Лекция № 5

СОЗДАНИЕ ПРОСТЕЙШЕЙ ИГРЫ в PHONE 7

Создание игры для телефона

Итак, проиллюстрируем процесс создания игры на простом примере — морского боя. Сразу скажу, что морской бой — это не та игра, в которую многие играли в школе на бумаге в клеточку. Мы будем реализовывать игру, которая была доступна лет 30 назад в игровых автоматах. В ней надо было попасть торпедой в плывущий вражеский корабль. В нашей игре также будет корабль, плавающий в верхней части экрана, и снаряды, которые можно будет выпускать из нижней части экрана путем прикосновения к нему. Основной экран игры изображен на рис. 1.

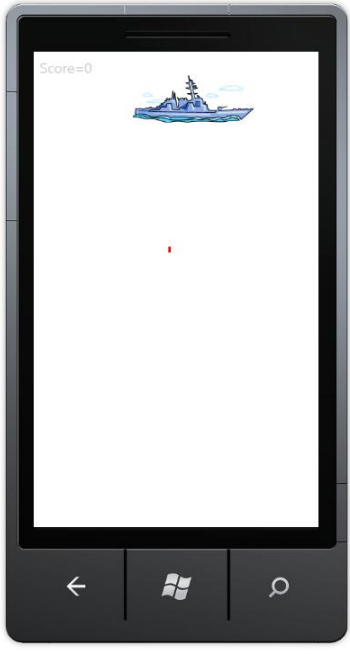

Рис. 1. Основной экран игры

Создаем графические артефакты

Для начала определимся с графическими артефактами. Для игры нам потребуются три основных рисунка: корабль, ракета и изображение взрыва (его не видно на рис. 2 — взрыв появляется только в случае поражения корабля ракетой).

Вопрос рисования игровых графических артефактов выходит за рамки данной книги — хотя отметим, что для данного примера все рисунки были получены из стандартного набора Clipart от Microsoft Office и сохранены в формате PNG в виде графических файлов (табл. 1).

Таблица 1. *Используемые графические файлы*

Файл	Изображение
Explode.png	
Ship.png	
Rocket.png	

Создаем игровой проект

Теперь приступим к созданию самой игры. Для начала запустим Visual Studio и выберем новый проект типа **XNA Game Studio | Windows Phone Game (4.0)**, как показано на рис. 3.

Для примера, назовем проект `WindowsPhoneShipBattle`. После этого выберем версию Windows Phone 7.1 — и Visual Studio создаст для нас каркас игры, который будет состоять из двух проектов (рис. 4).

- WindowsPhoneShipBattle — это основной проект игры, в котором располагается вся логика, а также ресурсы, связанные с приложением Windows Phone (значки приложения, метаданные и т. д.). Обратите внимание на следующие основные файлы:
  - Game1.cs — это главный игровой файл, описывающий класс игры, в котором программируются упомянутые ранее функции игрового цикла `LoadContent`, `Update` и `Draw`;

- Program.cs — файл, запускающий игру на выполнение. Он не представляет собой особого интереса и почти никогда не требует модификации;
- Background.png и PhoneGameThumb.png — сгенерированные по умолчанию значки приложения.

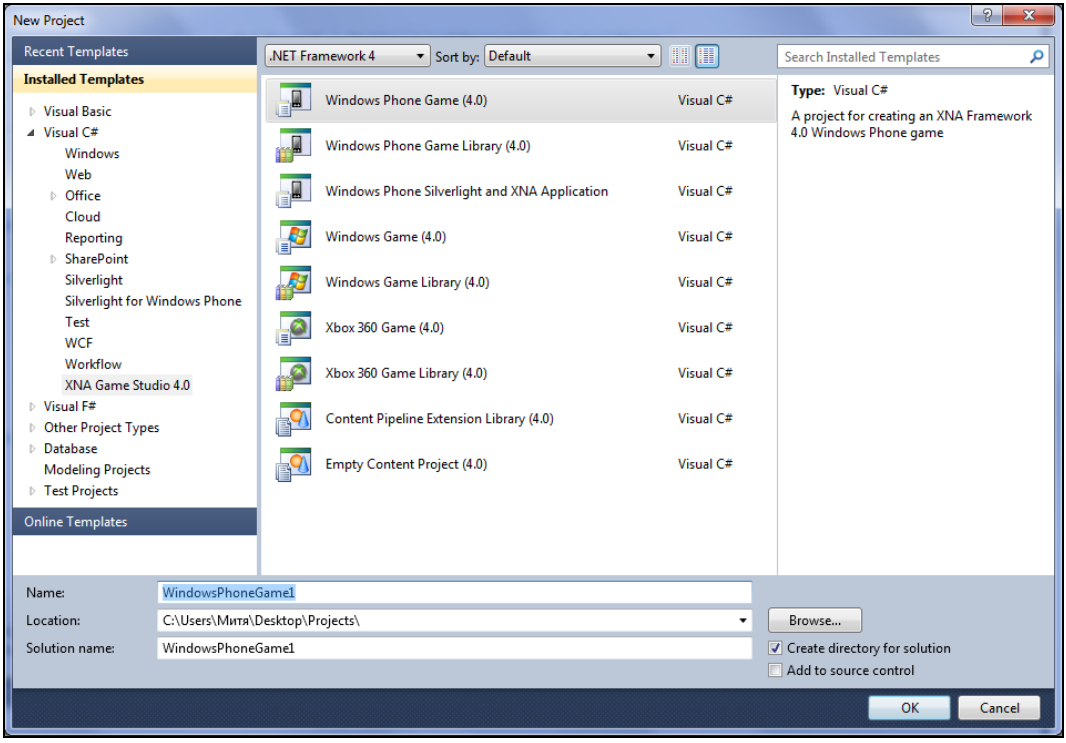

Рис. 3. Создание нового проекта игры

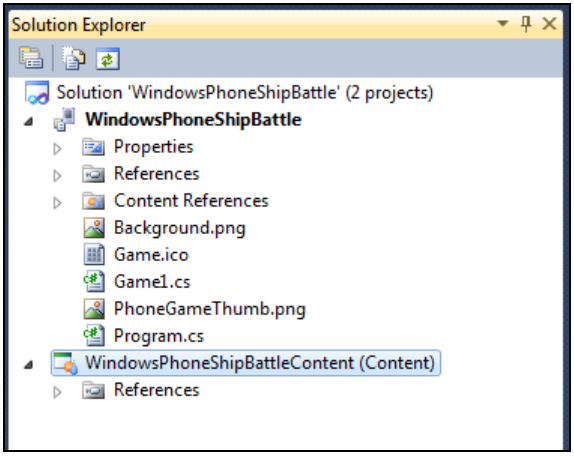

Рис. 4. Проекты, созданные Visual Studio

- WindowsPhoneShipBattleContent — это отдельный проект (так называемый Content Pipeline), в котором должны располагаться игровые артефакты: изображения, звуковые и видеофайлы и т. д. В процессе компиляции игры все артефакты в Content Pipeline преобразуются из исходных форматов во внутренний формат, используемый XNA, с помощью специальных процессоров. Для стандартных графических, музыкальных и 3D-форматов могут использоваться встроенные стандартные процессоры, однако возможно и создание своих процессоров для нестандартных ситуаций: например, преобразование формата хранения уровней и др.

Для того чтобы поместить в проект созданные нами ранее графические артефакты, просто перетащим их из Проводника Windows в проект WindowsPhoneShipBattle (рис. 5).

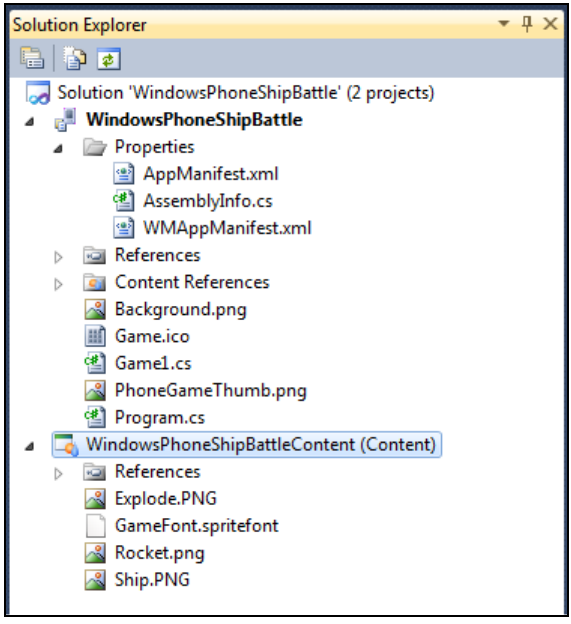

Рис. 5. Добавление графических файлов в проект

Рис. 6. Каркас игры в эмуляторе

Созданный каркас игры представляет собой работающую игру — мы можем откомпилировать и запустить ее на эмуляторе телефона. Как выглядит такая игра, показано на рис. 6.

Первые шаги

Посмотрим, почему игра выглядит следующим образом. Если мы откроем метод `Draw` в Game1.cs, то увидим, что он выглядит так, как представлено в листинге 1 (для краткости комментарии были удалены).

Листинг 1. Метод `Draw`

```
protected override void Draw(GameTime gameTime)
{
    GraphicsDevice.Clear(Color.CornflowerBlue);
    base.Draw(gameTime);
}
```

Мы видим, что в методе явно указано закрашивание экрана васильковым цветом. Чтобы нарисовать на экране что-то другое, нужно явным образом изменить метод `Draw` (попробуйте, например, заменить цвет закраски на белый).

Для начала нарисуем корабль на экране. Для этого в классе игры создадим переменные, которые будут хранить рисунок корабля (типа `Texture2D`), а также его положение и скорость (типа `Vector2`) (листинг 2).

Листинг 2. Переменные для хранения рисунка корабля, его положения и скорости

```
Texture2D ship;
Vector2 ship_pos = new Vector2(0, 30);
Vector2 ship_dir = new Vector2(3, 0);
```

Метод `LoadContent` запрограммируем на загрузку изображения корабля (помимо этого, в нем останется еще одна строчка для создания объекта `SpriteBatch`, необходимого для рисования) (листинг 3).

Листинг 3. Загрузка изображения корабля

```
protected override void LoadContent()
{
    spriteBatch = new SpriteBatch(GraphicsDevice);
    ship = Content.Load<Texture2D>("Ship");
}
```

Наконец, метод `Draw` будет заниматься рисованием корабля (листинг 4).

Листинг 4. Рисование корабля

```
protected override void Draw(GameTime gameTime)
{
    GraphicsDevice.Clear(Color.White);
    spriteBatch.Begin();
    spriteBatch.Draw(ship, ship_pos, Color.White);
    spriteBatch.End();
    base.Draw(gameTime);
}
```

Здесь следует отметить одну тонкость — при рисовании спрайтов на экране мы рисуем картинки "порциями", называемыми SpriteBatch. Соответственно, мы открываем такую "рисовательную транзакцию" вызовом `spriteBatch.Begin()` и заканчиваем вызовом `spriteBatch.End()`, между которыми расположены вызовы `spriteBatch.Draw()` или `DrawString()`. В XNA принят подход к рисованию, называемый Immediate Mode Rendering, т. е. выполнение команд без промежуточной буферизации. При этом явно указываемая программистом транзакционность рисования позволяет избежать мерцания путем оптимизации процесса вывода на экран для последовательных кадров.

Запустив игру на этом этапе, мы увидим изображение корабля на экране (рис. 7).

Мы видим, что XNA по умолчанию использует ландшафтную (горизонтальную) ориентацию для игры. Для изменения ориентации экрана на портретную добавим три строчки кода в конструктор игрового класса (листинг 5, рис. 8).

Листинг 5. Изменение ориентации экрана

```
graphics.SupportedOrientations = DisplayOrientation.Portrait;
graphics.PreferredBackBufferHeight = 800;
graphics.PreferredBackBufferWidth = 480;
```

Чтобы заставить корабль двигаться (т. е. действительно сделать те самые первые шаги, о которых говорилось в названии раздела), необходимо в методе `Update`

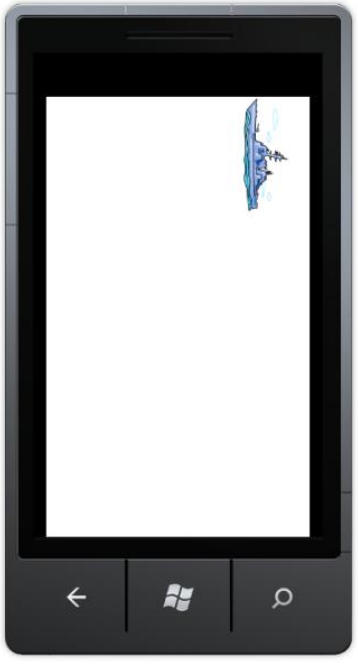

Рис. 7. Изображение корабля на экране

Рис. 8. Изображение корабля в портретной ориентации

изменять его координаты, которые в данном случае представляют собой состояние игры (листинг 6).

Листинг 6. Изменение состояния игры

```
protected override void Update(GameTime gameTime)
{
    width = GraphicsDevice.Viewport.Width;
    height = GraphicsDevice.Viewport.Height;

    ship_pos += ship_dir;
    if (ship_pos.X <= 0 || ship_pos.X > width — ship.Width)
    {
        ship_dir = -ship_dir;
    }
    base.Update(gameTime);
}
```

Здесь мы по ходу дела определяем ширину и высоту экрана (их можно было бы задать в виде констант, но это затруднит в дальнейшем поддержку различных ориентаций телефона) в переменных `height` и `width` (соответствующие переменные необходимо также описать в игровом классе). Далее корректируем позицию корабля (это делает "самая главная" строчка, выделенная жирным шрифтом), а затем обрабатываем "отскакивание" корабля от края экрана.

Обратите внимание, что XNA поддерживает типы данных из линейной алгебры (в данном случае двумерные векторы, но также и трехмерные векторы, матрицы и т. д.) и операции с ними, что сильно упрощает многие операции, поскольку они выполняются естественным образом. Например, в данном случае мы корректируем положение корабля простым добавлением к вектору координат вектора скорости (неявно умноженного на время, предположительно равное 1).

В заключение, для более красивого рисования корабля, слегка модифицируем ме-тод рисования. Теперь, если корабль плывет вправо, его картинка отражается вер-тикально вокруг свой оси, так что корабль все время плывет носом вперед, как это обычно происходит в реальной жизни (листинг 7).

Листинг 7. Отражение корабля при движении вправо

```
spriteBatch.Draw(ship, ship_pos, null, Color.White,0f,Vector2.Zero,1.0f,
ship_dir.X>0 ? SpriteEffects.FlipHorizontally : SpriteEffects.None,0f);
```

Наполняем игру содержанием: обработка действий пользователя

Теперь, когда мы научили корабль двигаться, добавим ракету и, собственно, возможность игры, т. е. взаимодействия программы с пользователем.

Для начала в игровом классе нам потребуется определить переменные для графического изображения ракеты и взрыва. Дополнительно для отрисовки взрыва понадобится переменная `explode` — она будет вести обратный отсчет числа кадров, во время которых вместо корабля отображается взрыв (листинг 8).

Листинг 8. Переменные для поддержки ракеты и взрыва

```
Texture2D rocket, explosion; // текстуры
Vector2 rocket_pos = new Vector2(0, 0); // положение ракеты
int explode = 0; // признак того, что недавно был взрыв
```

Положение ракеты, равное нулевому вектору, соответствует ситуации, когда ракета не была выпущена.

Отрисовка ракеты в методе `Draw` не представляет сложностей; кроме того, если недавно был взрыв и переменная `explode` не равна нулю, то вместо корабля нам надо отрисовывать соответствующий спрайт со взрывом. Метод `Draw` теперь содержит код, представленный в листинге 9.

Листинг 9. Метод `Draw`

```
protected override void Draw(GameTime gameTime)
{
    GraphicsDevice.Clear(Color.White);
    spriteBatch.Begin();
    if (explode == 0)
    {
        spriteBatch.Draw(ship, ship_pos, null,
          Color.White,0f,Vector2.Zero,1.0f,
          ship_dir.X>0 ? SpriteEffects.FlipHorizontally :
            SpriteEffects.None, 0f);
    }
    else spriteBatch.Draw(explosion, ship_pos, Color.White);
    if (rocket_pos != Vector2.Zero)
    {
        spriteBatch.Draw(rocket, rocket_pos, Color.Red);
    }
    spriteBatch.End();
    base.Draw(gameTime);
}
```

Теперь перейдем к обсуждению метода `Update`. Его будем рассматривать по частям. Первая часть отвечает за отрисовку взрыва: когда переменная-флаг `explode` ненулевая, единственная задача нашей игры — отрисовать взрыв. Поэтому мы просто уменьшаем счетчик кадров, в течение которых демонстрируется взрыв, а когда он (счетчик) достигает нулевой отметки — возвращаем корабль в исходное положение, чтобы игра началась снова (листинг 10).

Листинг 10. Метод `Update`

```
protected override void Update(GameTime gameTime)
{
    if (explode > 0)
    {
        explode--;
        if (explode == 0)
        {
            ship_pos.X = 0;
            ship_dir.X = 3;
        }
        base.Update(gameTime);
        return;
    }
    ....
    base.Update(gameTime);
}
```

Обратите внимание, что в конце метода `Update` вызывается метод `Update` базового класса.

Следующий фрагмент кода, рассмотренный нами ранее, отвечает за движение корабля влево-вправо (листинг 11).

Листинг 11. Движение корабля влево-вправо

```
ship_pos += ship_dir;
if (ship_pos.X + ship.Width >= width || ship_pos.X <= 0)
{
    ship_dir = -ship_dir;
}
```

Теперь перейдем к обработке действий пользователя. Следует решить, как будет происходить управление игрой — касанием экрана, жестами, поворотом или встряхиванием телефона (в этом случае надо задействовать акселерометр), нажатием на кнопки или как-то еще. В любом случае обработка действий пользователя занимает значимое место в функции `Update`.

В отличие от элементов управления Silverlight, управляемых событиями, в XNA обработка действий пользователя ведется по принципу опроса состояния в цикле игры. Например, нажатие на каком-то элементе управления в Silverlight-приложении ведет к возникновению однократного события нажатия, в то время как XNA-приложение может опрашивать состояние сенсорной панели телефона и определять, касается ли его палец в данный момент или нет.

В нашем случае мы будем управлять игрой именно с помощью прикосновений к экрану. Для обработки прикосновений существует специальный объект `TouchPanel`,

для которого метод `TouchPanel.GetState()` возвращает текущее состояние панели телефона. Состояние в свою очередь может содержать информацию о нескольких одновременных касаниях (поддержка multi-touch). Свойство `Count` в этом случае содержит количество одновременных касаний или 0, если пользователь вообще не касается панели.

Мы будем обрабатывать лишь одно (первое) касание, и в случае, если такое касание есть, будем запускать ракету из точки, *X*-координата которой совпадает с касанием, а координата по вертикали — фиксирована где-то внизу экрана (лис-тинг 12).

Листинг 12. Обработка касания

```
var tc = TouchPanel.GetState();
if (tc.Count>0)
{
    rock_pos.X = tc[0].Position.X;
    rock_pos.Y = 750;
}
```

Далее следует код, отвечающий за движение ракеты и отработку столкновения ракеты с кораблем (листинг 13).

Листинг 13. Обработка движения ракеты и столкновения с кораблем

```
if (rock_pos != Vector2.Zero)
{
    rock_pos += new Vector2(0, -7);
    if (rock_pos.Y >= 0 && rock_pos.Y <= ship_pos.Y + ship.Height &&
       rock_pos.X >= ship_pos.X && rock_pos.X <= ship_pos.X + ship.Width)
    {
        explode = 20;
        ship_pos.X = rock_pos.X – explosion.Width / 2;
        rock_pos = Vector2.Zero;
    }
    if (rock_pos.Y == 0) rock_pos = Vector2.Zero;
}
```

Для движения ракеты мы просто прибавляем на каждом цикле игры значение скорости в виде двумерного вектора. Столкновение определяется "вручную" по координатам (для более сложных фигур имеет смысл использовать другие функции распознавания пересечений из XNA) — в случае поражения устанавливается переменная `explode`, означающая, что следующие несколько кадров вместо корабля будут отображать взрыв. Если же ракета достигает верхней границы экрана — ее движение прекращается (устанавливаются нулевые координаты).

Улучшение игры: шрифты

Чтобы игра выглядела профессионально, недостает немногого — подсчета числа пораженных кораблей. На следующем этапе игры можно, конечно, добавлять диалоговые экраны, лучшие результаты и т. д., но наша цель в этой главе — создать минималистичную игру, которую, тем не менее, не стыдно будет выложить на Marketplace.

Добавляем счет игрока

Для подсчета успехов игрока введем переменную `score`, которую будем увеличивать на 1 при каждом поражении корабля. Посмотрим, как можно вывести значение счета на экран.

Как вы уже знаете, за все, что рисуется на экране, в XNA отвечает программист. Соответственно, весь текст выводится на экран в растровой форме, и для его отображения необходимо использовать специальный ресурс — шрифт.

Первое, что надо сделать, — поместить в проект какой-нибудь шрифт. Для описания шрифта используется специальный файл с расширением spritefont, который задает, помимо собственно названия и размера используемого шрифта, еще и другие параметры отображения, вроде кернинга и дополнительного межсимвольного расстояния. Это XML-файл, поэтому вы можете легко править параметры прямо в редакторе Visual Studio. Соответственно, можно либо использовать и включить в проект готовый spritefont-файл, либо добавить новый файл шрифта и затем поменять параметры по необходимости.

Заметим, что по умолчанию в XNA есть проблема с отображением символов кириллицы. Чтобы ее преодолеть, следует в spritefont-файле прописать явным образом, что мы намерены использовать интервалы кириллических символов. Для этого необходимо найти секцию `<CharacterRegions>` и добавить туда кириллический диа-пазон — в результате вся секция будет выглядеть так, как представлено в листин-ге 14.

Листинг 14. Добавление диапазона кириллических символов

```
<CharacterRegions>
    <CharacterRegion>
        <Start> </Start>
        <End>~</End>
    </CharacterRegion>
    <CharacterRegion>
        <Start>А</Start>
        <End>я</End>
    </CharacterRegion>
</CharacterRegions>
```

После этого нам необходимо описать переменную `font` типа `SpriteFont` и загрузить шрифт в методе `LoadContent` (предполагаем, что файл шрифта в контентном проекте назывался GameFont.spritefont) (листинг 15).

Листинг 15. Загрузка шрифта

```
font = Content.Load<SpriteFont>("GameFont");
```

Для вывода на экран строки текста используем специальную функцию `DrawString` в методе `Draw` (листинг 18.16).

Листинг 16. Вывод строки на экран

```
spriteBatch.DrawString(font, String.Format("Score={0}", score),
new Vector2(10, 10), Color.LightGray);
```

Помимо собственно строки и названия шрифтового ресурса, задается цвет и координаты надписи на экране.