

O'REILLY®

Head First

# Изучаем C#

Эндрю Стиллмен  
Дженнифер Грин

Четвертое  
издание



ПОДАРОК ДЛЯ МОЗГА



## Содержание (сводка)



Сделаем игру чуть более азартной! В нижней части окна выводится время, прошедшее с момента запуска игры. Показания таймера постоянно увеличиваются, а останавливается таймер только после нахождения последней пары.

	Введение	29
1	Начало работы с C#: <i>Быстро сделать что-то классное!</i>	41
2	Погружение в C#: <i>Команды, классы и код</i>	89
	<i>Лабораторный курс Unity № 1: Исследование C# с Unity</i>	127
3	Ориентируемся на объекты: <i>Написание осмысленного кода</i>	143
4	Типы и ссылки: <i>Данные и ссылки</i>	195
	<i>Лабораторный курс Unity № 2: Написание кода C# для Unity</i>	000
5	Инкапсуляция: <i>Умейте хранить секреты</i>	267
6	Наследование: <i>Генеалогическое древо объектов</i>	313
	<i>Лабораторный курс Unity № 3: Экземпляры GameObject</i>	383
7	Интерфейсы, приведение типов и is: <i>Классы должны держать обещания</i>	395
8	Перечисления и коллекции: <i>Организация данных</i>	445
	<i>Лабораторный курс Unity № 4: Пользовательские интерфейсы</i>	493
9	LINQ и лямбда-выражения: <i>Контроль над данными</i>	507
10	Чтение и запись файлов: <i>Прибереги последний байт для меня</i>	569
	<i>Лабораторный курс Unity № 5: Отслеживание лучей</i>	617
11	Капитан Великолепный: <i>Смерть объекта</i>	627
12	Обработка исключений: <i>Борьба с огнем надоедает</i>	663
	<i>Лабораторный курс Unity № 6: Перемещение по сцене</i>	691
I	Проекты ASP.NET Core Blazor: <i>Visual Studio для пользователей Mac</i>	703
II	Ката программирования: <i>Ката программирования для опытных и/или нетерпеливых</i>	765



## Содержание (настоящее)

### Введение

**Ваш мозг и C#.** Вы учитесь — готовитесь к экзамену. Или пытаетесь освоить сложную техническую тему. Ваш мозг хочет оказать вам услугу. Он старается сделать так, чтобы на эту очевидно несущественную информацию не тратились драгоценные ресурсы. Их лучше потратить на что-нибудь важное. Так как же заставить его изучить C#?

Для кого написана эта книга?	30
Кому эта книга не подойдет?	30
Мы знаем, о чем вы думаете	31
Метапознание: наука о мышлении	33
Вот что сделали МЫ	34
Что можете сделать ВЫ, чтобы заставить свой мозг повиноваться	35
Информация	36
Научные редакторы	38
Благодарности	39
И наконец...	39

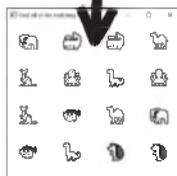




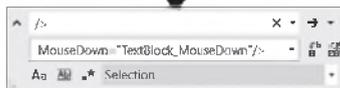
**СОЗДАНИЕ  
ПРОЕКТА**



**КОНСТРУИРОВАНИЕ ОКНА**



**НАПИСАНИЕ  
КОДА C#**



**ОБРАБОТКА  
ЩЕЛЧКОВ**



**ДОБАВЛЕНИЕ  
ТАЙМЕРА**

# 1

## Начало работы с C#

### Быстро сделать что-то классное!

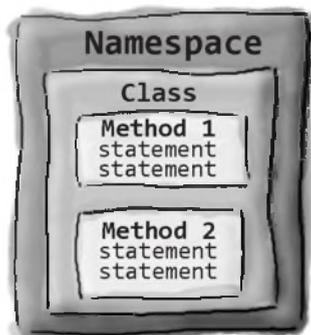
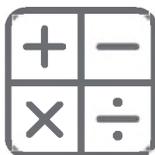
Хотите программировать быстро? C# — это мощный язык программирования. Благодаря Visual Studio вам не потребуется писать непонятный код, чтобы заставить кнопку работать. Вместо того чтобы запоминать параметры метода для имени и для ярлыка кнопки, вы сможете создать действительно классное приложение. Звучит заманчиво? Тогда переверните страницу и приступим к делу.

Зачем вам изучать C#	42
Visual Studio — инструмент для написания кода и изучения C#	43
Создание вашего первого проекта в Visual Studio	44
Давайте построим игру!	46
Как построить игру	47
Создание проекта WPF в Visual Studio	48
Построение окна с использованием XAML	52
Построение окна для игры	53
Определение размера окна и текста заголовка в свойствах XAML	54
Добавление строк и столбцов в сетку XAML	56
Выравнивание размеров строк и столбцов	57
Размещение элементов TextBlock в сетке	58
Теперь можно переходить к написанию кода игры	61
Генерирование метода для настройки игры	62
Завершение метода SetUpGame	64
Запуск программы	66
Добавление нового проекта в систему управления версиями	70
Следующий шаг построения игры — обработка щелчков	73
Реакция TextBlock на щелчки	74
Добавление кода TextBlock_MouseDown	77
Вызов обработчика события MouseDown остальными элементами TextBlock	78
Добавление таймера	79
Добавление таймера в код игры	80
Диагностика ошибок в отладчике	82
Добавьте оставшийся код и завершите построение игры	86
Обновление кода в системе управления версиями	87
Еще лучше, если...	88

# 2 Погружение в C#

## Команды, классы и код

**Вы не просто пользователь IDE. Вы — разработчик.** IDE может сделать за вас очень многое, и все же ее возможности неограничены. Visual Studio — одна из самых совершенных систем разработки программного обеспечения, однако мощная IDE — только начало. Пришло время заняться углубленным изучением кода C#: какую структуру он имеет, как он работает, как управлять им... Потому что нет предела тому, что вы можете делать в ваших приложениях.



Присмотримся к файлам консольного приложения	90
Два класса могут находиться в одном пространстве имен (и файле!)	92
Команды являются структурными элементами приложений	95
Переменные используются в программах для работы с данными	96
Генерирование нового метода для работы с переменными	98
Добавление кода с использованием операторов	99
Использование отладчика для наблюдения за изменением переменных	100
Использование операторов для работы с переменными	102
Принятие решений в командах if	103
Циклы выполняют некоторые действия снова и снова	104
Используйте фрагменты кода для написания циклов	107
Элементы управления определяют механики ваших пользовательских интерфейсов	111
Создание приложения WPF для экспериментов с элементами управления	112
Добавление элемента TextBox в приложение	115
Добавление кода C# для обновления TextBox	118
Добавление обработчика события, который разрешает вводить только числовые данные	119
Добавление ползунков в нижнюю строку сетки	123
Добавление кода C#, обеспечивающего работу элементов управления	124



# Лабораторный курс Unity № 1

## Исследование C# с Unity

Добро пожаловать на первый урок **«Лабораторный курс Unity»**. Написание кода — навык, и, как и любой другой навык, он развивается за счет **практики** и **экспериментирования**. И в этом отношении Unity может стать очень полезным инструментом.

В первой лабораторной работе мы введем вас в курс дела. Вы начнете ориентироваться в редакторе Unity, а также создавать 3D-объекты и оперировать ими.

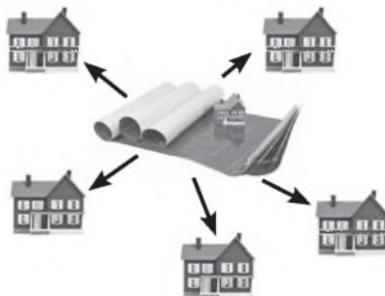
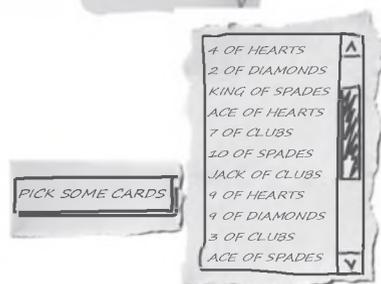
Unity — мощный инструмент для разработки игр	128
Загрузка Unity Hub	129
Использование Unity Hub для создания нового проекта	130
Управление макетом Unity	131
Сцена как 3D-среда	132
Игры Unity состоят из объектов GameObject	133
Использование инструмента Move для перемещения объектов GameObject	134
В окне Inspector выводятся компоненты GameObject	135
Добавление материала к объекту GameObject	136
Вращение сферы	139
Проявите фантазию!	142



# 3 Ориентируемся на объекты

## Написание осмысленного кода

Каждая написанная вами программа решает некоторую задачу. Когда вы пишете программу, всегда желательно заранее подумать, какую задачу должна решать ваша программа. Вот почему объекты приносят такую пользу. Они позволяют сформировать структуру кода в соответствии с решаемой задачей, чтобы вы могли тратить время на задачу, над которой работаете, не отвлекаясь на механику написания кода. Если вы правильно используете объекты (и действительно хорошо продумали их при проектировании), получившийся код будет интуитивно понятным, будет легко читаться и изменяться.



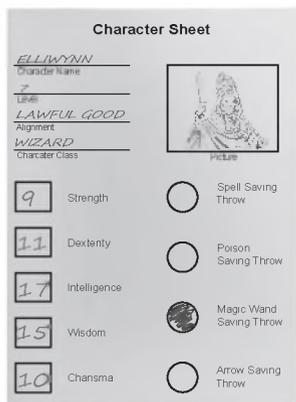
Если код полезен, он используется повторно	144
Некоторые методы получают параметры и возвращают значение	145
Программа для выбора карт	146
Создание консольного приложения PickRandomCards	147
Завершение метода PickSomeCards	148
Готовый класс CardPicker	150
Анна работает над следующей игрой	153
Игра Анны развивается...	154
Построение бумажного прототипа для классической игры	156
Следующий шаг: построение WPF-версии приложения для выбора карт	158
StackPanel — контейнер для наложения элементов	159
Повторное использование класса CardPicker в новом приложении WPF	160
Использование Grid и StackPanel для формирования макета главного окна	161
Формирование макета окна приложения Card Picker	162
Прототипы Анны выглядят замечательно...	165
Анна может воспользоваться объектами для решения своей задачи	166
Класс используется для построения объектов	167
Новый объект, созданный на базе класса, называется экземпляром этого класса	168
Хорошее решение для Анны (с объектами)	169
Экземпляры хранят данные в полях	173
Куча	176
Что на уме у вашей программы	177
Иногда код плохо читается	178
Использование содержательных имен классов и методов	180
Классы, парни и деньги	186
Простой способ инициализации объектов в C#	188
Используйте интерактивное окно C# для выполнения кода C#	194

# 4

## Типы и ссылки

### Данные и ссылки

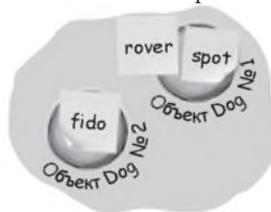
Чем были бы наши приложения без данных? Задумайтесь на минуту. Без данных наши программы... в общем, трудно представить, что кто-то станет писать код без данных. Вы запрашиваете информацию у ваших пользователей; эта информация используется для поиска данных или генерирования новой информации, которая возвращается пользователю. Собственно, практически все, что вы делаете в программировании, требует работы с данными в той или иной форме. В этой главе вы узнаете все тонкости типов данных и ссылок C#, узнаете, как работать с данными в программах, и даже узнаете кое-что новое об объектах (представьте, объекты — тоже данные!).



**Создание ссылки выглядит так, словно вы пишете имя на наклейке и прикрепляете ее к объекту. Надпись становится своего рода «меткой», по которой вы можете обращаться к объекту в будущем.**



Оуэну нужна наша помощь!	196
На листах персонажей хранятся разные виды данных	197
Тип переменной определяет, какие данные в ней могут храниться	198
В C# существует несколько типов для хранения целых чисел	199
Поговорим о строках	201
Литерал — значение, записанное непосредственно в вашем коде	202
Переменные как емкости для данных	205
Другие типы тоже могут иметь разные размеры	206
10 литров в 5-литровой банке	207
Приведение типов позволяет копировать значения, которые C# не может автоматически преобразовать к другому типу	208
C# выполняет некоторые преобразования автоматически	211
При вызове метода аргументы должны быть совместимы с типами параметров	212
Оуэн постоянно старается улучшить свою игру...	214
Поможем Оуэну в экспериментах с характеристиками	216
Использование компилятора C# для поиска проблемной строки кода	218
Использование ссылочных переменных для обращения к объектам	226
Ссылки напоминают наклейки на ваших объектах	227
Если ни одной ссылки не осталось, объект уничтожается сборщиком мусора	228
Множественные ссылки и их побочные эффекты	230
Две ссылки — ДВЕ переменные, по которым можно изменять данные одного объекта	237
Объекты используют ссылки для взаимодействия друг с другом	238
Массивы содержат группы значений	240
Массивы могут содержать ссылочные переменные	241
null означает, что ссылка не указывает ни на что	243
Тест-драйв со случайными числами	247
Добро пожаловать в забегаловку эконом-класса «У неторопливого Джо»!	248

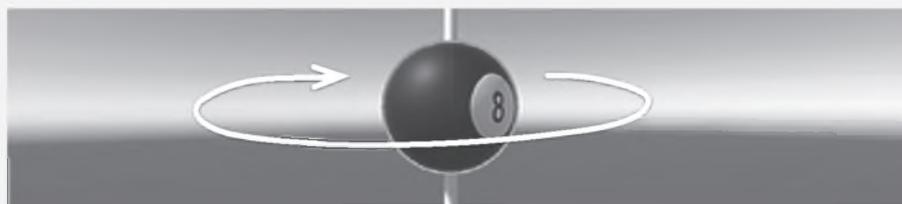


# Лабораторный курс Unity № 2

## Написание кода C# для Unity

Unity – не только мощный кросс-платформенный движок и редактор для построения 2D- и 3D-игр и моделирования. Также это **отличный способ потренироваться в написании кода C#**. В этой лабораторной работе мы начнем писать код для управления объектами GameObject.

Сценарии C# добавляют поведение к объектам GameObject	254
Добавление сценария C# к объекту GameObject	255
Написание кода C# для поворота сферы	256
Добавление точки прерывания и отладка игры	258
Использование отладчика для понимания Time.deltaTime	259
Добавление цилиндра для обозначения оси Y	260
Добавление полей для угла поворота и скорости	261
Debug.DrawRay и 3D-векторы	262
Запуск игры для отображения луча в представлении Scene	263
Поворот шара вокруг точки сцены	264
Эксперименты с поворотами и векторами в Unity	265
Проявите фантазию!	266



## 5

## Инкапсуляция

## Умейте хранить секреты

Вам когда-нибудь хотелось, чтобы посторонние не лезли в ваши личные дела? Вот и вашим объектам этого иногда хочется. И если вы не желаете, чтобы чужие люди читали ваш дневник или просматривали банковские выписки, хорошие объекты не позволяют другим объектам копаться в их полях. В этой главе вы узнаете о мощи инкапсуляции — приеме программирования, который делает ваш код более гибким. Такой код проще использовать и его труднее использовать некорректно. Данные вашего объекта объявляются приватными, и к ним добавляются свойства, защищающие обращения к этим данным.



<b>SwordDamage</b>
Roll
MagicMultiplier
FlamingDamage
Damage
CalculateDamage
SetMagic
SetFlaming



Поможем Оуэну реализовать броски на повреждения	268
Создание консольного приложения для вычисления повреждений	269
Разработка XAML для WPF-версии калькулятора повреждений	271
Код программной части для WPF-калькулятора повреждений	272
Разговор за столом (или, может, дискуссия о кубиках?)	273
Попробуем исправить ошибку	274
Использование Debug.WriteLine для вывода диагностической информации	275
Возможность некорректного использования объектов	278
Инкапсуляция подразумевает ограничение доступа к части данных класса	279
Применение инкапсуляции для управления доступом к методам и полям класса	280
Но ДЕЙСТВИТЕЛЬНО ЛИ поле RealName надежно защищено?	281
К приватным полям и методам могут обращаться только экземпляры того же класса	282
Для чего нужна инкапсуляция? Представьте объект в виде «черного ящика»...	287
Воспользуемся инкапсуляцией для улучшения класса SwordDamage	291
Инкапсуляция обеспечивает безопасность данных	292
Консольное приложение для тестирования класса PaintballGun	293
Свойства упрощают инкапсуляцию	294
Изменение метода Main для использования свойства Balls	295
Автоматически реализуемые свойства упрощают ваш код	296
Использование приватного set-метода для создания свойств, доступных только для чтения	297
А если потребуется изменить размер магазина?	298
Использование конструктора с параметрами для инициализации свойств	299
Передача аргументов при использовании ключевого слова "new"	300



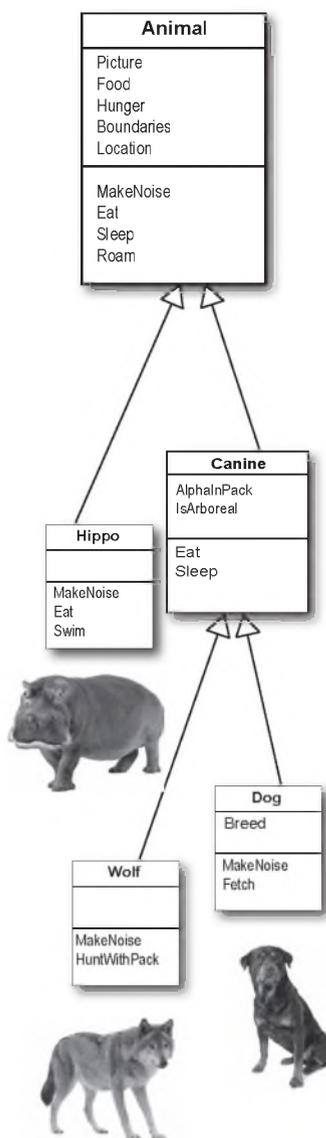
RealName: "Herb Jones"  
 Alias: "Dash Martin"  
 Password: "the crow flies at midnight"

## 6

## Наследование

## Генеалогическое древо объектов

Иногда люди ХОТЯТ быть похожими на своих родителей. Вы встречали объект, который действует почти так, как нужно? Думали ли вы о том, что при изменении всего нескольких элементов класс стал бы идеальным? Наследование позволяет расширять существующие классы, чтобы новый класс получал все поведение существующего — сохраняя при этом гибкость для внесения изменений, чтобы класс можно было адаптировать под любые конкретные требования. Наследование является одним из самых мощных инструментов C#: в частности, оно помогает избегать дублирования кода, более адекватно моделировать реальный мир и в конечном итоге упрощает их сопровождение и снижает риск ошибок.



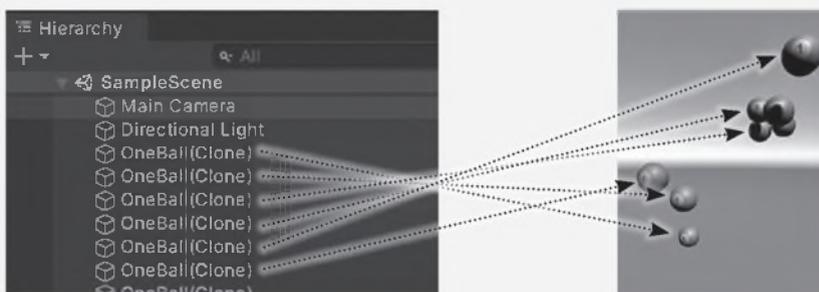
Вычисление повреждений для ДРУГИХ видов оружия	314
Команды switch для выбора из нескольких кандидатов	315
И еще... Можно ли вычислять повреждения от кинжала? От булавы? И песта? И...	317
Если в ваших классах используется наследование, код достаточно написать только один раз	318
Постройте модель классов: начните с общего и переходите к конкретике	319
Как бы вы спроектировали симулятор зоопарка?	320
У разных животных разное поведение	322
Каждый subclass расширяет свой базовый класс	325
Расширение базового класса	330
Subclass может переопределять методы для изменения или замены унаследованных компонентов	332
Некоторые компоненты реализованы только в subclasse	337
Анализ переопределения в отладчике	338
Построение приложения для изучения virtual и override	340
Subclass может скрывать методы базового класса	342
Использование ключевых слов override и virtual для наследования поведения	344
Если базовый класс содержит конструктор, ваш subclass должен его вызвать	347
Subclass и базовый класс могут иметь разные конструкторы	348
Пора доделать приложение для Оуэна	349
Построение системы управления ульем	356
Модель классов системы управления ульем	357
Класс Queen: как матка управляет рабочими	358
Пользовательский интерфейс: добавление кода XAML главного окна	359
Обратная связь направляет работу системы управления ульем	368
Система управления ульем работает в пошаговом режиме...	
Преобразуем ее для работы в реальном времени	370
Экземпляры некоторых классов никогда не должны создаваться	372
Абстрактный класс — намеренно незавершенный класс	374
У абстрактных методов нет тела	377
Абстрактные свойства работают как абстрактные методы	378
Смертельный ромб	381

# Лабораторный курс Unity № 3

## Экземпляры GameObject

Unity – не только мощный кросс-платформенный движок и редактор для построения 2D- и 3D-игр и моделирования. Также это **отличный способ потренироваться в написании кода C#**. В этой лабораторной работе мы начнем писать код для управления объектами GameObject.

Построим игру в Unity!	384
Создайте новый материал в папке Materials	385
Создание бильярдного шара в случайной точке сцены	386
Применение отладчика для понимания Random.value	387
Преобразование объекта GameObject в заготовку	388
Создание сценария для управления игрой	389
Присоединение сценария к главной камере	390
Запустите свой код кнопкой Play	391
Работа с экземплярами GameObject в окне Inspector	392
Предотвращение перекрытия шаров	393
Проявите фантазию!	394



# 7 Интерфейсы, приведение типов и is

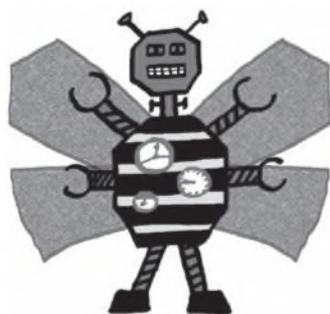
## Классы должны держать обещания

Вам нужен объект для выполнения конкретной задачи? Используйте интерфейс. Иногда возникает необходимость сгруппировать объекты по выполняемым ими функциям, а не по классам, от которых они наследуют. На помощь приходят интерфейсы. Интерфейсы могут использоваться для определения конкретных задач. Любой экземпляр класса, реализующего интерфейс, гарантированно выполняет эту задачу независимо от того, с какими другими классами он связан. Чтобы эта схема работала, каждый класс, реализующий интерфейс, должен гарантировать выполнение всех своих обязательств... иначе программа компилироваться не будет.

Защищать  
улей любой ценой.



Да,  
повелительница!



Улей под атакой!	396
Мы можем воспользоваться приведением типов для вызова метода DefendHive...	397
Интерфейс определяет методы и свойства, которые должны быть реализованы классом...	398
Потренируемся в использовании интерфейсов	400
Создать экземпляр интерфейса невозможно, но можно получить ссылку на интерфейс	406
Ссылки на интерфейсы являются обычными ссылками на объекты	409
RoboBee 4000 может выполнять работу пчел без расхода драгоценного меда	410
Свойство Job в интерфейсе IWorker — костыль	414
Использование is для проверки типа объекта	415
Использование is для обращения к методам subclasses	416
А если мы захотим, чтобы другие животные плавали или охотились в стае?	418
Использование интерфейсов для работы с классами, выполняющими одну задачу	419
В C# также существует другой инструмент для безопасного преобразования типов: ключевое слово as	421
Пример повышающего приведения типа	423
Повышающее приведение преобразует CoffeeMaker в Appliance	424
Повышающие и понижающие приведения типов также работают и с интерфейсами	426
Интерфейсы могут наследовать от других интерфейсов	428
Интерфейсы могут содержать статические компоненты	435
Реализации по умолчанию определяют тело методов интерфейса	436
Добавление метода ScareAdults с реализацией по умолчанию	437
Связывание данных обеспечивает автоматическое обновление элементов WPF	439
Связывание данных в системе управления ульем	440
«Полиморфизм» означает, что один объект может существовать в разных формах	443

## 8

## Перечисления и Коллекции

## Организация данных

Данные не всегда бывают такими аккуратными и ухоженными, как нам хотелось бы. В реальном мире данные, как правило, не хранятся маленькими аккуратными кусочками. Нет, данные поступают вагонами, штабелями и кучами. Для их систематизации нужны мощные инструменты, и тут вам на помощь приходят перечисления и коллекции. Перечисления — типы, позволяющие определять значения для классификации ваших данных. Коллекции — специальные объекты, способные хранить и сортировать данные, которые обрабатывает программа, и управлять ими. В результате вы можете сосредоточиться на основной идее программирования, оставив задачу управления данных коллекциям.

Строки не всегда подходят для хранения категорий данных	446
Перечисления предназначены для работы с наборами допустимых значений	447
Для создания колоды карт можно воспользоваться массивом...	451
С массивами бывает неудобно работать	452
В списках можно хранить коллекции... чего угодно	453
Списки обладают большей гибкостью, чем массивы	454
Построим приложение для хранения обуви	457
В обобщенных коллекциях могут храниться любые типы	460
Инициализаторы коллекций похожи на инициализаторы объектов	466
Создание списка уток	467
Списки удобны, но с сортировкой могут возникнуть проблемы	468
Comparable<Duck> помогает списку List сортировать объекты Duck	469
Использование Comparable для определения порядка сортировки	470
Создание экземпляра компаратора	471
Компараторы могут выполнять сложные сравнения	472
Переопределение метода ToString позволяет объекту описать себя	475
Обновите циклы foreach, чтобы объекты Duck и Card выводили свои описания на консоль	476
Использование Dictionary для хранения ключей и значений	482
Краткая сводка функциональности Dictionary	483
Построение программы с использованием словаря	484
Другие разновидности коллекций...	485
Очередь работает по принципу FIFO — «первым вошел, первым вышел»	486
Стек работает по принципу LIFO — «последним вошел, первым вышел»	487
Упражнение: две колоды	492



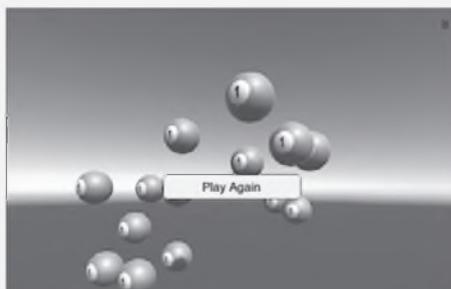
Карта «Герцог быков».  
В природе не встречается.

# Лабораторный курс Unity № 4

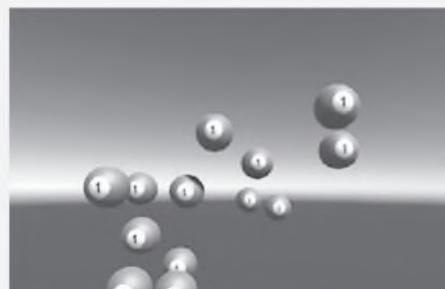
## Пользовательские интерфейсы

В предыдущей лабораторной работе Unity вы начали строить игру. Мы использовали заготовку для создания экземпляров `GameObject`, которые появлялись в случайных точках трехмерного пространства игры и летали по кругу. В этой лабораторной работе мы продолжим с того места, на котором остановились в предыдущей главе; в ней вы сможете применить то, что узнали об интерфейсах C#, и многое другое.

Вывод текущего счета	494
Включение двух режимов в игру	495
Добавление игрового режима	496
Добавление пользовательского интерфейса к игре	498
Настройка объекта <code>Text</code> для вывода счета в UI	499
Кнопка для вызова метода, запускающего игру	500
Кнопка <code>Play Again</code> и текущий счет	501
Завершение кода игры	502
Проявите фантазию!	506



На этом снимке экрана показана игра в рабочем режиме. Шары добавляются в сцену, а игрок может щелкать на них, чтобы получать очки.



Когда на экране появится последний шар, игра переходит в режим завершения. На экране появляется кнопка `Play Again`, и новые шары перестают появляться.

# 9 LINQ и лямбда-выражения

## Контроль над данными

Этим миром правят данные... И нам нужно знать, как в нем жить. Прошли те времена, когда можно было программировать днями и даже неделями, не имея дела с огромными объемами данных. В наши дни данные стали сутью любой программы. LINQ – технология C# и .NET, которая позволяет не только обращаться с запросами к данным в коллекциях .NET на интуитивно понятном уровне, но и группировать данные и выполнять слияние данных из разных источников. Модульные тесты помогут убедиться в том, что ваш код работает так, как предполагалось. А когда вы освоитесь с задачей разбиения данных на блоки, с которыми удобно работать, вы можете воспользоваться лямбда-выражениями, провести рефакторинг кода C# и сделать его еще более выразительным.



Джимми – фанат Капитана Великолепного...	508
Использование LINQ для управления коллекциями	510
LINQ работает с любыми реализациями IEnumerable<T>	512
Синтаксис запросов LINQ	515
LINQ работает с объектами	517
Использование запроса LINQ в приложении для Джимми	518
Ключевое слово var позволяет C# определить тип переменной за вас	520
Запросы LINQ выполняются только при обращении к результатам	527
Использование запросов group для разделения последовательности на группы	528
Использование запросов join для слияния данных из двух последовательностей	531
Использование ключевого слова new для создания анонимных типов	532
Модульные тесты помогают понять, как работает код	540
Добавление проекта модульного теста в приложение Джимми	542
Первый модульный тест	543
Написание модульного теста для метода GetReviews	545
Написание модульных тестов для обработки граничных случаев и аномальных данных	546
Оператор => и создание лямбда-выражений	548
Тест-драйв лямбда-выражений	549
Рефакторинг клоунов с использованием лямбда-выражений	550
Использование оператора ?: для принятия решений в лямбда-выражениях	553
Лямбда-выражения и LINQ	554
Запросы LINQ могут записываться в виде сцепленных вызовов методов LINQ	555
Использование оператора => для создания выражений switch	557
Исследование класса Enumerable	561
Ручное создание последовательности с поддержкой перебора	562
Упражнение: Go Fish	567

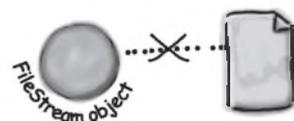
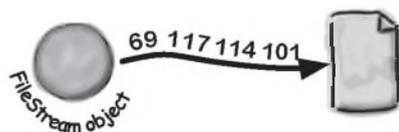


## 10

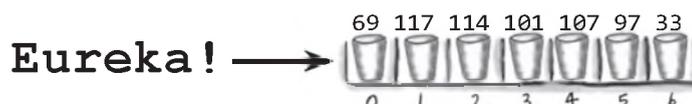
## Чтение и запись файлов

## Прибереги последний байт для меня

**Иногда настойчивость окупается.** Пока что все ваши программы жили недолго. Они запускались, некоторое время работали и закрывались. Но этого недостаточно, когда имеешь дело с важной информацией. Вы должны уметь сохранять свою работу. В этой главе мы поговорим о том, как записать данные в файл, а затем о том, как прочитать эту информацию. Вы познакомитесь с потоками данных, узнаете о сохранении объектов в файлах с использованием сериализации, а также освоите работу с шестнадцатеричными и двоичными данными и кодировку Юникод.



Для чтения и записи данных в .NET используются потоки данных	570
Различные потоки для разных данных	571
Объект FileStream читает и записывает байты в файл	572
Запись текста в файл за три простых шага	573
Дьявольский план Пройдохи	574
Использование StreamReader для чтения файла	577
Данные могут проходить через несколько потоков	578
Работа с файлами и каталогами с использованием статических классов File и Directory	582
Интерфейс IDisposable обеспечивает корректное закрытие объектов	585
Предотвращение ошибок файловой системы командами using	586
Потоки MemoryStream и хранение данных в памяти	587
При сериализации объекта также сериализуются все объекты, на которые он ссылается...	595
Использование JsonSerialization для сериализации объектов	596
JSON включает только данные, но не конкретные типы C#	599
Следующий шаг: углубленный анализ данных	601
Строки C# кодируются в Юникоде	603
Поддержка Юникода в Visual Studio	605
.NET использует Юникод для хранения символов и текста	606
C# может использовать массивы байтов для перемещения данных	608
Использование BinaryWriter для записи двоичных данных	609
Использование BinaryReader для чтения данных	610
Дамп позволяет просматривать байты в файлах	612
Использование StreamReader для вывода шестнадцатеричного дампа	613
Использование Stream.Read для чтения байтов из потока	614
Аргументы командной строки	615
Упражнение: Hide and Seek	616



# Лабораторный курс Unity № 5

## Отслеживание лучей

Создавая сцену в Unity, вы создаете виртуальный 3D-мир, в котором перемещаются персонажи вашей игры. Но в большинстве игр объекты окружающей обстановки не контролируются игроком напрямую. Как же эти объекты определяют свое место в сцене? В этой лабораторной работе мы построим сцену из объектов GameObject и используем навигацию для перемещения персонажей по сцене.

Создание нового проекта Unity и начало создания сцены	618
Настройка камеры	619
Создание объекта GameObject для игрока	620
Знакомство с системой навигации Unity	621
Создание сетки NavMesh	622
Автоматическая навигация в игровой области	623



# Капитан Великолепный

## Смерть объекта

Head First C#	
Четыре доллара	Глава 11



Жизнь и смерть объекта	630
Для принудительной сборки мусора используйте класс GC (осторожно!)	631
Когда именно выполняется финализатор?	633
Финализаторы не могут зависеть от других объектов	635
Структура похожа на объект...	639
...но не является объектом	639
Значения копируются, ссылки присваиваются	640
Структуры относятся к типам значений; объекты относятся к ссылочным типам	641
Стек и куча: подробнее о памяти	643
Параметры out и возвращение нескольких значений методом	646
Передача по ссылке с модификатором ref	647
Необязательные параметры и значения по умолчанию	648
Ссылка null не указывает ни на какой объект	649
Ссылочные типы, не допускающие null, помогут избежать NRE	650
Оператор объединения с null ??	651
Безопасная работа с типами значений, допускающими null	652
Капитан... уже не такой Великолепный	653
Методы расширения добавляют новое поведение в существующие классы	657
Расширение фундаментального типа: string	659

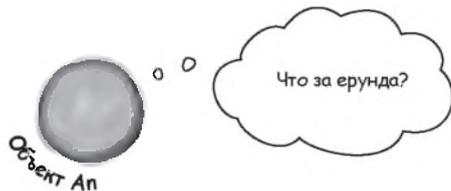


- У МЕНЯ... ОСТАЛОСЬ...  
-ОХ! -  
ОДНО... ПОСЛЕДНЕЕ... ДЕЛО...

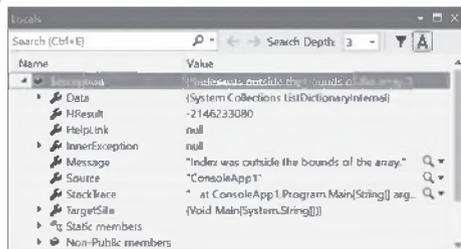
# 12 Обработка исключений

## Борьба с огнем надоедает

Программисты не должны уподобляться пожарным. Вы усердно работали, штудировали справочники и руководства и, наконец, достигли вершины. Но вам до сих пор продолжают звонить с работы по ночам, потому что программа упала или работает не так, как должна работать. Ничто так не выбивает из колеи, как необходимость устранять странные ошибки... но благодаря обработке исключений вы сможете написать код, который сам будет разбираться с возможными проблемами. А еще лучше, что вы можете планировать такие проблемы и восстанавливать работоспособность программы при их возникновении.



```
int[] anArray = {3, 4, 1, 11};
int aValue = anArray[15];
```



- Программа вывода шестнадцатеричного дампа читает имя файла из командной строки 664
- Когда ваша программа выдает исключение, CLR генерирует объект Exception 668
- Все объекты Exception наследуют от System.Exception 669
- Для некоторых файлов вывод дампа невозможен 672
- Что происходит при вызове небезопасного метода? 673
- Обработка исключений с try и catch 674
- Отслеживание передачи управления в try/catch 675
- Код блока finally выполняется всегда 676
- Перехват всех исключений 677
- Использование исключения, подходящего для конкретной ситуации 682
- Фильтры исключений повышают точность обработки исключений 686
- Наихудший блок catch: универсальный перехват с комментариями 688
- Временные решения допустимы (но только временно) 689

# Лабораторный курс Unity № 6

## Перемещение по сцене

В последней лабораторной работе Unity была создана сцена с полом (плоскость) и игроком (сфера с цилиндром). При этом использовался объект NavMesh, NavMesh Agent и отслеживание лучей, чтобы игрок следовал за щелчками в сцене. В этой работе мы воспользуемся навигационной системой Unity, чтобы объекты GameObject сами перемещались по сцене.

Продолжим с того места, на котором прервалась последняя лабораторная работа Unity	692
Добавление платформы в сцену	693
Изменение настроек предварительного построения	694
Включение лестницы и наклонной плоскости в NavMesh	695
Решение проблем с высотой в NavMesh	697
Добавление препятствия в сетку NavMesh	698
Добавление сценария для перемещения препятствия вверх и вниз	699
Проявите фантазию!	700

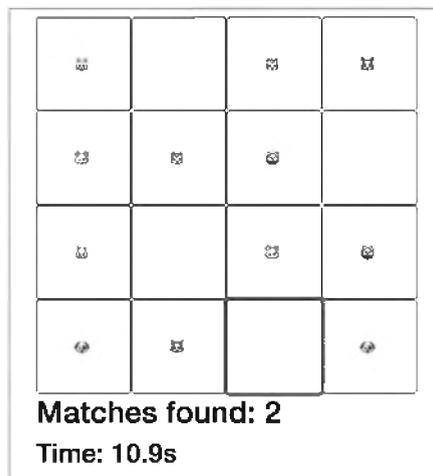


Это препятствие NavMesh создает движущийся проем в NavMesh, который мешает игроку перемещаться вверх по наклонной плоскости. Мы добавим сценарий, который позволяет игроку перетаскивать его мышью, чтобы блокировать и освобождать наклонную плоскость.

# I

## Приложение I. Проекты ASP.NET Core Blazor

### Visual Studio для пользователей Mac



Зачем вам изучать C#	704
Создание вашего первого проекта в Visual Studio for Mac	706
Давайте построим игру!	710
Как построить игру	711
Создание проекта Blazor WebAssembly в Visual Studio	712
Запуск веб-приложения Blazor в браузере	714
Visual Studio помогает в написании кода C#	718
Завершение создания списка эмодзи и вывод их в приложении	720
Перестановка животных в случайном порядке	722
Добавление нового проекта в систему управления версиями	728
Добавление кода C# для обработки щелчков	729
Назначение обработчиков щелчков кнопкам	730
Тестирование обработчика события	732
Диагностика проблемы в отладчике	733
Отладка обработчика события	734
Поиск ошибки, породившей проблему...	736
Добавление кода для сброса игры при победе	738
Добавление таймера	741
Добавление таймера в код игры	742
Очистка меню навигации	744
Создание нового проекта Blazor WebAssembly App	747
Создание страницы с ползунком	748
Добавление текстового поля	750
Добавление селекторов для выбора цвета и даты	753
Построение Blazor-версии приложения для выбора карт	754
Страница состоит из строк и столбцов	756
Ползунок использует связывание данных для обновления переменной	757
Добро пожаловать в забегаловку эконом-класса «У неторопливого Джо»!	760

# II

## Приложение II. Ката программирования

### Ката программирования для опытных и/или нетерпеливых

