

Содержание (сводка)

| | | |
|----|---|-----|
| | Введение | 29 |
| 1 | Сделать что-то классное... и побыстрее! | 41 |
| 2 | Стремительный полет в C# | 105 |
| | <i>Лабораторный курс Unity № 1: Исследование C# с Unity</i> | 151 |
| 3 | Организация кода | 167 |
| 4 | Управление данными приложения | 229 |
| | <i>Лабораторный курс Unity № 2: Написание кода C# для Unity</i> | 297 |
| 5 | Как объекты хранят свои секреты | 311 |
| 6 | Генеалогическое древо объектов | 365 |
| | <i>Лабораторный курс Unity № 3: Экземпляры GameObject</i> | 443 |
| 7 | Классы должны держать обещания | 455 |
| 8 | Организация данных | 513 |
| | <i>Лабораторный курс Unity № 4: Пользовательские интерфейсы</i> | 579 |
| 9 | Контроль над данными | 593 |
| 10 | Прибереги последний байт для меня | 661 |
| | <i>Лабораторный курс Unity № 5: Отслеживание лучей</i> | 713 |
| 11 | Captain Amazing: Смерть объекта | 727 |
| 12 | Борьба с огнем надоедает | 771 |
| | <i>Лабораторный курс Unity № 6: Навигация по сцене</i> | 803 |

Содержание (настоящее)

Введение

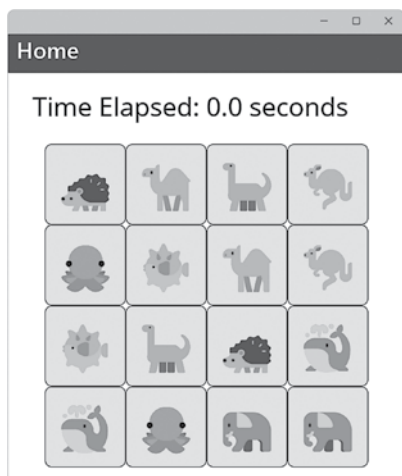
Ваш мозг и C#. Вы пытаетесь чему-то научиться, но ваш мозг «помогает» вам, делая все возможное, чтобы новые знания не закрепились. Он рассуждает примерно так: лучше оставить место для более важных вещей — например, поразмыслить о том, каких диких животных стоит избегать и плохая ли идея кататься на сноуборде голышом. Так как же обмануть мозг и заставить его думать, что от знания C# зависит ваша жизнь?

| | | |
|--|---|----|
| | Для кого написана эта книга? | 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 |
| | «Плечи гигантов» | 39 |
| | Благодарности | 40 |

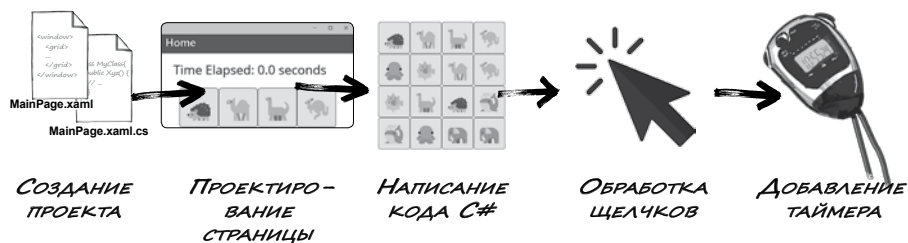
1 Начало работы с C# Сделать что-то классное... и побыстрее!

Хотите поскорее научиться программировать?

C# — современный и мощный язык программирования. С **Visual Studio** вы получаете в распоряжение великолепную, интуитивно понятную среду разработки, которая старается сделать процесс программирования по возможности простым и приятным. Но Visual Studio — не только превосходный инструмент для написания кода, это также **чрезвычайно эффективный учебный инструмент** для освоения C#. Звучит заманчиво? Тогда за дело!



| | |
|--|-----|
| Изучайте C# и узнайте, как стать настоящим разработчиком... | 42 |
| Visual Studio — инструмент для написания кода и изучения C# | 43 |
| Установка Visual Studio Community Edition | 44 |
| Запуск Visual Studio | 45 |
| Создание и запуск первого проекта в Visual Studio | 46 |
| Использование Visual Studio Code# | 52 |
| Создание и запуск первого проекта в Visual Studio Code | 54 |
| Подготовка Visual Studio Code для следующего проекта | 57 |
| Давайте построим игру! | 58 |
| Создание проекта .NET MAUI в Visual Studio | 62 |
| Запуск приложения .NET MAUI | 64 |
| Приложения MAUI работают на всех устройствах | 65 |
| Начало редактирования кода XAML | 67 |
| Использование элемента FlexLayout для отображения сетки кнопок | 74 |
| Написание кода C# для добавления животных на кнопки | 78 |
| Запустите свое приложение! | 86 |
| Visual Studio упрощает работу с Git | 91 |
| Добавление кода C# для обработки щелчков | 92 |
| Добавление таймера в код игры | 100 |
| Допишите код своей игры | 102 |

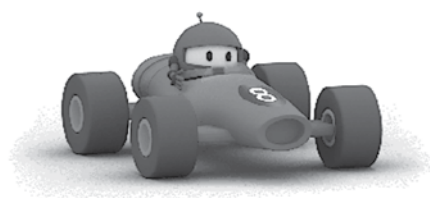
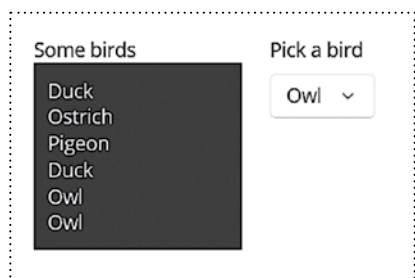


2 Переменные, Команды и Методы

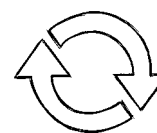
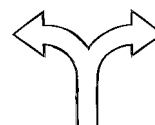
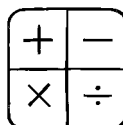
Стремительный полет в C#

Вы не просто пользователь IDE. Вы — разработчик.

IDE может сделать за вас очень многое, и все же ее возможности не безграничны. 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 | 106 |
| Команды – структурные элементы приложений | 108 |
| Команды находятся в методах | 109 |
| Переменные используются в программах для работы с данными | 110 |
| Генерирование нового метода для работы с переменными | 112 |
| Добавление кода с использованием операторов | 113 |
| Использование отладчика для наблюдения за изменением переменных | 114 |
| Используйте фрагменты кода для написания циклов | 116 |
| Использование операторов для работы с переменными | 117 |
| Принятие решений в командах if | 118 |
| Циклы выполняют некоторые действия снова и снова | 119 |
| Элементы управления определяют механику ваших пользовательских интерфейсов | 128 |
| Другие элементы, используемые в книге | 129 |
| Создание нового приложения для экспериментов с элементами управления | 131 |
| Исследуйте новое приложение MAUI и разберитесь, как оно работает | 132 |
| Добавление элемента Entry в приложение | 136 |
| Добавление свойств к элементу Entry | 137 |
| Обновление Label при изменении элемента Entry | 138 |
| Объединение горизонтального размещения с вертикальным | 143 |
| Добавление элемента Picker для вывода списка вариантов | 144 |

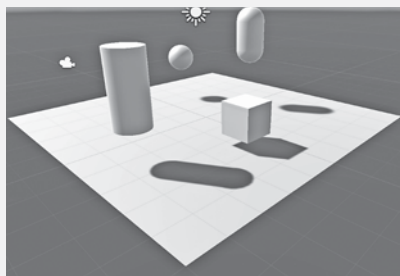


Лабораторный курс Unity № 1

Исследование C# с Unity

Добро пожаловать на первый урок «**Head First C# 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 этом лабораторном курсе вы сможете начать применять на практике то, что узнали о C# в главах 1 и 2.

| | |
|--|-----|
| Unity — мощный инструмент для разработки игр | 152 |
| Загрузка Unity Hub | 153 |
| Использование Unity Hub для создания нового проекта | 154 |
| Сцена как 3D-среда | 156 |
| Игры Unity состоят из объектов GameObject | 157 |
| Использование инструмента Move для перемещения объектов GameObject | 158 |
| В окне Inspector выводятся компоненты GameObject | 159 |
| Добавление материала к объекту GameObject | 160 |
| Вращение сферы | 163 |
| Проявите фантазию! | 166 |



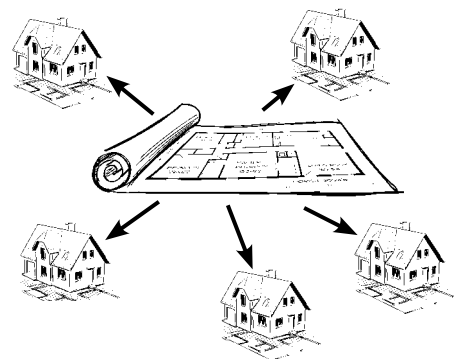
3

Пространства имен и классы

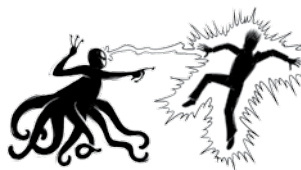
Организация кода

У хорошего разработчика код и данные всегда четко организованы.

С чего начинается создание приложения? Вы прежде всего думаете, что **оно должно делать**, независимо от того, решаете вы научную задачу, создаете игру или просто развлекаетесь. Но не всегда очевидно, как отдельные команды вписываются в общую картину приложения... и тогда на помощь приходят **классы**. Они позволяют **структурировать код** вокруг создаваемой функциональности и задач, которые приложение должно решать. Классы также помогают **организовать данные** — они используются для создания **объектов**, представляющих любые сущности, используемые в вашей программе. Класс служит своего рода чертежом, по которому строятся объекты, используемые в приложении.



| | |
|---|-----|
| Классы помогают организовать код | 168 |
| Некоторые методы получают параметры и возвращают значения | 170 |
| Приложение для выбора карт | 172 |
| Создание приложения с методом Main | 174 |
| Используйте быстрые действия для удаления лишних строк с using | 178 |
| Преобразование стилей пространств имен | 179 |
| Использование ключевого слова new для создания массива строк | 180 |
| Построение бумажного прототипа для классической игры | 188 |
| Построение MAUI-версии приложения для выбора карт | 190 |
| Повторное использование класса CardPicker | 194 |
| Директива using для использования кода из другого пространства имен | 195 |
| Класс используется для построения объектов | 199 |
| Хорошее решение для Анны (с объектами) | 201 |
| Экземпляры хранят данные в полях | 205 |
| Использование содержательных имен классов и методов | 212 |
| Классы, парни и деньги | 218 |
| Используйте интерактивное окно C# или csi для выполнения кода C# | 228 |



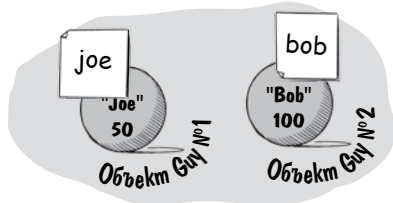
4

Данные, типы, объекты и ссылки

Управление данными приложения

Данные и объекты — структурные элементы, из которых строятся ваши приложения.

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 изучите все тонкости **типов данных и ссылок C#**, поймете, как работать с данными в программах, и даже узнаете кое-что новое об объектах (представьте, объекты — тоже данные!).



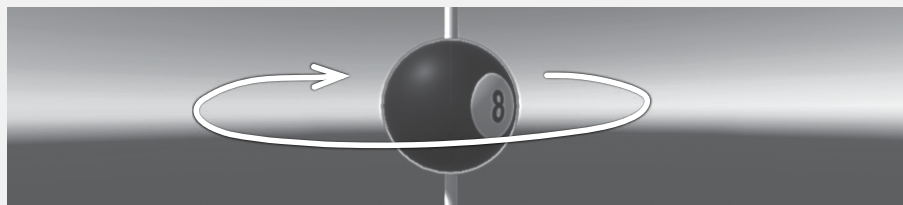
| | |
|--|-----|
| Тип переменной определяет, какие данные в ней могут храниться | 232 |
| В C# существует несколько типов для хранения целых чисел | 233 |
| Поговорим о строках | 235 |
| Литерал — значение, записанное непосредственно в вашем коде | 236 |
| Приведение типов позволяет копировать значения, которые C# не может автоматически преобразовать к другому типу | 242 |
| C# выполняет некоторые преобразования автоматически | 245 |
| Использование ссылочных переменных для обращения к объектам | 262 |
| Ссылки напоминают наклейки на ваших объектах | 263 |
| Множественные ссылки и их побочные эффекты | 266 |
| Две ссылки — ДВЕ переменные, по которым можно изменять данные одного объекта | 273 |
| Объекты используют ссылки для взаимодействия друг с другом | 274 |
| Массивы содержат группы значений | 276 |
| null означает, что ссылка не указывает ни на что | 281 |
| Используйте тип string?, если строка может содержать null | 283 |
| Добро пожаловать в забегаловку эконом-класса «У неторопливого Джо»! | 286 |
| Элементы Grid | 288 |
| Создание приложения для меню и настройка элемента Grid | 290 |
| Изменение семантических свойств элемента методом SetValue | 296 |

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 продолжим работу над проектом в Unity.

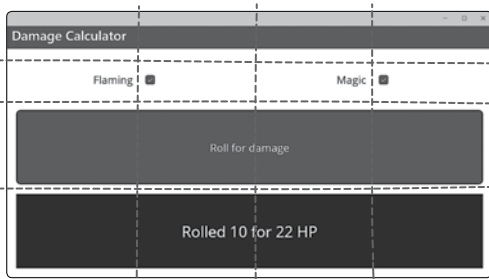
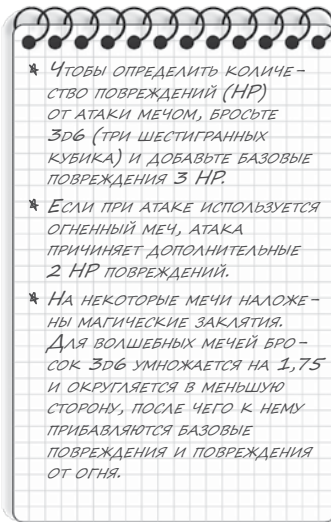
| | |
|--|-----|
| Сценарии C# добавляют поведение к объектам GameObject | 298 |
| Добавление сценария C# к объекту GameObject | 299 |
| Написание кода C# для поворота сферы | 300 |
| Добавление точки прерывания и отладка игры | 302 |
| Использование отладчика для понимания Time.deltaTime | 303 |
| Добавление цилиндра для обозначения оси Y | 304 |
| Добавление полей для угла поворота и скорости | 305 |
| Debug.DrawRay и 3D-векторы | 306 |
| Запуск игры для отображения луча в представлении Scene | 307 |
| Поворот шара вокруг точки сцены | 308 |
| Эксперименты с поворотами и векторами в Unity | 309 |
| Проявите фантазию! | 310 |



5 Инкапсуляция

Как объекты хранят свои секреты

Вам когда-нибудь хотелось, чтобы посторонние не лезли в ваши 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 — механизме программирования, который делает ваш код более гибким. Такой код трудно использовать некорректно. **Данные вашего объекта объявляются приватными**, и к ним добавляются свойства, защищающие обращения к данным, — таким образом предотвращается **доступ к важным данным со стороны других объектов** и случайные злоупотребления ими.



| | |
|---|-----|
| Поможем Оуэну реализовать броски на повреждения | 312 |
| Создание консольного приложения для вычисления повреждений | 313 |
| Разработка MAUI-версии калькулятора повреждений | 315 |
| Использование Debug.WriteLine для вывода диагностической информации | 321 |
| Применение инкапсуляции для управления доступом к методам и полям класса | 326 |
| К приватным полям и методам могут обращаться только экземпляры того же класса | 328 |
| Для чего нужна инкапсуляция? Представьте объект в виде «черного ящика»... | 333 |
| Воспользуемся инкапсуляцией для улучшения класса SwordDamage | 337 |
| Консольное приложение для тестирования класса PaintballGun | 339 |
| Автоматически реализуемые свойства упрощают ваш код | 342 |
| Использование приватного set-метода для создания свойств, доступных только для чтения | 343 |
| Использование конструктора с параметрами для инициализации свойств | 345 |
| Передача аргументов при использовании ключевого слова new | 346 |
| Инициализация полей и свойств — встроенная или в конструкторе | 353 |

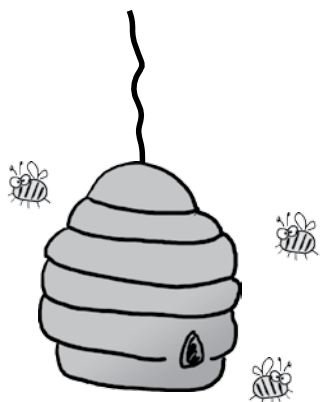
Наследование

6

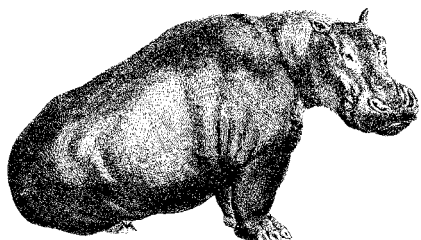
Генеалогическое древо объектов

Иногда люди ХОТЯТ быть похожими на своих родителей.

Вы встречали класс, который действует почти так, как нужно? Думали ли вы о том, что при изменении **всего** нескольких элементов класс стал бы идеальным? **Наследование** позволяет расширить существующие классы, чтобы новый класс получал все поведение существующего, сохраняя при этом **гибкость** для внесения изменений, чтобы его 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 счете **упрощает сопровождение и снижает риск ошибок**.



| | |
|--|-----|
| Команды switch для выбора из нескольких кандидатов | 367 |
| Если в ваших классах используется наследование, код достаточно написать только один раз | 370 |
| Как бы вы спроектировали симулятор зоопарка? | 372 |
| В любом месте, где может использоваться базовый класс, вместо него можно использовать один из субклассов | 378 |
| Субкласс может переопределять методы для изменения или замены унаследованных компонентов | 384 |
| Построение приложения для изучения virtual и override | 392 |
| Субкласс может скрывать методы базового класса | 394 |
| Использование ключевых слов override и virtual для наследования поведения | 396 |
| Класс должен делать что-то одно | 406 |
| Построение системы управления ульем | 410 |
| Обратная связь направляет работу системы управления ульем | 428 |
| Система управления ульем работает в пошаговом режиме... | |
| Преобразуем ее для работы в реальном времени | 430 |
| Абстрактный класс – намеренно незавершенный класс | 434 |
| Абстрактные свойства работают как абстрактные методы | 438 |
| Смертельный ромб | 441 |

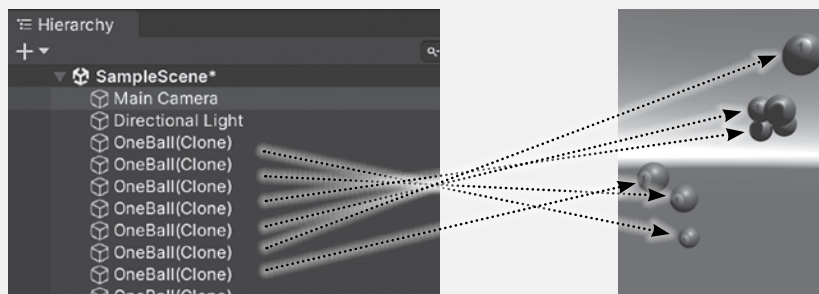


Лабораторный курс Unity №3

Экземпляры GameObject

C# – объектно-ориентированный язык, и поскольку эти лабораторные работы Head First C# Unity посвящены **практике написания кода на C#**, логично, что они будут сосредоточены на создании объектов.

| | |
|---|-----|
| Построим игру в Unity! | 444 |
| Создайте новый материал в папке Materials | 445 |
| Создание бильярдного шара в случайной точке сцены | 446 |
| Применение отладчика для понимания Random.value | 447 |
| Преобразование объекта GameObject в заготовку | 448 |
| Создание сценария для управления игрой | 449 |
| Присоединение сценария к главной камере | 450 |
| Запустите свой код кнопкой Play | 451 |
| Работа с экземплярами GameObject в окне Inspector | 452 |
| Предотвращение перекрытия шаров | 453 |
| Проявите фантазию! | 454 |



Интерфейсы, приведение типов и is

Классы должны держать обещания

Вам нужен объект для выполнения конкретной работы? 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.



| | |
|--|-----|
| Улей под атакой! | 456 |
| Можно воспользоваться приведением типов для вызова метода DefendHive... | 457 |
| Интерфейс определяет методы и свойства, которые должны быть реализованы классом... | 458 |
| Интерфейсы позволяют несвязанным классам выполнять одну задачу | 459 |
| Потренируемся с использованием интерфейсов | 460 |
| Создать экземпляр интерфейса невозможно, но можно получить ссылку на интерфейс | 466 |
| Ссылки на интерфейсы являются обычными ссылками на объекты | 469 |
| RoboBee 4000 может выполнять работу пчел без расхода драгоценного меда | 470 |
| А если мы захотим, чтобы другие животные плавали или охотились в стаях? | 478 |
| Использование интерфейсов для работы с классами, выполняющими одну задачу | 479 |
| is и безопасная навигация по иерархии классов | 480 |
| В C# также существует другой инструмент для безопасного преобразования типов: ключевое слово as | 481 |
| Использование повышающего и понижающего приведения типа для перемещения вверх и вниз по иерархии классов | 482 |
| Повышающие и понижающие приведения типов также работают и с интерфейсами | 486 |
| Реализации по умолчанию определяют тело методов интерфейса | 496 |
| Связывание данных обеспечивает автоматическое обновление элементов MAUI | 499 |
| «Полиморфизм» означает, что один объект может существовать в разных формах | 509 |

Защищать улей любой ценой.



Да, повелительница!



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.

Карта «Герцог быков».
В природе не встречается.



| | |
|--|-----|
| Если конструктор просто задает значения полей, используйте главный конструктор | 514 |
| Главный конструктор может расширять конструктор базового класса | 515 |
| Перечисления предназначены для работы с наборами допустимых значений | 517 |
| Перечисления позволяют представлять числа именами | 518 |
| В списках можно хранить коллекции... чего угодно | 523 |
| Построим приложение для хранения обуви | 527 |
| В обобщенных коллекциях могут храниться любые типы | 530 |
| Использование выражений коллекций при создании списков | 536 |
| IComparable<Duck> помогает списку List сортировать объекты Duck | 539 |
| Создание экземпляра компаратора | 541 |
| Компараторы могут выполнять сложные сравнения | 542 |
| Повышающее приведение типа всего списка с использованием IEnumerable<T> | 550 |
| Краткая сводка функциональности Dictionary | 553 |
| CollectionView — элемент MAUI для вывода коллекций | 564 |
| ObservableCollection — коллекция для связывания данных | 565 |
| Создание объектов для связывания данных в XAML | 569 |
| Измените приложение для использования словаря ресурсов | 570 |
| Измените обработчики событий, чтобы использовать словарь ресурсов | 572 |
| Используйте полученные знания для построения приложения | 573 |



По породе... →

Лабораторный курс Unity №4

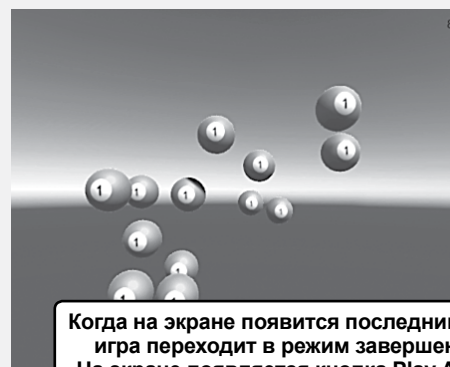
Пользовательские интерфейсы

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 | 580 |
| Включение двух режимов в игру | 581 |
| Добавление игрового режима | 582 |
| Добавление пользовательского интерфейса к игре | 584 |
| Настройка объекта <code>Text</code> для вывода счета в UI | 585 |
| Кнопка для вызова метода, запускающего игру | 586 |
| Кнопка <code>Play Again</code> и текущий счет | 587 |
| Завершение кода игры | 588 |
| Проявите фантазию! | 592 |



Щелкните на ссылке `Installs`, чтобы управлять установленными версиями Unity.



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

LINQ и Лямбда-Выражения

9

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 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 сможете воспользоваться **лямбда-выражениями**, провести рефакторинг кода C# и сделать его еще более выразительным.



| | |
|--|-----|
| Джимми — фанат Капитана Великолепного... | 594 |
| Использование LINQ для управления коллекциями | 596 |
| Использование запроса LINQ в приложении для Джимми | 604 |
| Ключевое слово var позволяет C# определить тип переменной за вас | 606 |
| Гибкость LINQ | 612 |
| Использование запросов group для разделения последовательности на группы | 614 |
| Использование запросов join для слияния данных из двух последовательностей | 617 |
| Использование ключевого слова new для создания анонимных типов | 618 |
| Модульные тесты помогают понять, как работает код | 627 |
| Создание первого тестового метода | 628 |
| Проект может обращаться только к открытым классам другого проекта | 630 |
| Применение паттерна AAA для написания эффективных тестов | 631 |
| Написание модульного теста для метода GetReviews | 634 |
| Оператор => и создание лямбда-выражений | 638 |
| Использование оператора ?: для принятия решений в лямбда-выражениях | 643 |
| LINQ-запросы состоят из методов | 644 |
| Декларативный синтаксис LINQ можно преобразовать в сцепленные методы | 646 |
| Использование оператора => для создания выражений switch | 649 |
| Исследование класса Enumerable | 653 |
| Использование yield return для создания собственной последовательности | 655 |

10

Чтение и запись файлов

Прибереги последний байт для меня

Иногда долгосрочное планирование окупается.

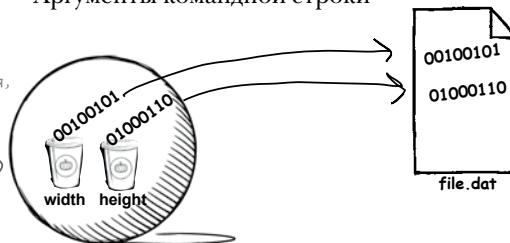
Пока что все ваши программы жили недолго. Они запускались, некоторое время работали и закрывались. Но этого недостаточно, когда имеешь дело с важной информацией. Вы должны уметь **сохранять свою работу**. В этой главе мы поговорим о том, как **записать данные в файл**, а затем о том, как **прочитать 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».



```
0000: 45 6c 65 6d 65 6e 74 61 Elementa
0005: 72 79 2c 20 6d 79 20 64 ry, my d
0010: 65 61 72 20 57 61 74 73 ear Wats
0015: 6f 6e 21 on!
```

| | |
|---|-----|
| Для чтения и записи данных в .NET используются потоки данных | 662 |
| Различные потоки для разных данных | 663 |
| Использование StreamReader для чтения файла | 669 |
| Работа с файлами и каталогами с использованием статических классов File и Directory | 674 |
| Интерфейс IDisposable обеспечивает корректное закрытие объектов | 677 |
| Предотвращение ошибок файловой системы командами using | 678 |
| Потоки MemoryStream и хранение данных в памяти | 679 |
| Что происходит с объектами при сериализации? | 685 |
| Использование JsonSerializer для сериализации объектов | 688 |
| JSON включает только данные, но не конкретные типы C# | 691 |
| Строки C# кодируются в «Юникоде» | 695 |
| .NET использует «Юникод» для хранения символов и текста | 698 |
| C# может использовать массивы байтов для перемещения данных | 700 |
| Использование BinaryWriter для записи двоичных данных | 701 |
| Использование BinaryReader для чтения данных | 702 |
| Использование StreamReader для вывода шестнадцатеричного дампа | 705 |
| Использование Stream.Read для чтения байтов из потока | 706 |
| Изменение программы вывода дампа для прямого чтения из потока | 707 |
| Аргументы командной строки | 708 |

Этот объект имеет два байтовых поля, width и height.

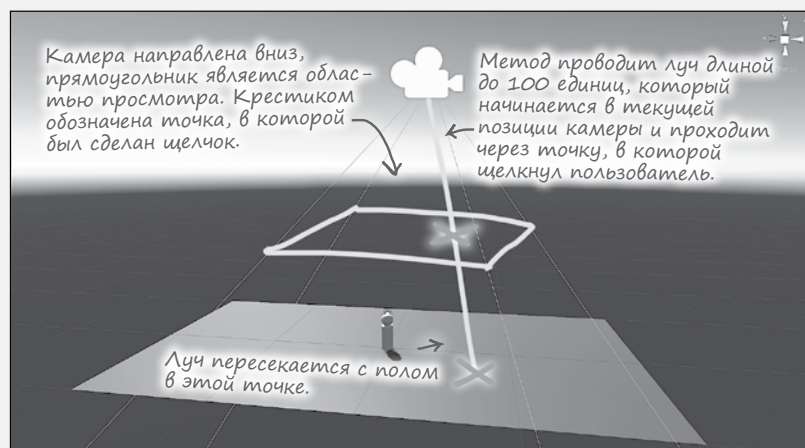


Лабораторный курс Unity №5

Отслеживание лучей

Создавая сцену в Unity, вы строите 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 посмотрим, чем тут поможет C#.

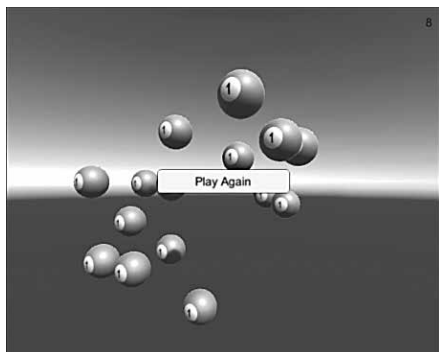
| | |
|---|-----|
| Создание нового проекта Unity и начало создания сцены | 714 |
| Настройка камеры | 715 |
| Создание объекта GameObject для игрока | 716 |
| Знакомство с системой навигации Unity | 717 |
| Установка пакета AI Navigation | 718 |
| Что можно делать с навигацией | 719 |
| Создание сетки NavMesh | 720 |
| Автоматическая навигация в игровой области | 723 |



CAPTAIN AMAZING

СМЕРТЬ ОБЪЕКТА

| Head First C# | |
|----------------|----------|
| Четыре доллара | Глава 11 |



У МЕНЯ... ОСТАЛОСЬ...
 >ОХ!<
 ОДНО... ПОСЛЕДНЕЕ... ДЕЛО...

| | |
|---|-----|
| Жизнь и смерть объекта | 730 |
| Для принудительной сборки мусора используйте класс GC (осторожно!) | 731 |
| Финализатор объекта — последний шанс что-то сделать | 732 |
| Когда ИМЕННО выполняется финализатор? | 733 |
| Финализаторы не могут зависеть от других объектов | 735 |
| Структура похожа на объект... | 739 |
| Значения копируются, ссылки присваиваются | 740 |
| Структуры относятся к типам значений; объекты относятся к ссылочным типам | 741 |
| Стек и куча: подробнее о памяти | 743 |
| Параметры out и возвращение нескольких значений методом | 746 |
| Передача по ссылке с модификатором ref | 747 |
| Необязательные параметры и значения по умолчанию | 748 |
| Ссылка null не указывает ни на какой объект | 749 |
| Ссылочные типы, не допускающие null, помогут избежать NRE | 750 |
| Безопасная работа с типами значений, допускающими null | 753 |
| Оператор объединения с null ?? автоматически проверяет значение на null | 754 |
| Капитан... уже не такой Великолепный | 755 |
| Записи предоставляют эквивалентность объектов автоматически | 757 |
| Не изменяйте записи — копируйте их | 758 |
| Методы расширения добавляют новое поведение в СУЩЕСТВУЮЩИЕ классы | 763 |
| Расширение фундаментального типа: string | 764 |

12

Обработка исключений

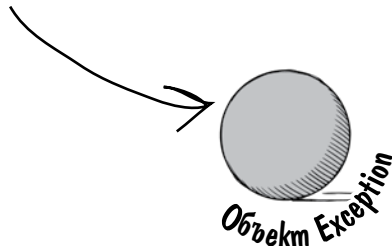
Борьба с огнем надоедает

Программисты не должны уподобляться пожарным.

Представьте: прошло много лет. Вы усердно работали над развитием своих навыков C#, продолжали учиться и совершенствоваться и стали одним из ведущих разработчиков в крупной технологической компании. Но вам до сих пор продолжают звонить с работы по ночам, потому что **программа упала** или **работает не так, как должна работать**. Однако вы хотите проводить свое время за написанием кода, а не за тушением пожаров! Ничто так не выбивает из колеи, как необходимость устранять странные ошибки... но благодаря **обработке исключений C#** 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 | 772 |
| Когда ваша программа выдает исключение, CLR генерирует объект Exception | 776 |
| Все объекты Exception наследуют от System.Exception | 777 |
| Для некоторых файлов вывод дампа невозможен | 780 |
| Что происходит при вызове небезопасного метода? | 781 |
| Обработка исключений с try и catch | 782 |
| Отслеживание передачи управления в try/catch | 783 |
| Перехват всех исключений | 785 |
| Использование исключения, подходящего для конкретной ситуации | 790 |
| Фильтры исключений повышают точность обработки исключений | 794 |
| Наихудший блок catch: универсальный перехват с комментариями | 796 |
| Временные решения допустимы (но только временно) | 797 |
| Использование NuGet для добавления библиотеки ведения журнала в приложение | 799 |
| Добавление поддержки журнала в приложение ExceptionExperiment | 800 |



Интересно, что будет, если щелкнуть здесь...

Если мой метод Process получит некорректные данные, он перестанет работать!

```
public class Data {
    public void
    Process(Data d) {
        if (d.IsBad())
            explode();
    }
}
```

Лабораторный курс Unity №6

Навигация 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 дополним сцену с помощью C#.

| | |
|---|-----|
| Продолжим с того, на чем прервалась последняя лабораторная работа Unity | 804 |
| Добавление платформы в сцену | 805 |
| Изменение настроек предварительного построения | 806 |
| Включение лестницы и наклонной плоскости в NavMesh | 807 |
| Навигация с обходом препятствий | 809 |
| Проявите фантазию! | 810 |



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.