, и щёлкаем кнопку Add. Эти файлы добавляются в панель Solution Explorer, как показано на рис. 2.4 (файла PongSound. xap здесь пока не должно быть, он появится после создания проекта XACT). Если в панели Solution Explorer выделить файл SpaceBackground. dds, то ниже в панели Properties мы увидим тип этого файла (рис. 2.5).

Чтобы добавить в проект звуковые файлы, щёлкаем значок Open File (или в меню выбираем File, Open, File), в появившейся панели Open File в окне «Look in:" находим (например, в Интернете) папку со звуковыми файлами (в дан-

ном примере, PongBallHit. wav и PongBallLost. wav), с нажатой клавишей Shift или Ctrl выделяем все эти звуковые файлы, выполняем по ним правый щелчок и в контекстном меню выбираем команду Копировать (Copy) для копирования этих файлов в буфер обмена.



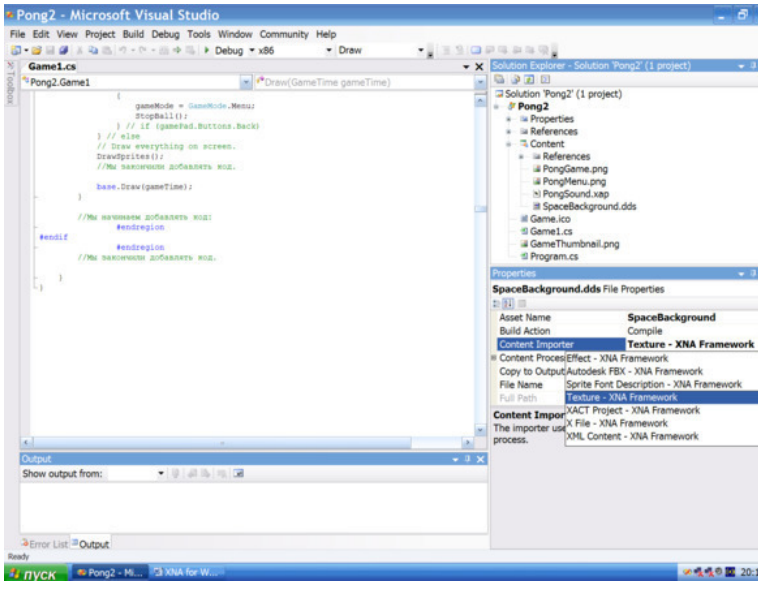


Рис. 2.4. Панель Solution Explorer. **Рис. 2.5.** Панель Properties.

Теперь в этой же панели Open File в окне «Look in:" находим наш проект Visual Studio и в нём папку Content, на белом поле центрального окна выполняем правый щелчок и в контекстном меню выбираем команду Вставить (Paste) для вставки звуковых файлов из буфера обмена. Эти звуковые файлы вставятся в папку Content только в панели Open File, а в панели проводника решения Solution Explorer

в этой же папке Content этих звуковых файлов не будет видно.

Кратко поясним, что в данной игре файл PongBallHit. wav будет воспроизводится, когда будет происходить столкновение мяча с ракеткой. А файл PongBallLost. wav будет воспроизводится, когда игрок пропускает мяч, мяч пролетает мимо ракетки и исчезает с формы, а игрок теряет одну жизнь.

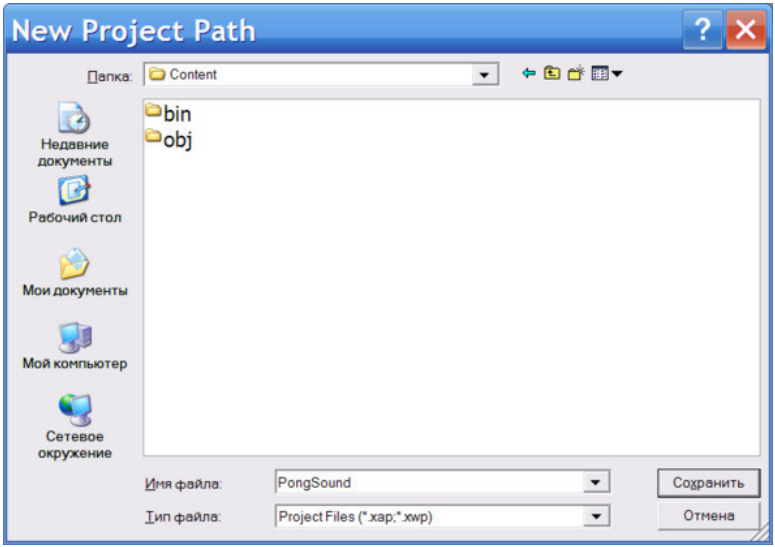
На данном этапе программирования приложения построение (Build, Build Solution) и выполнение (Debug, Start Without Debugging) программы должно происходить без ошибок, но и без звука. А чтобы получить звуковое сопровождение приложения, нам нужно создать звуковой проект типа XACT в виде файла формата (.xap), как описано в следующем параграфе. Сворачиваем или закрываем Visual Studio.

2.4. Создание звукового проекта XACT

Приведём пошаговую инструкцию создания звукового проекта типа XACT в виде файла формата Audio Project или (.xap).

1. Чтобы запустить XACT, выбираем команды Start (Пуск), All Programs (Все программы), | Microsoft XNA Game Studio Tools, Microsoft Cross-Platform Audio Creation Tool (XACT). Появляется показанная ранее панель Microsoft Cross-Platform Audio Creation Tool (XACT) v.2.0 (Windows).

2. В меню File выбираем New Project. Появляется панель New Project Path. В этой панели в окне «Папка:" находим в проекте Visual Studio папку Content, в окне «Имя файла:" записываем любое имя, например, PongSound и щёлкаем кнопку Сохранить (Save), рис. 2.6. Отметим, что если в этой панели в окне «Тип файла:" мы выберём строку All Files, то увидим также и добавленные нами в эту папку звуковые файлы.


New Project Path
Папка:
Content
bin
obj
Недавние документы
Рабочий стол
Мои документы
Мой компьютер
Сетевое окружение
Имя файла:
PongSound
Тип файла:
Project Files (*.xap;*.xwp)
Сохранить
Отмена

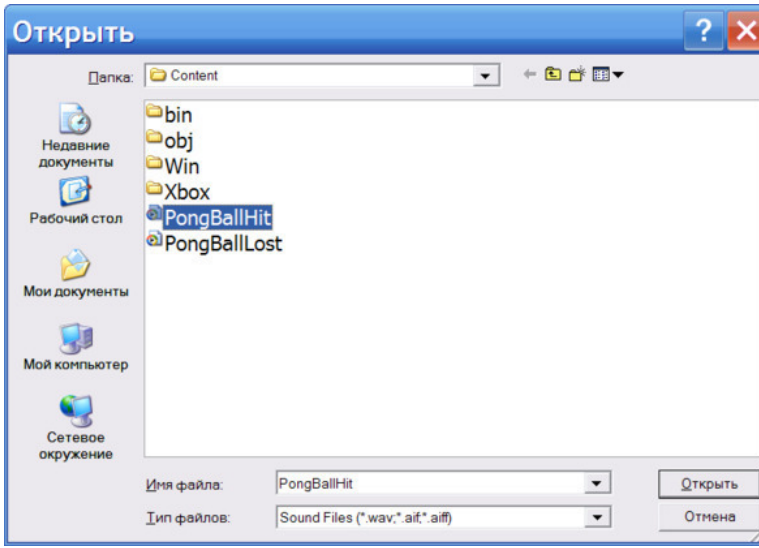


Рис. 2.6. В проекте VS в папке Content записываем имя. **Рис. 2.7.** Выделяем файл.

В панели Microsoft Cross-Platform Audio Creation Tool (XACT) v.2.0 (Windows) вместо слов New Project появляется записанное нами имя файла.

3. В меню Wave Banks выбираем команду New Wave Bank. В панели XACT в центральном окне появляется панель Wave Bank.

В меню Sound Banks выбираем команду New Sound Bank. В панели XACT в центральном окне появляется панель

Sound Bank.

Размещаем эти две панели Wave Bank и Sound Bank в центральном панели XACT так, как нам удобно. При щелчке по любой из этих панелей, в нижнем левом углу панели XACT появляются параметры соответствующей панели Wave Bank или Sound Bank.

4. Выполняем правый щелчок на белом поле панели Wave Bank и в контекстном меню выбираем команду Insert Wave File (s).

В появившейся панели Открыть (Open) проверяем, чтобы в окне «Папка:" была папка Content (если этой папки в окне нет, то находим ее), выделяем первый добавленный нами звуковой файл (рис. 2.7) и щёлкаем кнопку Открыть. Панель Открыть закрывается, а этот файл появляется в панели Wave Bank (со всеми заполненными свойствами).

Аналогично выполняем правый щелчок на белом поле панели Wave Bank, в контекстном меню выбираем команду Insert Wave File (s), в панели Открыть выделяем второй файл Lost. wav и щёлкаем кнопку Открыть. Этот файл появляется в панели Wave Bank.

На рисунке в окне «Тип файлов:" видно, какого формата звуковые файлы поддерживает XNA, а именно: Sound Files (*.wav, *.aif, *.aiff).

5. В панели Wave Bank сначала выделяем указателем мыши все добавленные нами звуковые файлы, затем захватываем их указателем мыши и переносим в панель Sound Bank,

точнее, в нижнюю левую часть со свойством Cue Name. Эти файлы появляются в двух левых окнах панели Sound Bank. Щёлкаем по файлам, в двух правых окнах появляется информация о файлах.

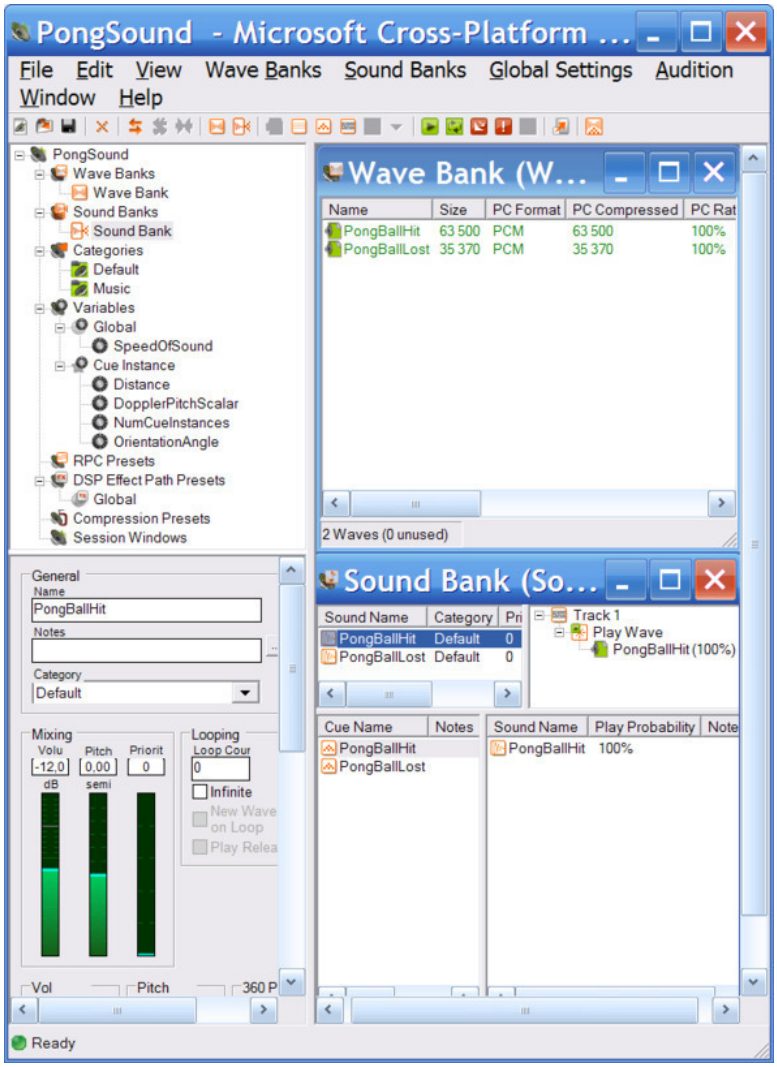

PongSound - Microsoft Cross-Platform ...
File Edit View Wave Banks Sound Banks Global Settings Audition
Window Help
PongSound
Wave Banks
Wave Bank
Sound Banks
Sound Bank
Categories
Default
Music
Variables
Global
SpeedOfSound
Cue Instance
Distance
DopplerPitchScalar
NumCueInstances
OrientationAngle
RPC Presets
DSP Effect Path Presets
Global
Compression Presets
Session Windows
Wave Bank (W...
Name Size PC Format PC Compressed PC Rat
PongBallHit 63 500 PCM 63 500 100%
PongBallLost 35 370 PCM 35 370 100%
2 Waves (0 unused)
General
Name
PongBallHit
Notes
Category
Default
Mixing
Volu Pitch Priorit
-12,0 0,00 0
dB semi
Looping
Loop Cour
0
Infinite
New Wave on Loop
Play Relea
Vol Pitch 360 P
Sound Bank (So...
Sound Name Category Pri
PongBallHit Default 0
PongBallLost Default 0
Track 1
Play Wave
PongBallHit (100%)
Cue Name Notes
PongBallHit
PongBallLost
Sound Name Play Probability Note
PongBallHit 100%
Ready

Рис. 2.8. Панель Microsoft Cross-Platform Audio Creation Tool (XACT) v.2.0 (Windows).

6. Выбираем меню View и убеждаемся в том, что по умолчанию флажок установлен в команде View Windows Property (если не так, то делаем так).

7. Стандартно сохраняем законченный проект XACT (File, Save Project), рис. 2.8.

8. Закрываем этот проект XACT, щёлкая значок Close (или File, Exit).

9. Открываем (если мы его закрыли) наш проект Visual Studio и папку Content (по схеме File, Open, File). Мы увидим, что в этой папке кроме добавленных нами звуковых файлов находятся автоматически добавленные системой две пустые папки Win и Xbox и, главное, созданный нами звуковой проект типа XACT с записанным нами именем и с автоматически добавленным расширением (.xap). В расширении (.xap) две последние буквы являются первыми буквам слов Audio Project (звуковой проект).

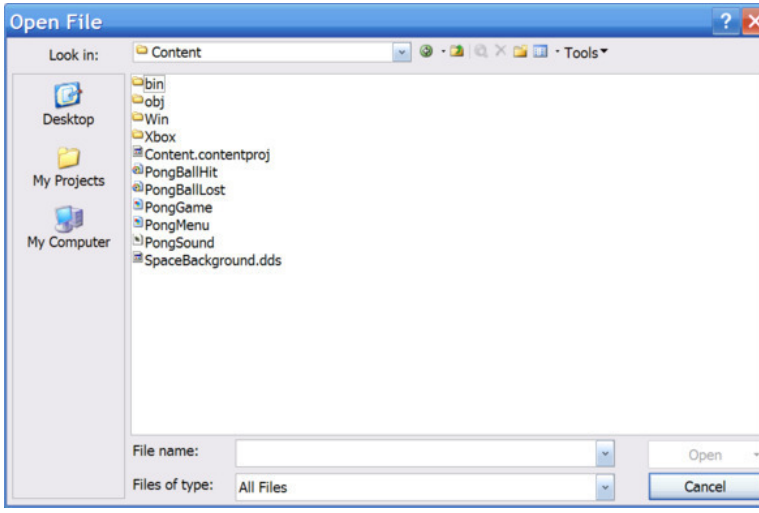
Проверяем, чтобы созданный нами файл (.xap) был виден и в папке Content панели Open File, и обязательно в этой же папке Content в панели проводника решения Solution Explorer.

10. Если этого нет, то поступаем следующим образом. В панели Solution Explorer выполняем правый щелчок по имени проекта Content, в контекстном меню выбираем

Add, Existing Item, в панели Add Existing Item в окне «Files of type» выбираем «All Files», в окне «Look in:" находим папку Content данного проекта Visual Studio, в центральном окне выделяем созданный нами файл (.xap), и щёлкаем кнопку Add (или дважды щёлкаем по имени файла). В панели Solution Explorer в папке Audio должен появиться созданный нами файл (.xap).

Окончательно, в нашем проекте Visual Studio в папке Content панели Open File (которая появляется после File, Open, File) должны быть папки и файлы, показанные на рис. 2.9. А в папке Content проводника решения Solution Explorer должны быть папки и файлы, показанные на приведённом выше рисунке.

Рис. 2.9. Панель Open File.



На любом этапе разработки данного проекта XACT мы можем либо сохранить (File, Save Project), либо не сохранять (если допустили ошибки) проект и закрыть XACT. А затем снова запустить XACT и открыть наш проект по стандартной схеме: File, Open Project, в панели Открыть выделяем созданный нами файл проекта формата (.xap) и щёлкаем кнопку Открыть (либо дважды щёлкаем по имени этого файла). Затем дважды щёлкаем в директории проекта XACT по второму имени Wave Bank и тем самым открываем волновой банк, дважды щёлкаем в директории по второму имени Sound Bank и тем самым открываем звуковой банк. В обоих этих банках мы увидим добавленные нами звуковые файлы.

Далее мы можем редактировать этот проект, сохранять, и все наши изменения будут учтены при выполнении приложения.

Конец ознакомительного фрагмента.