4-е издание

УЧИМ PYTHON, ДЕЛАЯ КРУТЫЕ ИГРЫ

ЭЛ СВЕЙГАРТ

НЕ ИГРАЙ В ИГРЫ – СОЗДАВАЙ ИХ!

no starch press

мировой
компьютерный
бестселлер

AL SWEIGART

INVENT YOUR OWN COMPUTER GAMES WITH PYTHON

DON'T JUST PLAY GAMES – MAKE THEM!

no starch press

ЭЛ СВЕЙГАРТ

УЧИМ PYTHON, ДЕЛАЯ КРУТЫЕ ИГРЫ

НЕ ИГРАЙ В ИГРЫ – СОЗДАВАЙ ИХ!

БОМБОРА™
Москва 2018

УДК 004.43
ББК 32.973.26-018.1
С24

Al Sweigart

INVENT YOUR OWN COMPUTER GAMES WITH PYTHON, 4th EDITION

Copyright © 2017 by Al Sweigart. Title of English language original:
Invent Your Own Computer Games with Python, 4th Edition, ISBN 978-1-59327-795-6,
published by No Starch Press. Russian language edition copyright
© 2018 by EKSMO Publishing House. All rights reserved.

Свейгарт, Эл.

С24 Учим Python, делая крутые игры / Эл Свейгарт ; [пер. с англ. М.А. Райтмана]. — Москва : Эксмо, 2018. — 416 с. — (Мировой компьютерный бестселлер).

ISBN 978-5-699-99572-1

Перед вами — увлекательный самоучитель по языку Python для начинающих. Книга подходит даже читателям с нулевым уровнем. Создавайте собственными руками веселые классические и необычные, продвинутые игры, такие как «Виселица» или «Охотник за сокровищами», — в процессе вы поймете основные принципы программирования и выучите Python играючи!

УДК 004.43
ББК 32.973.26-018.1

ISBN 978-5-699-99572-1

© Райтман М.А., перевод на русский язык, 2017
© Оформление. ООО «Издательство «Эксмо», 2018

ОГЛАВЛЕНИЕ

Об авторе **15**

Введение **16**
Для кого эта книга? 17
О книге 18
Как пользоваться этой книгой 19
Номера строк и отступов 19
Длинные строки кода 20
Загрузка и установка Python 21
Запуск интерактивной среды разработки 22
Поиск справочной документации 23

Глава 1. Интерактивная среда разработки **24**
Немного арифметики 24
Целые числа и числа с плавающей запятой 25
Выражения 26
Вычисление выражений 26
Синтаксические ошибки 27
Сохранение значений в переменных 28
Заключение 33

Глава 2. Написание программ **34**
Строковые значения 34
Конкатенация строк 35
Написание кода в среде разработки 36
Создание программы «Привет, мир!» 37
Сохранение программы 38
Запуск программы 39
Как работает программа «Привет, мир!» 40
Комментарии для программиста 40
Функции: Мини-программы внутри программ 41
Функция `print()` 41
Функция `input()` 41
Выражения в вызовах функций 42
Завершение программы 42
Имена переменных 43
Заключение 44

Глава 3. Игра «Угадай число» **45**
Пример запуска игры «Угадай число» 46
Исходный код игры «Угадай число» 46
Импорт модуля `random` 47
Генерация случайных чисел при помощи функции `random.randint()` 49
Приветствие игрока 51

Инструкции управления потоком ... 51
Использование циклов для повторения кода ... 51
Группировка в блоки ... 52
Создание циклов с инструкцией `for` ... 53
Игрок угадывает число ... 55
Преобразование значений при помощи функций `int()`, `float()` и `str()` ... 55
Логический тип данных ... 57
Операторы сравнения ... 58
Проверка условий и определение истинности/ложности ... 58
Эксперименты с логическими операторами, операторами сравнения и условиями ... 59
Разница между операторами `=` и `==` ... 61
Инструкции `if` ... 61
Выход из цикла до его завершения при помощи инструкции `break` ... 62
Проверка, победил ли игрок ... 62
Проверка, проиграл ли игрок ... 63
Заключение ... 64

Глава 4. Программа-шутник ... 65
Пример запуска программы «Шутки» ... 65
Исходный код программы «Шутки» ... 66
Как работает код ... 66
Экранируемые символы ... 67
Одинарные и двойные кавычки ... 68
Параметр `end` функции `print()` ... 70
Заключение ... 71

Глава 5. Игра «Царство драконов» ... 72
Как играть в «Царство драконов» ... 72
Пример запуска игры «Царство драконов» ... 73
Блок-схема игры «Царство драконов» ... 73
Исходный код игры «Царство драконов» ... 74
Импорт модулей `random` и `time` ... 76
Функции в игре «Царство драконов» ... 76
Инструкции `def` ... 76
Вызов функции ... 77
Где указывать определение функций? ... 77
Многострочный текст ... 78
Выполнение циклов с помощью инструкций `while` ... 79
Логические операторы ... 80
Оператор `and` ... 80
Оператор `or` ... 81
Оператор `not` ... 82
Вычисление логических операций ... 82
Возвращаемые значения функций ... 84
Глобальная и локальная области видимости переменных ... 84
Параметры функции ... 86
Отображение результатов игры ... 87
Определение пещеры с дружелюбным драконом ... 88

Игровой цикл ... 89
Вызов функций в программе ... 90
Запрос «сыграть снова» ... 91
Заключение ... 91

Глава 6. Использование отладчика ... 93
Типы багов ... 93
Отладка ... 95
Запуск отладчика ... 95
Пошаговое выполнение программы с помощью отладчика ... 97
Область глобальных переменных ... 97
Область локальных переменных ... 98
Обычное выполнение и завершение работы ... 98
Навигация по коду ... 98
Поиск багов ... 100
Установка точек останова ... 104
Использование точек останова ... 105
Заключение ... 107

Глава 7. Проектирование игры «Виселица» с помощью блок-схем ... 108
Правила игры «Виселица» ... 108
Пример запуска игры «Виселица» ... 109
ASCII-графика ... 110
Проектирование игры с помощью блок-схем ... 111
Создание блок-схем ... 112
Ветвление в блок-схемах ... 113
Заканчиваем или начинаем игру сначала ... 115
Следующая попытка ... 115
Обратная связь с игроком ... 117
Заключение ... 118

Глава 8. Написание кода игры «Виселица» ... 119
Исходный код игры «Виселица» ... 119
Импорт модуля `random` ... 123
Константы ... 123
Списки ... 124
Доступ к элементам по их индексам ... 125
Индекс за пределами диапазона и ошибка `IndexError` ... 126
Присваивание индексов элементам ... 126
Конкатенация списков ... 126
Оператор `in` ... 127
Вызов методов ... 127
Методы списков `reverse()` и `append()` ... 128
Строковый метод `split()` ... 128
Получение секретного слова из списка ... 129
Отображение игрового поля для игрока ... 130
Функции `list()` и `range()` ... 131
Срезы списков и строк ... 132
Вывод секретного слова с пробелами ... 134

Получение предположений игрока ... 136
Строковые методы `lower()` и `upper()` ... 136
Завершение цикла `while` ... 138
Инструкции `elif` ... 138
Проверка допустимости предположения игрока ... 139
Предложение игроку сыграть заново ... 140
Обзор функций игры ... 141
Игровой цикл ... 141
Вызов функции `displayBoard()` ... 142
Ввод игроком угадываемой буквы ... 142
Проверка наличия буквы в секретном слове ... 143
Проверка — не победил ли игрок ... 143
Обработка ошибочных предположений ... 144
Проверка — не проиграл ли игрок ... 144
Завершение или перезагрузка игры ... 145
Заключение ... 146

Глава 9. Доработка игры «Виселица» ... 147
Увеличение числа угадываний ... 147
Словари ... 148
Определение размера словаря с помощью функции `len()` ... 149
Различия между списком и словарем ... 150
Методы словаря `keys()` и `values()` ... 151
Использование словаря слов в игре «Виселица» ... 152
Случайный выбор из списка ... 152
Удаление элементов списка ... 154
Множественное присваивание ... 156
Выбор игроком категории слов ... 157
Заключение ... 158

Глава 10. Игра «Крестики-нолики» ... 159
Пример запуска игры «Крестики-нолики» ... 160
Исходный код игры «Крестики-нолики» ... 161
Проектирование программы ... 166
Данные для прорисовки игрового поля ... 166
Стратегия игры ИИ ... 167
Импорт модуля `random` ... 168
Вывод игрового поля на экран ... 169
Предоставление игроку выбора между «X» или «O» ... 170
Выбор — кто будет ходить первым ... 171
Размещение меток на игровом поле ... 171
Ссылки на список ... 172
Использование ссылок на списки в функции `makeMove()` ... 175
Проверка — не победил ли игрок ... 175
Дублирование данных игрового поля ... 178
Проверка — свободна ли клетка игрового поля ... 178
Разрешение игроку сделать ход ... 179
Вычисление по короткой схеме ... 180

Выбор хода из списка ... 182
Значение None ... 183
Создание искусственного интеллекта ... 184
Проверка — сможет ли компьютер победить, сделав ход ... 185
Проверка — сможет ли игрок победить, сделав ход ... 185
Проверка угловых, центральной и боковых клеток (в порядке очереди) ... 186
Проверка — заполнено ли поле ... 187
Игровой цикл ... 187
Выбор буквы игрока и того, кто будет ходить первым ... 188
Переменная turn в значении 'Человек' ... 188
Переменная turn в значении 'Компьютер' ... 190
Предложение игроку сыграть заново ... 190
Заключение ... 191

Глава 11. Дедуктивная игра «Холодно-горячо» ... 192
Пример запуска игры «Холодно-горячо» ... 193
Исходный код игры «Холодно-горячо» ... 193
Блок-схема игры «Холодно-горячо» ... 195
Импорт модуля random и определение функции getSecretNum() ... 196
Перетасовка уникального набора цифр ... 197
Изменение порядка элементов списка с помощью функции random.shuffle() ... 197
Получение секретного числа из перетасованных цифр ... 198
Расширенные операторы присваивания ... 198
Подсчет выдаваемых подсказок ... 200
Метод списка sort() ... 201
Строковый метод join() ... 202
Проверка на содержание в строке только чисел ... 203
Начало игры ... 203
Интерполяция строк ... 204
Игровой цикл ... 205
Получение предположения игрока ... 206
Получение подсказок в зависимости от предположения игрока ... 206
Проверка победы или поражения игрока ... 207
Предложение сыграть снова ... 207
Заключение ... 208

Глава 12. Декартова система координат ... 209
Сетки и декартовы координаты ... 209
Отрицательные числа ... 211
Система координат компьютерного экрана ... 213
Математические хитрости ... 213
Хитрость 1: минус «съедает» плюс слева от себя ... 214
Хитрость 2: два минуса в сумме дают плюс ... 214
Хитрость 3: два слагаемых числа можно переставлять местами ... 214
Абсолютные величины и функция abs() ... 215
Заключение ... 216

Глава 13. Игра «Охотник за сокровищами» ... 217
Пример запуска игры «Охотник за сокровищами» ... 218
Исходный код игры «Охотник за сокровищами» ... 222
Проектирование программы ... 227
Импорт модулей `random`, `sys` и `math` ... 227
Создание поля для новой игры ... 228
Генерация игрового поля ... 229
Изображение координат *x* вдоль верхней части поля ... 230
Рисование океана ... 231
Вывод ряда в океане ... 232
Изображение координат *x* вдоль нижней части игрового поля ... 233
Создание случайных сундуков с сокровищами ... 233
Определение допустимости хода ... 234
Отражение хода на игровом поле ... 234
Поиск ближайшего сундука с сокровищами ... 235
Удаление значений с помощью метода списка `remove()` ... 238
Получение хода игрока ... 239
Вывод игроку инструкций по игре ... 241
Игровой цикл ... 242
Демонстрация игроку статуса игры ... 243
Обработка хода игрока ... 243
Нахождение затонувшего сундука с сокровищами ... 244
Проверка победы игрока ... 245
Проверка проигрыша игрока ... 245
Завершение работы программы с помощью функции `sys.exit()` ... 246
Заключение ... 246

Глава 14. Шифр Цезаря ... 248
Криптография и шифрование ... 248
Как работает шифр Цезаря ... 249
Пример запуска программы «Шифр Цезаря» ... 251
Исходный код программы «Шифр Цезаря» ... 252
Установление максимальной длины ключа ... 253
Выбор между шифрованием и расшифровыванием сообщения ... 254
Получение сообщения от игрока ... 254
Получение ключа от игрока ... 255
Шифрование/расшифровывание сообщения ... 255
Нахождение переданных строк с помощью строчного метода `find()` ... 256
Шифрование/расшифровка каждой буквы ... 257
Запуск программы ... 258
Полный перебор ... 259
Добавление режима полного перебора ... 260
Заключение ... 261

Глава 15. Игра «Реверси» ... 263
Как играть в «Реверси» ... 263
Пример запуска игры «Реверси» ... 266
Исходный код игры «Реверси» ... 268

Импорт модулей и создание констант ... 275
Структура данных игрового поля ... 275
Отображение на экране структуры данных игрового поля ... 276
Создание структуры данных нового игрового поля ... 277
Проверка допустимости хода ... 278
Проверка каждого из восьми направлений ... 279
Определение наличия фишек, которые можно перевернуть ... 280
Проверка допустимости координат ... 282
Получение списка со всеми допустимыми ходами ... 282
Вызов функции `bool()` ... 283
Получение игрового счета ... 284
Получение сделанного игроком выбора фишки ... 285
Определение первого игрока ... 285
Помещение фишки на поле ... 286
Создание копии структуры данных игрового поля ... 286
Определение того, находится ли клетка в углу ... 287
Получение хода игрока ... 287
Получение хода компьютера ... 290
Разработка стратегии с угловыми ходами ... 290
Получения списка самых результативных ходов ... 291
Вывод игрового счета на экран ... 292
Начало игры ... 292
Проверка на попадание в тупик ... 293
Выполнение хода игроком ... 294
Выполнение хода компьютером ... 295
Игровой цикл ... 296
Предложение сыграть снова ... 297
Заключение ... 298

Глава 16. Искусственный интеллект игры «Реверси» ... 299
Компьютер против компьютера ... 300
Пример запуска модели 1 ... 300
Исходный код модели 1 ... 301
Удаление приглашений для игрока и добавление игрока-компьютера ... 303
Компьютер против компьютера — несколько партий ... 304
Пример запуска модели 2 ... 304
Исходный код модели 2 ... 305
Отслеживание результатов партий ... 306
Отключение вызовов функции `print()` ... 307
Оценка искусственных интеллектов в процентном соотношении ... 307
Деление приводит к получению числа с плавающей запятой ... 308
Функция `round()` ... 309
Сравнение различных алгоритмов ИИ ... 309
Исходный код модели 3 ... 310
Принципы работы искусственных интеллектов в модели 3 ... 312
ИИ лучшего углового хода ... 312
ИИ худшего хода ... 312

ИИ случайного хода ... 313
Проверка на граничные ходы ... 314
ИИ лучшего углового-граничного хода ... 314
Сравнение искусственных интеллектов ... 315
ИИ худшего хода против ИИ лучшего углового хода ... 315
ИИ случайного хода против ИИ лучшего углового хода ... 316
ИИ лучшего углового-граничного хода против ИИ лучшего углового хода ... 317
Заключение ... 318

Глава 17. Создание графики ... 319
Установка pygame ... 320
Привет, pygame! ... 320
Пример запуска `pygame`-программы «Привет, мир!» ... 321
Исходный код `pygame`-программы «Привет, мир!» ... 322
Импорт модуля pygame ... 323
Инициализация `pygame` ... 324
Настройка окна `pygame` ... 324
Кортежи ... 325
Объекты поверхности ... 326
Работа с цветом ... 326
Вывод текста в окне `pygame` ... 327
Использование шрифтов для оформления текста ... 327
Рендеринг объекта `Font` ... 328
Настройка местоположения текста с помощью атрибутов `Rect` ... 329
Заливка цветом объекта `Surface` ... 331
Функции рисования pygame ... 332
Рисование многоугольника ... 332
Рисование линии ... 333
Рисование круга ... 333
Рисование эллипса ... 334
Рисование прямоугольника ... 334
Окрашивание пикселей ... 335
Метод `blit()` для объектов `Surface` ... 336
Вывод объекта `Surface` на экран ... 336
События и игровой цикл ... 336
Получение объектов `Event` ... 337
Выход из программы ... 338
Заключение ... 338

Глава 18. Анимированная графика ... 339
Пример запуска игры программы ... 339
Исходный код программы ... 339
Перемещение и контроль отскока блоков ... 342
Создание констант ... 343
Константы для направлений ... 344
Константы для цвета ... 345
Создание структуры данных блока ... 345

Игровой цикл ... 346
Обработка решения игрока завершить игру ... 346
Перемещение каждого блока ... 347
Управление отскакиванием блока ... 349
Отображение в окне блоков в новых положениях ... 350
Отображение окна на экране ... 350
Заключение ... 351

Глава 19. Обнаружение столкновений ... 352
Пример работы программы ... 353
Исходный код программы ... 353
Импорт модулей ... 356
Использование объекта `Clock` для управления скоростью работы программы ... 357
Настройка окна и структур данных ... 357
Создание переменных для отслеживания перемещения ... 359
Обработка событий ... 359
Обработка события `KEYDOWN` ... 361
Обработка события `KEYUP` ... 363
Телепортация игрока ... 364
Добавление новых блоков «еды» ... 364
Перемещение игрока по окну ... 365
Отображение блока игрока в окне ... 366
Проверка на столкновения ... 366
Отображение блоков «еды» ... 367
Заключение ... 368

Глава 20. Использование звуков и изображений ... 369
Добавление изображений с помощью спрайтов ... 369
Графические и звуковые файлы ... 370
Пример запуска игры ... 371
Исходный код программы ... 371
Настройка окна и создание структуры данных ... 375
Добавление спрайта ... 375
Изменение размера спрайта ... 376
Установка музыки и звуков ... 376
Добавление аудиофайлов ... 376
Включение/отключение звука ... 377
Отображение спрайта игрока в окне ... 378
Проверка на столкновения ... 378
Отображение спрайтов вишен в окне ... 379
Заключение ... 380

Глава 21. Игра «Ловкач» с графикой и звуком ... 381
Обзор основных типов данных `pygame` ... 381
Пример запуска игры «Ловкач» ... 383
Исходный код игры «Ловкач» ... 383
Импорт модулей ... 388
Создание констант ... 389

Определение функций 390
Завершение игры и добавление паузы 390
Отслеживание столкновений со злодеями 391
Отображение текста в окне 392
Инициализация pygame и настройка окна 393
Установка шрифтов, изображений и звуков 394
Отображение начального экрана 395
Начало игры 396
Игровой цикл 398
Обработка событий клавиатуры 398
Обработка событий мыши 400
Добавление новых злодеев 401
Перемещение спрайтов игрока и злодеев 402
Реализация чит-кодов 404
Удаление спрайтов злодеев 404
Отображение окна 405
Отображение очков игрока 405
Отображение спрайтов игрока и злодеев 406
Проверка на столкновения 406
Экран окончания игры 407
Изменение игры «Ловкач» 408
Заключение 409

Предметный указатель 410

ОБ АВТОРЕ

Эл Свейгарт — разработчик программного обеспечения, автор технических книг и человек, у которого всегда при себе полотенце*. Он написал несколько книг по программированию для начинающих, в том числе «Программирование для детей. Делай игры и учи язык Scratch!». Его книги на языке оригинала можно бесплатно прочитать на сайте www.inventwithpython.com.

* В оригинале: «…who really knows where his towel is» — отсылка к роману «Автостопом по галактике» (1979 г.) Дугласа Адамса, британского писателя, оказавшего влияние на культовое для программистов на Python комедийное шоу «Летающий цирк Монти Пайтона» (Monty Python's Flying Circus). — *Примеч. ред.*

Посвящается Каро

ВВЕДЕНИЕ

Когда я в детстве впервые запустил компьютерную игру, меня моментально затянуло. Но я хотел не только играть в игры, я хотел их создавать. Я нашел книгу, похожую на эту, которая научила меня писать свои первые программы и игры. Это было просто и весело. Первые игры, которые я делал, были простыми и похожими на показанные в этой книге. Они не были такими вычурными, как бестселлеры Nintendo, которые мне покупали родители, но это были игры, которые я сделал сам.

Сейчас, став совершеннолетним, я все еще получаю удовольствие от программирования и мне за это еще и платят. Но даже если вы не хотите связывать с этим жизнь, программирование — полезный и веселый навык. Он тренирует ваш мозг мыслить логически, строить планы и пересматривать свои решения каждый раз, как вы обнаружите ошибку в коде.

Многие книги, посвященные программированию для начинающих, попадают под две категории. Первая категория включает книги, которые обучают не столь программированию, сколько «программному обеспечению для создания игр» или языкам, которые упрощаются настолько, что предмет изучения уже и нельзя назвать программированием. Вторая категория состоит из книг, которые обучают программированию подобно математическому учебнику — со всеми принципами и понятиями и парочкой реальных приложений в пример читателю. Эта книга прибегает к иному подходу и учит вас, как программировать с помощью создания видеоигр. Я показываю исходный код для игр и напротив объясняю принцип программирования из примера. Такой подход был ключевым для меня, когда я учился программированию.

Чем больше я узнавал, как работают программы, созданные другими людьми, тем больше идей ко мне приходило касательно моих собственных.

Все, что вам нужно — компьютер, бесплатное программное обеспечение, которое называется интерпретатор Python и эта книга. Как только вы узнаете, как создавать игры из этой книги, вы сможете разрабатывать игры сами.

Компьютеры — поразительные механизмы и обучение программированию не такое неподъемное, как сначала кажется. Компьютерная *программа* — это набор инструкций, которые компьютер может понять, точно так же, как книга с историями — это куча предложений, которые может понять читатель. Чтобы создать инструкцию, вы пишете программу на языке, который компьютер понимает. Эта книга научит вас языку программирования Python. Существует множество других языков программирования, которые вы можете освоить, например, BASIC, Java, JavaScript, PHP и C++.

В детстве я выучил BASIC, но более новые языки программирования типа Python учатся гораздо легче. Python также используется профессиональными программистами в работе и любителями в качестве хобби. Вдобавок он абсолютно бесплатный для установки и использования — вам всего-то нужен Интернет, чтобы скачать его.

Поскольку видеоигры — ничто иное как компьютерные программы, они также состоят из инструкций. Игры, которые вы создадите благодаря этой книге, будут проще сравнительно с играми для Xbox, Paystation или Nintendo. В этих играх не будет невероятной графики, потому что они рассчитаны на то, чтобы обучить вас основам кодинга. Они преднамеренно просты, поэтому вы можете сосредоточиться на обучении программированию. Игры не должны быть сложными, чтобы быть интересными!

Для кого эта книга?

Программирование — это не сложно, сложнее найти материалы, которые обучат вас делать интересные вещи с его помощью. Другие книги предоставляют слишком много того, что начинающему кодеру не понадобится. Эта книга научит вас программировать собственные игры; вы получите полезный опыт и интересные видеоигры, которые сможете продемонстрировать.

Эта книга для:

- абсолютных новичков, которые хотят обучиться программированию, даже не имея при этом предыдущего опыта;
- детей и подростков, которые хотят научиться программированию, создавая игры;
- взрослых и учителей, которые хотят освоить программирование;
- кого угодно, от мала до велика, кто хочет научиться программировать, изучая профессиональный язык программирования.

О книге

В большинстве разделов этой книги вводится и объясняется по одному новому игровому проекту. В некоторых главах рассматриваются дополнительные полезные темы, такие как отладка. Новые концепции программирования по мере применения их в играх и главы предназначены для чтения по порядку.

Вот краткое изложение того, что вы найдете в каждой главе:

- Глава 1 «Интерактивная среда разработки» пошагово объясняет, как интерактивная среда разработки Python может использоваться для экспериментов с кодом.
- Глава 2 «Написание программ» объясняет, как писать полноценные программы в редакторе файлов Python.
- В главе 3 «Игра “Угадай число”» вы запрограммируете первую игру в этой книге, в которой попросите игрока угадать секретное число, а затем дадите подсказки относительно того, насколько его предположение близко или далеко от правильного ответа.
- В главе 4 «Программа-шутник» вы напишете простую программу, которая рассказывает пользователю несколько шуток.
- В главе 5 «Игра “Царство драконов”» вы запрограммируете игру для угадывания, в которой игрок должен выбрать между двумя пещерами: в одной живет дружелюбный дракон, а в другой — голодный.
- Глава 6 «Использование отладчика» описывает способы использования отладчика для устранения неполадок в коде.
- Глава 7 «Проектирование игры “Виселица” с помощью блок-схем» объясняет, как блок-схемы помогают в планировании более длинных программ, таких как игра «Виселица».
- В главе 8 «Написание кода игры “Виселица”» вы напишете игру «Виселица», опираясь на блок-схему из главы 7.
- Глава 9 «Доработка игры “Виселица”» расширяет игру «Виселица» новыми функциями, используя базу данных словаря Python.
- В главе 10 «Игра “Крестики-нолики”» вы узнаете, как писать игру типа «человек против компьютера», использующую искусственный интеллект.
- В главе 11 «Дедуктивная игра “Холодно-горячо”» вы узнаете, как сделать дедуктивную игру, в которой игрок должен угадать секретные числа на основе подсказок.
- Глава 12 «Декартова система координат» объясняет декартову систему координат, которую вы позже будете использовать в играх.
- В главе 13 «Игра “Охотник за сокровищами”» вы узнаете, как написать игру, в которой игрок прочесывает океан в поисках потерянных сундуков с сокровищами.

- В главе 14 «Шифр Цезаря» вы создадите простую программу шифрования, которая позволяет создавать и декодировать секретные сообщения.
- В главе 15 «Игра “Реверси”» вы запрограммируете передовую игру типа «Реверси» «человек против компьютера», в которой будет почти непревзойденный противник искусственного интеллекта.
- Глава 16 «Искусственный интеллект игры “Реверси”» расширяет игру из главы 15, реализуя несколько искусственных интеллектов, которые конкурируют в играх типа «человек против компьютера».
- Глава 17 «Создание графики» представляет модуль `pygame` и показывает, как использовать его для рисования двухмерной графики.
- Глава 18 «Анимированная графика» показывает, как сделать из графики анимацию с помощью модуля `pygame`.
- В главе 19 «Обнаружение столкновений» вы узнаете, как определять, когда объекты сталкиваются друг с другом в двухмерных играх.
- В главе 20 «Использование звуков и изображений» вы улучшите свои простые игры на основе модуля `pygame`, добавив звуки и изображения.
- Глава 21 «Игра “Ловкач” с графикой и звуком» объединяет концепции в главах с 17 по 20, чтобы сделать анимированную игру под названием «Ловкач».

Как пользоваться этой книгой

Большинство глав в этой книге начинаются с примера запуска программы, разбор которой затем описан в главе. Этот пример показывает, как выглядит вывод программы после ее запуска. Полужирным шрифтом выделены данные, которые вводит пользователь. Я рекомендую вам вводить код для каждой программы в редакторе файлов самостоятельно, а не загружать или копировать и вставлять его. Если вы потратите время на написание кода, то усвоите информацию лучше.

Номера строк и отступов

При вводе исходного кода из этой книги *не* вводите номера строк в начале каждой строки. Например, если вы видите следующую строку кода, вам не нужно вводить `9.` с левой стороны или ставить пробел сразу после него.

```
9. number = random.randint(1, 20)
```

Вы вводите только это:

```
number = random.randint(1, 20)
```

Эти числа приведены только для того, чтобы в книге автор мог ссылаться на конкретные строки в программе. Они не являются реальной частью кода. Помимо номеров строк, вводите код точно так, как он показан в этой книге. Обратите внимание, что некоторые строки кода имеют отступы в четыре, восемь или более пробелов. Пробелы в начале строки меняют то, как Python интерпретирует инструкции, поэтому их наличие очень важно.

Давайте посмотрим на пример.

```
while guesses < 10:
••••if number == 42:
••••••••print('Привет')
```

Отступы здесь отмечены черными кружками (•), чтобы вы могли их видеть.

В первой строке отступов нет, вторая строка содержит четыре пробела, а в третьей строке — отступ в восемь пробелов. Хоть в примерах в этой книге нет черных кружков для обозначения пробелов, каждый символ в среде разработки имеет одинаковую ширину, поэтому вы можете подсчитать количество пробелов, посчитав количество символов в строке выше или ниже.

Длинные строки кода

Некоторые инструкции кода слишком длинные, чтобы уместиться в одну строку в книге, и потому переносятся на следующую. Но строка на вашем компьютере будет переноситься иначе (в зависимости от разрешения монитора), поэтому вводите все на одной строке, не нажимая клавишу **Enter**. Вы можете узнать, когда начинается новая инструкция, просматривая номера строк слева.

У следующего примера только две инструкции:

```
1. print('Это первая инструкция!xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxx')
2. print('Это вторая инструкция, а не третья.')
```

Первая инструкция переносится на вторую строку на странице, но вторая строка не имеет порядкового номера, поэтому она относится к первой строке кода.

Загрузка и установка Python

Вам нужно будет установить на ваш компьютер программное обеспечение, которое называется *интерпретатором* Python. Программа понимает инструкции, которые вы пишете на Python. С этого момента я буду ссылаться на программное обеспечение интерпретации Python просто как на *Python*.

В этом разделе я покажу, как скачать и установить Python 3 — в частности, Python 3.6.2 для Windows, macOS или Ubuntu. Это актуальная версия на момент написания книги. На момент чтения книги версия может быть иной, следовательно, ваши шаги могут немного отличаться. Также важно знать, что есть значительная разница между Python 2 и Python 3. Программы в этой книге основаны на версии Python 3, и вы получите сообщение об ошибке, если попытаетесь запустить их в среде Python 2.

Это настолько важно, что автор добавил мультяшного пингвина, чтобы напомнить вам об этом.

Для операционной системы Windows скачайте файл по ссылке **Windows x86 web-based installer** на странице **www.python.org/downloads/release/python-362/** и дважды щелкните по нему мышью. Возможно, вам потребуется запустить данный файл от имени администратора.

Следуйте инструкциям, которые установщик отображает на экране, чтобы инсталлировать Python, как указано ниже.

1. Нажмите кнопку **Customize Installation** (Настроить установку).
2. Нажмите кнопку **Next** (Далее).
3. Установите флажок **Install for All Users** (Установить для всех пользователей) и укажите путь установки **C:\Python36**. Нажмите кнопку **Install** (Установить).
4. Нажмите кнопку **Close** (Закрыть).

В операционной системе macOS скачайте файл по ссылке **Mac OS X 64-bit/32-bit installer** на странице **www.python.org/downloads/release/python-362/** и дважды щелкните по нему мышью.

Следуйте инструкциям установщика на экране, чтобы установить Python, как указано ниже.

1. Если отобразилось предупреждение, что файл *Python.mpkg* не может быть открыт, потому что принадлежит неустановленному разработчику, то, нажав и удерживая клавишу **Ctrl**, щелкните по файлу *Python.mpkg* правой кнопкой мыши, а затем в появившемся меню выберите команду Открыть (**Open**). Возможно, вам потребуется ввести пароль администратора для вашего компьютера.

2. Нажмите кнопку **Continue** (Продолжить) в области приветствия и щелкните мышью по кнопке **Agree** (Согласен), чтобы принять условия лицензии.
3. Выберите диск **Macintosh HD** (может называться иначе на вашем компьютере) и нажмите кнопку **Install** (Установить).

Если вы используете операционную систему Ubuntu, вы можете установить Python из центра приложений Ubuntu, выполнив следующие шаги:

1. Откройте центр приложений Ubuntu.
2. Введите *Python* в поле поиска в правом верхнем углу окна.
3. Выберите пункт **IDLE (Python 3.6.2 GUI 64 bit)**.
4. Нажмите кнопку **Install** (Установить). Возможно, вам потребуется ввести пароль администратора для вашего компьютера.

Если вышеуказанные шаги не работают, вы можете найти альтернативные инструкции по установке среды разработки Python на странице **www.nostarch.com/inventwithpython/**.

Запуск интерактивной среды разработки

Аббревиатура IDLE означает **I**nteractive **D**eve**L**opment **E**nvironment — интерактивная среда разработки. IDLE — это нечто вроде текстового процессора для написания кода программ Python.

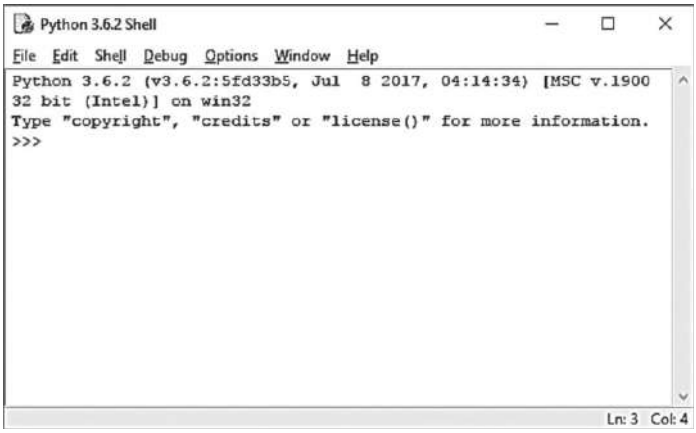


Рис. 1. IDLE — интерактивная среда разработки

Запуск среды разработки отличается в каждой операционной системе:

- В операционной системе Windows откройте меню **Пуск** (Start), наберите *IDLE* и выберите пункт **IDLE (Python 3.6 32-bit)**.
- В macOS запустите приложение Finder и щелкните мышью по пункту **Приложения** (Applications). Дважды щелкните мышью по папке *Python 3.x*, а затем дважды щелкните мышью по значку **IDLE**.

- В операционной системе Ubuntu и других дистрибутивах Linux откройте окно командной строки и введите `idle3`. Вы также можете нажать кнопку **Приложения** (Applications) в верхней части экрана. Затем последовательно выберите пункты **Программирование** (Programming) и **IDLE 3**.

При запуске IDLE откроется окно интерактивной среды разработки, как показано на рис. 1.

Приглашение >>> в окне интерактивной оболочки позволяет вам вводить отдельные инструкции, которые Python будет выполнять сразу. После выполнения инструкции вы получите новое приглашение >>> для ввода следующей.

Поиск справочной документации

Вы можете найти файлы с исходным кодом примеров и другие ресурсы для этой книги на странице **eksmo.ru/files/python_games_sweigart.zip**. Если вы хотите задать вопросы, связанные с этой книгой, вы можете отправить свои вопросы на английском языке автору по адресу **al@inventwithpython.com**.

Прежде чем задавать какие-либо вопросы, убедитесь, что вы делаете следующее:

- Если вы ввели код программы из этой книги, но получили ошибку, перед тем, как задать свой вопрос, проверьте опечатки с помощью инструмента онлайн-анализа на сайте **inventwithpython.com/diff/**. Скопируйте и вставьте свой код в текстовое поле, чтобы найти различия между кодом из книги и вашим.
- Погуглите, чтобы узнать, спрашивал ли кто-то еще (и получал ли ответ) на ваш вопрос. Имейте в виду, что чем понятнее вы будете формулировать свои вопросы, тем быстрее другие смогут вам помочь.

Когда задаете вопросы, делайте следующее:

- Расскажите, что вы пытались сделать, когда получили ошибку. Это позволит быстрее определить, где вы сбились с пути.
- Скопируйте и вставьте полное сообщение об ошибке и код.
- Укажите название используемой операционной системы и ее версию.
- Расскажите, что вы уже пытались сделать, чтобы решить проблему. Это говорит о том, что вы уже проделали какую-то работу, чтобы попытаться разобраться самостоятельно.
- Будьте вежливы. Не требуйте скорой помощи и не давите на собеседников, чтобы те среагировали моментально.

Теперь, когда вы знаете, как обратиться за помощью, вы научитесь программировать свои собственные компьютерные игры в кратчайшие сроки!

1

ИНТЕРАКТИВНАЯ СРЕДА РАЗРАБОТКИ

Прежде чем приступить к разработке игр, вам нужно изучить несколько базовых концепций программирования. Эту главу вы начнете с того, что научитесь использовать интерактивную среду разработки Python и выполнять базовые арифметические задачи.

В ЭТОЙ ГЛАВЕ РАССМАТРИВАЮТСЯ СЛЕДУЮЩИЕ ТЕМЫ:

- Операторы
- Целые числа и числа с плавающей запятой
- Значения
- Выражения
- Синтаксические ошибки в коде
- Сохранение значений в переменных

Немного арифметики

Запустите IDLE, выполнив указания из раздела «Запуск интерактивной среды разработки» во введении. На первых порах вы будете использовать Python для решения простых арифметических задач. IDLE может быть использована как калькулятор. Введите `2 + 2` в интерактивной оболочке после символов приглашения, **>>>**, и нажмите клавишу **Enter**. (На некоторых клавиатурах эта клавиша обозначена как **Return**.) На рис. 1.1 показано, как

эта арифметическая задача выглядит в интерактивной оболочке — обратите внимание, что получен ответ в виде числа 4.

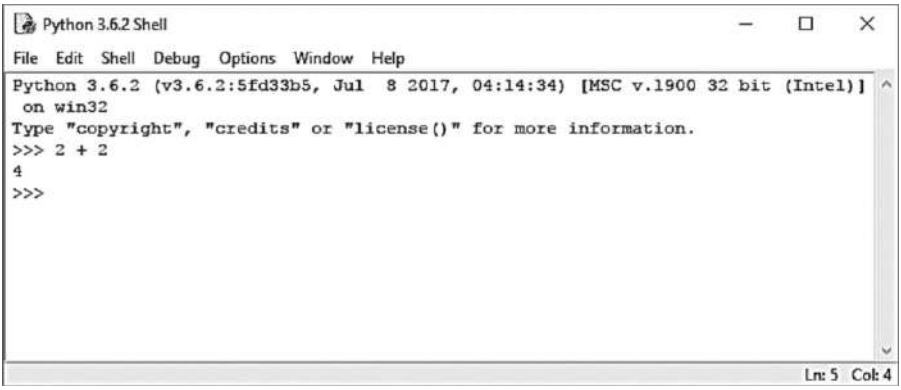


Рис. 1.1. Ввод `2 + 2` в интерактивной оболочке

Эта арифметическая задача — простая инструкция программирования. Символ «плюс» (+) указывает компьютеру сложить 2 и 2. Компьютер выполняет операцию и отвечает числом 4 в следующей строке. Таблица 1.1 содержит доступные в Python арифметические операторы.

Таблица 1.1. Арифметические операторы

Оператор	Операция
+	Прибавить
-	Вычесть
*	Умножить
/	Разделить

Символ минус (-) вычитает числа, звездочка (*) умножает, а косая черта (/) делит. При использовании таким образом, знаки +, -, * и / называются *операторами*. Операторы сообщают Python, какую операцию необходимо произвести над числами.

Целые числа и числа с плавающей запятой

Целые числа (на английском `int`) представляют собой числа, такие как 4, 99 или 0. *Числа с плавающей запятой* (на английском `float`) представляют собой дроби или числа с десятичными точками, такие как 3.5, 42.1 и 5.0. Для Python число 5 — целое, а 5.0 — число с плавающей запятой.

Эти числа называют *значениями*. Позже вы узнаете о других видах значений помимо чисел. В арифметической задаче, введенной в интерактивной оболочке, 2 и 2 — это целые (или *целочисленные*) значения.

Выражения

Арифметическая задача `2 + 2` — пример *выражения*. Как демонстрирует рис. 1.2, выражения состоят из значений (чисел), связанных между собой с помощью операторов (арифметических символов), что создает новое значение, которое может быть использовано в коде. Компьютер способен вычислить значения миллиардов выражений в секунду.

Рис. 1.2. Выражение, состоящее из значений и операторов

Введите следующие выражения в интерактивной оболочке, нажимая клавишу **Enter** после каждой строки.

```
>>> 2+2+2+2+2
10
>>> 8*6
48
>>> 10-5+6
11
>>> 2 +  2
4
```

Все эти выражения выглядят как обычные арифметические уравнения, но обратите внимание на пробелы в примере `2 +  2`. В Python вы можете поставить любое количество пробелов между значениями и операторами. Тем не менее вы всегда должны начинать инструкцию в начале строки без пробелов при вводе их в интерактивной оболочке.

Вычисление выражений

Когда среда вычисляет выражение `10 + 5` и возвращает значение `15`, это значит, что он его *вычислил*. Вычисление выражения сводит его к единому значению, так же как и решение арифметической задачи сводит ее к единственному числу — ответу. Например, два выражения `10 + 5` и `10 + 3 + 2` вычисляются с результатом `15`.

Когда Python вычисляет выражение, он придерживается того же порядка арифметических операций, какому вас научили в школе.

Правила следующие:

- Сначала вычисляются части выражения внутри круглых скобок.
- Умножение и деление выполняются перед сложением и вычитанием.
- Порядок действий слева направо.

Выражение `1 + 2 * 3 + 4` равно `11`, а не `13`, потому что сначала выполняется умножение `2 * 3`. Если бы выражение выглядело как `(1 + 2) * (3 + 4)`, оно бы равнялось `21`, потому что части `(1 + 2)` и `(3 + 4)` находятся внутри скобок и вычисляются перед умножением.

Выражения могут быть любого размера, и Python вычислит значение каждого из них. Даже одиночные цифры — это выражения. Например, выражение `15` является выражением со значением `15`. Выражение `8 * 3 / 2 + 2 + 7 - 9` равняется значению `12.0` после следующих вычислений:

```
8 * 3 / 2 + 2 + 7 - 9
  24 / 2 + 2 + 7 - 9
    12.0 + 2 + 7 - 9
      14.0 + 7 - 9
        21.0 - 9
            12.0
```

Несмотря на то что компьютер выполняет все эти шаги, вы не видите их в интерактивной оболочке . Интерактивная среда разработки показывает вам только результат.

```
>>> 8 * 3 / 2 + 2 + 7 - 9
12.0
```

Обратите внимание, что выражения с оператором (/) деления Python в итоге приводит к числу с плавающей запятой; Например, `24 / 2` вычисляется как `12.0`. Математические операции хотя бы с одним значением с плавающей запятой так же записываются и в дальнейшем, поэтому `12.0 + 2` вычисляется как `14.0`.

Синтаксические ошибки

Если в интерактивной оболочке вы напишете `5 +` и нажмете клавишу **Enter**, то получите следующее сообщение об ошибке:

```
>>> 5 +
SyntaxError: invalid syntax
```

Эта ошибка произошла потому, что `5 +` не является выражением. Выражения состоят из значений, соединенных операторами. В данном случае оператор сложения `+` подразумевает наличие значений с *двух* сторон. Сообщение об ошибке появляется, когда ожидаемое значение отсутствует.

«Синтаксическая ошибка» означает, что Python не понимает инструкцию, потому что вы задали ее неверно. При написании программ важно не только давать инструкции компьютеру, но и знать, как делать это правильно.

Впрочем, не переживайте из-за ошибок в коде. Ошибки не нанесут вреда вашему компьютеру. Просто правильно перепишите инструкцию в интерактивной оболочке после следующего приглашения `>>>`.

Сохранение значений в переменных

Вы можете использовать полученные значения выражений, сохраняя их в *переменных*. Для простоты можете представить, что переменная — это такая «коробка», в которой вы можете хранить значения. Для сохранения значения выражения в переменной используйте *инструкцию присваивания*. Введите имя переменной, затем знак равенства (=), который называется *оператором присваивания*, и затем значение. Например, введите в интерактивной оболочке следующую команду:

```
>>> spam = 15
>>>
```

В «коробке» переменной `spam` теперь хранится значение `15`, как показано на рис. 1.3.

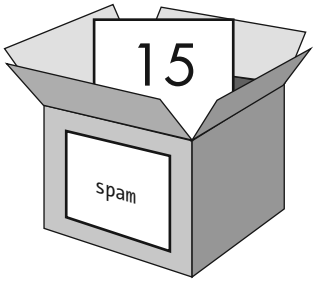


Рис. 1.3. Переменные подобны «коробкам», которые могут содержать значения

Когда вы нажмете клавишу **Enter**, вы ничего не получите в ответ. Вы уже знаете, что если Python не выдает сообщение об ошибке, то он успешно выполнил введенную инструкцию. Появится приглашение >>> для ввода следующей инструкции.

В отличие от выражений, которые мы рассмотрели выше, инструкция присваивания является предписанием. *Предписания* не производят численных вычислений и, таким образом, не возвращают значения. Вот почему на следующей строке IDLE после `spam = 15` нет значения. Если вы путаетесь, какие инструкции являются выражениями, а какие предписаниями, помните, что для каждого выражения Python вычисляет значение. Таким образом, инструкции других типов являются предписаниями.

Переменные хранят значения, а не выражения. Например, рассмотрим выражения `spam = 10 + 5` и `spam = 10 + 7 - 2`. Они оба равны `15`. Конечный результат тот же — обе инструкции присваивания сохраняют значение `15` в переменной `spam`. Подходящее имя переменной описывает содержащиеся в нем данные. Представьте, что вы переехали в новый дом и пометили все ваши коробки как «Вещи». Так вы никогда ничего не найдете! Имена переменных `spam`, `eggs` и `bacon` — это примеры имен, используемых для переменных в этой книге.

Python создает переменную при первом ее использовании в инструкции присваивания. Чтобы проверить, какое значение находится в переменной, введите ее имя в интерактивной оболочке.

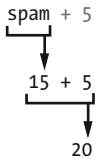
```
>>> spam = 15
>>> spam
15
```

Выражение `spam` вычисляется в значение внутри переменной `spam` и равняется `15`.

Вы также можете использовать переменные в выражениях. Введите в интерактивной оболочке следующие команды:

```
>>> spam = 15
>>> spam + 5
20
```

Вы устанавливаете значение переменной `spam`,равное `15`, поэтому `spam + 5` будет аналогично выражению `15 + 5`. Ниже приведены шаги, согласно которым вычисляется выражение `spam + 5`:



Вы не можете использовать переменную до того, как она будет создана инструкцией присваивания. Если вы попытаетесь это сделать, Python выдаст ошибку, потому что такой переменной на тот момент не существует. Ошибочное имя переменной также вызовет эту ошибку.

```
>>> spam = 15
>>> spma
Traceback (most recent call last):
    File "<pyshell#8>", line 1, in <module>
        spma
NameError: name 'spma' is not defined
```

Ошибка появилась потому, что существует переменная `spam`, но нет переменной `spma`.

Вы можете изменить значение переменной, введя еще одну инструкцию присваивания. Например, введите в интерактивной оболочке следующие команды:

```
>>> spam = 15
>>> spam + 5
20
>>> spam = 3
>>> spam + 5
8
```

Когда вы впервые вводите `spam + 5`, выражение вычисляется в сумму `20`, потому что вы храните `15` внутри переменной `spam`. Однако если вы введете `spam = 3`, значение `15` внутри переменной будет заменено или *перезаписано* значением `3`, так как переменная может хранить только одно значение. Поскольку значение переменной `spam` теперь равно `3`, когда вы вводите `spam + 5`, выражение вычисляется в сумму `8`. Перезапись — это словно извлечение одного значения из «коробки» переменной и помещение туда другого, как показано на рис. 1.4.

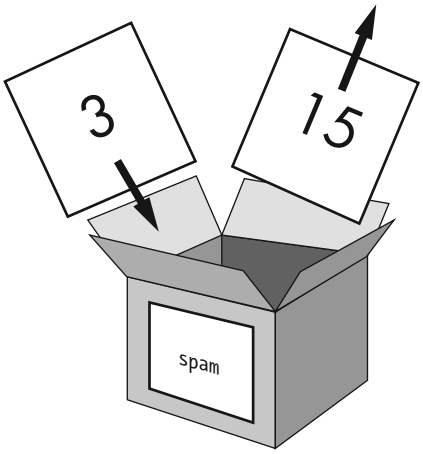


Рис. 1.4. Значение 15 в переменной spam перезаписывается значением 3

Вы даже можете использовать значение переменной spam для присвоения нового значения.

```
>>> spam = 15
>>> spam = spam + 5
20
```

Инструкция присваивания spam = spam + 5 сообщает: «Новое значение переменной spam будет текущим значением spam плюс пять». Чтобы увеличить значение переменной spam в 5 раз, введите в интерактивной оболочке следующие команды:

```
>>> spam = 15
>>> spam = spam + 5
>>> spam = spam + 5
>>> spam = spam + 5
>>> spam
30
```

В этом примере в первой инструкции вы присваиваете переменной spam значение 15. В следующей инструкции вы добавляете 5 к значению переменной spam и присваиваете ей новое значение spam + 5, которое вычисляется как 20. Когда вы делаете это три раза, вычисление переменной spam достигает 30.

Пока мы рассмотрели только одну переменную, но вы можете создать столько переменных, сколько вам нужно в ваших программах. Например, давайте присвоим разные значения двум переменным, названным eggs и bacon.

```
>>> bacon = 10
>>> eggs = 15
```

Теперь переменная bacon содержит значение 10, а переменная eggs — 15.

Каждая переменная может быть представлена как отдельная «коробка» со своим собственным значением, как показано на рис. 1.5.

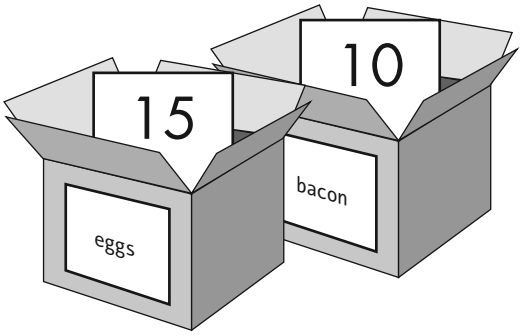


Рис. 1.5. Переменные bacon и eggs хранят разные значения

Введите инструкцию spam = bacon + eggs в интерактивной оболочке, а затем проверьте новое значение spam.

```
>>> bacon = 10
>>> eggs = 15
>>> spam = bacon + eggs
>>> spam
25
```

Теперь значение переменной spam равно 25. Когда вы складываете переменные bacon и eggs, вы складываете их значения, которые равняются 10 и 15, соответственно. Переменные содержат значения, а не выражения, поэтому переменной spam было присвоено значение 25, а не выражение bacon + eggs. После того как инструкция spam = bacon + eggs присвоила переменной spam значение 25, изменение значения переменной bacon или eggs не повлияет на значение переменной spam.

Заключение

Из этой главы вы узнали основы написания инструкций для Python. Поскольку компьютеры не имеют собственного интеллекта и понимают только конкретные инструкции, Python нуждается в ваших точных указаниях.

Выражения представляют собой значения (например, `2` или `5`) в сочетании с операторами (например, `+` или `-`). Python может вычислять выражения, т. е. приводить выражение к единому значению. Вы можете хранить значения внутри переменных, чтобы ваша программа могла запомнить эти значения и использовать их позже.

В Python также существует несколько других типов операторов и значений. В следующей главе мы рассмотрим пару основных принципов и попробуем написать первую программу. Вы узнаете о работе с текстом в выражениях. Python не ограничивается только числами и умеет гораздо больше, чем простой калькулятор!

2

НАПИСАНИЕ ПРОГРАММ

Теперь давайте посмотрим, что Python умеет делать с текстом. Почти все программы отображают пользователю некий текст, а пользователь, в свою очередь, вводит в программы текст при помощи клавиатуры. В этой главе вы создадите свою первую программу, которая выполняет обе эти операции. Вы узнаете, как хранить текст в переменных, комбинировать его и отображать на экране. Вы создадите программу, которая отобразит текст «Привет, мир!» и запросит имя пользователя.

В ЭТОЙ ГЛАВЕ РАССМАТРИВАЮТСЯ СЛЕДУЮЩИЕ ТЕМЫ:

- Строки
- Конкатенация строк
- Типы данных (например, строки или целые числа)
- Использование редактора файлов для написания программ
- Сохранение и запуск программ в IDLE
- Поток исполнения
- Комментарии
- Функция `print()`
- Функция `input()`
- Чувствительность к регистру

Строковые значения

В Python текстовые значения называются *строками*. Вы можете использовать их точно так же, как численные значения. Вы так же можете хранить

их в переменных. В коде Python строковые значения начинаются и заканчиваются символом одинарной кавычки — '.

Введите следующий код в интерактивной оболочке:

```
>>> spam = 'привет'
```

Одиночные кавычки сообщают среде начало и конец строкового значения. При этом сами кавычки частью текста не являются. Теперь, если вы введете имя переменной spam в Python, вы увидите ее значение. Помните, что среда находит значения переменных. В нашем случае это строка 'привет'.

```
>>> spam = 'привет'
>>> spam
'привет'
```

Строки могут содержать любой символ клавиатуры и могут быть любой длины. Ниже представлено несколько примеров строк.

```
'привет'
'привет всем!'
'КОТЯТА'
'7 яблок, 14 апельсинов, 3 лимона'
'Все, что не относится к слонам, не имеет значения.'
'Давным-давно, в далекой-далекой галактике...'
'O*&#wY%*&OCfsdYO*&gfC%YO*&%3yc8r2'
```

Конкатенация строк

С помощью операторов вы можете комбинировать строковые значения в выражения точно так же, как вы делали это с численными значениями. Объединение двух строк при помощи оператора + называется *конкатенацией*. Введите выражение 'Привет' + 'мир!' в интерактивной оболочке.

```
>>> 'Привет' + 'мир!'
'Приветмир!'
```

Выражение вычисляется в единое строковое значение `'Приветмир!'`. Между словами нет пробела, потому что его не было ни в одной из двух конкатенированных строк. Сравните:

```
>>> 'Привет ' + 'мир!'
'Привет мир!'
```

Оператор `+` работает со строковыми и целочисленными значениями по-разному, потому что это разные *типы данных*. Любое значение принадлежит к определенному типу данных. Тип данных значения `'Привет'` — строка. Тип данных значения `5` — целое число. При помощи типа данных Python определяет, как действуют операторы при вычислении выражений. Оператор `+` конкатенирует строковые значения, но складывает целочисленные значения и числа с плавающей запятой.

Написание кода в среде разработки

До сих пор вы вводили инструкции в IDLE по одной. Когда вы пишете программу, вы вводите несколько инструкций и запускаете их сразу. Прямо сейчас вы познаете это на практике. Пришло время написать вашу первую программу!

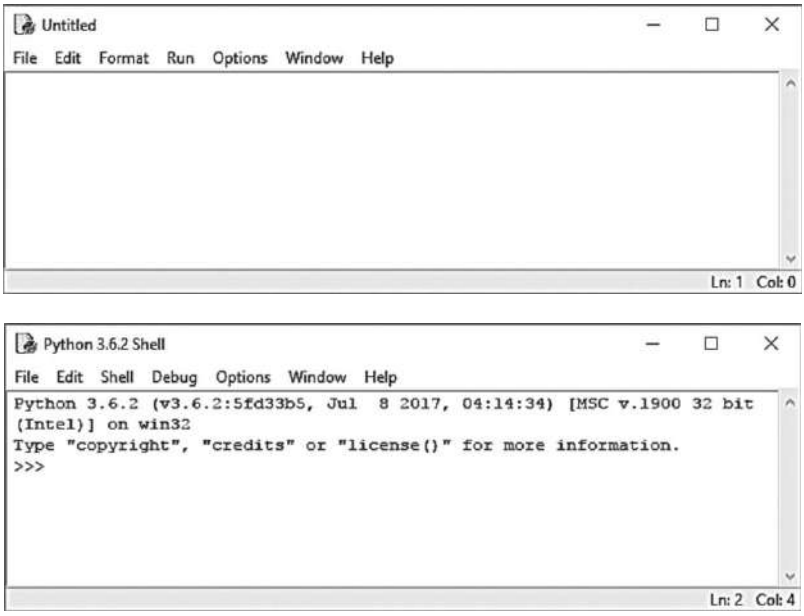


Рис. 2.1. Редактор файлов (сверху) и IDLE (снизу)

В дополнение к интерпретатору, интерактивная среда разработки имеет еще один интерфейс — редактор файлов. Чтобы открыть его, откройте меню **File** (Файл) в верхней части окна интерактивной среды разработки. Затем выберите пункт **New File** (Новый файл). Появится пустое окно для ввода кода программы, как показано на рис. 2.1.

Эти два окна хоть и похожи, но вам поможет их различить приглашение >>> в интерпретаторе Python, которого нет в редакторе файлов.

Создание программы «Привет, мир!»

Следуя традиции, наша первая программа отобразит на экране текст «Привет, мир!». Приступим. При написании программ помните, что номера в начале каждой строки кода указывать *не нужно*. Эти номера нужны для того, чтобы автор мог ссылаться на код по номеру строки. Правый нижний угол редактора файлов содержит позицию курсора, позволяя вам определить номер строки любой строки кода.

Рисунок 2.2 показывает, что курсор находится в строке 1 и столбце 0.

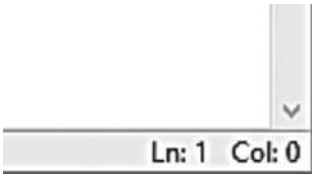


Рис. 2.2. Правый нижний угол редактора файлов указывает, в какой строке находится курсор

Введите следующий текст в окне редактора файлов. Это *исходный код* программы. Он содержит набор инструкций, который Python исполнит после запуска.

hello.py

```
1. # Эта программа здоровается и спрашивает имя.
2. print('Привет, мир!')
3. print('Как тебя зовут?')
4. myName = input()
5. print('Я так рада видеть тебя, ' + myName)
```

Интерактивная среда разработки оформляет разные типы инструкций разными цветами. Когда вы введете исходный код, окно должно выглядеть так, как показано на рис. 2.3.

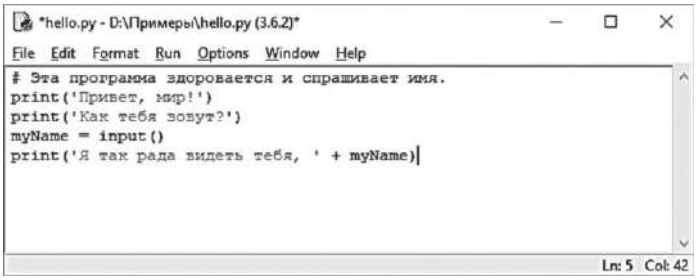


Рис. 2.3. Редактор файлов после ввода кода

Убедитесь, что ваше окно IDLE выглядит аналогично.

Сохранение программы

После того как вы ввели исходный код, вы можете сохранить его. Для этого выберите команду меню **File** ⇒ **Save As** (Файл ⇒ Сохранить как) или нажмите сочетание клавиш **Ctrl+S** на клавиатуре. На рис. 2.4 показано диалоговое окно **Сохранение** (Save As), которое откроется после этой команты. Введите слово **hello** в текстовое поле **Имя файла** (File name) и нажмите кнопку **Сохранить** (Save).

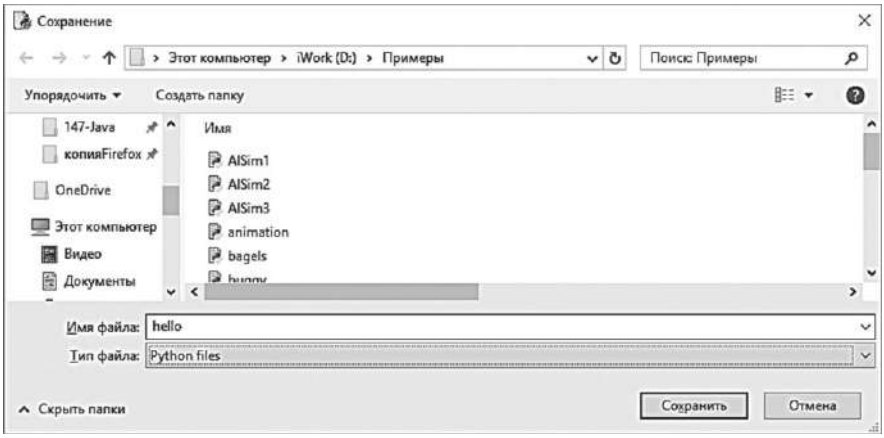


Рис. 2.4. Сохранение программы

Время от времени сохраняйте свои программы по мере написания кода. Таким образом, если компьютер внезапно выйдет из строя, или вы случайно завершите работу среды, то вам не понадобится набирать весь код заново. Чтобы загрузить ранее сохраненную программу, вызовите команду меню **File** ⇒ **Open** (Файл ⇒ Открыть). Выберите файл *hello.py* в диалоговом окне и нажмите кнопку **Открыть** (Open). Сохраненная программа *hello.py* откроется в окне редактора файлов.

Запуск программы

Теперь запустим программу. В редакторе файлов выберите команду меню **Run** ⇒ **Run Module** (Запустить ⇒ Запустить модуль) или нажмите клавишу **F5** (**fn+5** в операционной системе macOS). Ваша программа будет запущена в интерактивной оболочке. Когда программа запросит, введите свое имя. Процесс показан на рис. 2.5.

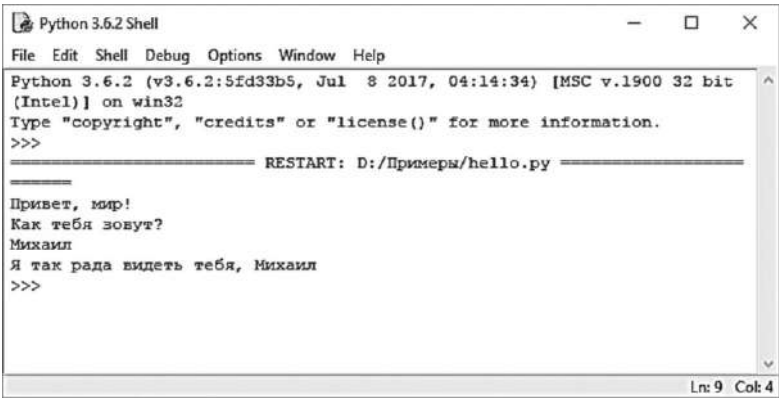


Рис. 2.5. Среда во время выполнения инструкций из файла *hello.py*

Когда вы наберете свое имя и нажмете клавишу **Enter**, программа поприветствует вас по имени. Поздравляю! Вы написали свою первую программу, что делает вас программистом. Нажмите клавишу **F5** еще раз для повторного запуска программы и введите другое имя. Если вы получили сообщение об ошибке, сравните свой код с кодом из этой книги с помощью инструмента онлайн-анализа на **inventwithpython.com/diff/**. Скопируйте и вставьте код из редактора файлов в веб-страницу и нажмите кнопку **Compare** (Сравнить). Этот инструмент выделит любые различия между вашим кодом и кодом в этой книге, как показано на рис. 2.6. Если во время работы программы вы получили ошибку `NameError`, как показано ниже, значит, вы используете Python версии 2, а не 3.

```
Привет, мир!
Как тебя зовут?
Михаил
Traceback (most recent call last):
    File "C:/Python36/test1.py", line 4, in <module>
        myName = input()
    File "<string>", line 1, in <module>
NameError: name 'Михаил' is not defined
```

Чтобы устранить проблему, установите Python 3 и перезапустите программу. (См. раздел «Загрузка и установка Python» во введении.)

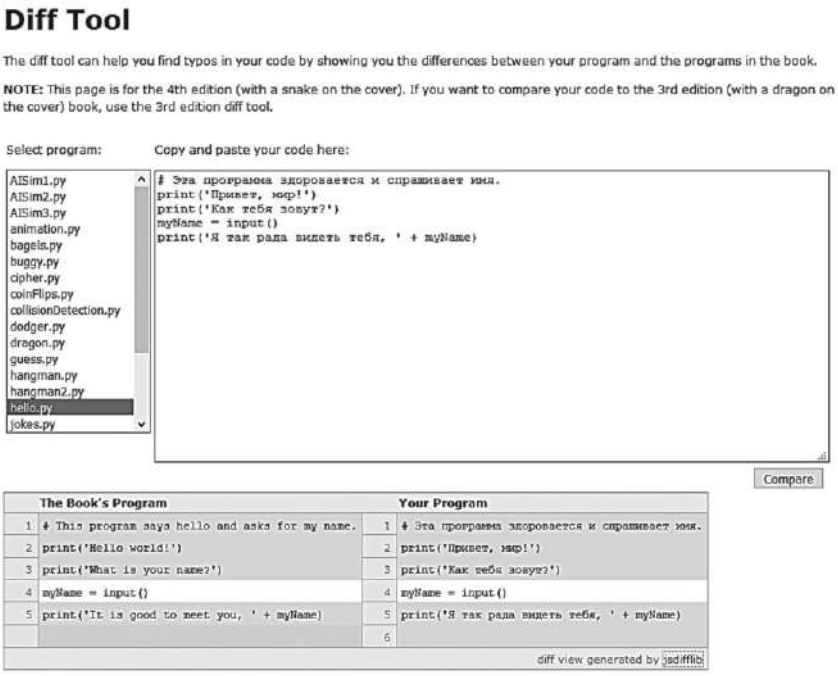


Рис. 2.6. Использование инструмента онлайн-анализа на **inventwithpython.com/diff/** (инструмент выделил строки как различающиеся из-за того, что в них использован русский текст вместо английского)

Как работает программа «Привет, мир!»

Каждая строка кода является инструкцией, которую Python будет интерпретировать. Набор этих инструкций составляет программу. Инструкции компьютерной программы в чем-то подобны этапам приготовления блюда по рецепту. Python выполняет каждую инструкцию по порядку сверху вниз.

Этап, на котором Python исполняет код программы, называется *выполнением*. Когда программа запускается, выполнение начинается с первой инструкции. После выполнения одной инструкции Python переходит к следующей. Давайте разберем каждую строку кода. Начнем со строки номер 1.

Комментарии для программиста

Первая строка программы «Привет, мир!» — это *комментарий*.

```
1. # Эта программа здоровается и спрашивает имя.
```

Комментарии — это важные заметки относительно кода. Их пишут не для Python, но для вас и других программистов. Среда игнорирует комментарии при выполнении программы. Программисты часто помещают перед основной частью кода комментарий, в котором описывают действие программы. Комментарий в программе «Привет, мир!» сообщает вам, что программа приветствует вас и спрашивает ваше имя.

Функции: Мини-программы внутри программ

Функция подобна на «мини-программе» внутри программы. Функция может содержать несколько инструкций для выполнения. Самое замечательное в функциях то, что часто вам нужно понимать лишь их предназначение и не требуется разбираться, как они работают. Python содержит некоторые встроенные функции. В программе «Привет, мир!» мы используем функции `print()` и `input()`.

Вызов функции — это инструкция, которая сообщает Python, что нужно выполнить код внутри функции. Например, ваша программа вызывает функцию `print()` для отображения строки на экране. Функция `print()` принимает строку, которую вы вводите между круглыми скобками в качестве входных данных или *аргументов*, и отображает этот текст на экране.

Функция `print()`

Строки 2 и 3 программы «Привет, мир!» — это вызовы функции `print()`.

```
2. print('Привет, мир!')
3. print('Как тебя зовут?')
```

Значение между круглыми скобками в вызове функции называется *аргументом*. Аргумент вызова функции `print()` в строке 2 — `'Привет, мир!'`, а аргумент функции `print()` в строке 3 — это `'Как тебя зовут?'`. Этот процесс называется *передачей* аргумента функции.

Функция `input()`

Строка 4 представляет собой инструкцию присваивания с переменной `myName` и вызовом функции `input()`.

```
4. myName = input()
```

Когда вызывается функция `input()`, программа ожидает ввода текста пользователем. Текстовая строка, которую вводит пользователь, становит-

ся значением, которое вычисляется при вызове функции. Вызовы функций могут использоваться в выражениях везде, где только значение может быть использовано.

Значение, которое вычисляется при вызове функции, называется *возвращаемое значением*. (Фактически «значение, возвращаемое вызовом функции» означает то же, что и «значение, вычисленное при вызове функции».) В этом случае возвращаемое значение функции `input()` представляет собой строку, введенную пользователем — его имя. Если пользователь вводит `Михаил`, вызов функции `input()` вычисляется в строку `'Михаил'`. Результат выглядит так:

```
myName = input()

myName = 'Михаил'
```

Вот так строковое значение `'Михаил'` хранится в переменной `myName`.

Выражения в вызовах функций

Последняя строка в программе «Привет, мир!» — это еще один вызов функции `print()`.

```
5. print('Я так рада видеть тебя, ' + myName)
```

Выражение `'Я так рада видеть тебя, ' + myName` находится между круглыми скобками функции `print()`. Поскольку аргументы всегда являются одиночными значениями, Python сначала вычисляет это выражение, а затем передает полученное значение в качестве аргумента. Если `'Михаил'` хранится в переменной `myName`, процесс вычисления выглядит следующим образом:

```
print('Я так рада видеть тебя, ' + myName)

print('Я так рада видеть тебя, ' + 'Михаил')

print('Я так рада видеть тебя, Михаил')
```

Так программа приветствует пользователя по имени.

Завершение программы

Когда программа выполняет последнюю строку, она *останавливается* или *завершает работу*. Это означает, что программа перестает работать. Python забывает все значения, хранящиеся в переменных, включая строку,

хранящуюся в переменной myName. Если вы снова запустите программу и введете другое имя, программа поприветствует вас по новому имени.

```
Привет, мир!
Как тебя зовут?
Каролина
Я так рада видеть тебя, Каролина
```

Помните, что компьютер выполняет именно то, что вы программируете. Компьютеры глупы и просто следуют инструкциям, которые вы им даете. Компьютеру все равно, введете ли вы свое имя, чужое или бессмысленное. Пишите что угодно. Компьютер поведет себя одинаково.

```
Привет, мир!
Как тебя зовут?
Ерунда
Я так рада видеть тебя, Ерунда
```

Имена переменных

Если имя вашей переменной будет описывать ее содержимое, то ваш код будет понятен, и работать с ним будет удобно. Вы могли бы вместо имени переменной myName использовать имя alSweigart или nAmE, что никак не повлияло бы на работу программы. Но эти имена в действительности не говорят вам о том, какую информацию хранит переменная. Как говорилось в главе 1, если вы переезжаете в новый дом и пометили все свои коробки для переезда как «Вещи», это никак не поможет вам при распаковке. В предыдущих примерах мы использовали такие имена переменных, как spam, eggs и bacon, потому что имена переменных в этих примерах не имели значения. Но в последующих программах будут применяться описательные имена, какие и должны быть и у ваших программ.

Имейте в виду, что имена переменных *чувствительны к регистру!* Это означает, что одно и то же имя, написанное без соблюдения регистра, будет относиться к другой переменной. Таким образом, в Python имена spam, SPAM, Spam и sPAM относятся к четырем разным переменным, каждая из которых содержит свои собственные значения. Во избежание путаницы вам следует избегать подобного набора имен переменных. Давайте вашим переменным описательные имена.

Как правило, имена переменных в Python состоят из строчных букв. Если имя переменной содержит более одного слова, каждое следующее слово после первого рекомендуется начинать с прописной буквы. Например, имя переменной `чтоЯКушалНаЗавтракСегодняУтром` намного легче читать, чем `чтоякушалназавтраксегодняутром` (*русские имена переменных приведены для примера, для имен нужно использовать только латинские буквы!*). Оформление имен переменных таким образом называется *горбатым* (или верблюжьим) *регистром* (потому что текст похож на горбы на спине верблюда), что повышает удобочитаемость вашего кода. Программисты также предпочитают использовать более короткие имена переменных, чтобы сделать код более понятным: `завтрак` или `сегодняшнийЗавтрак` прочитать легче, чем `чтоЯКушалНаЗавтракСегодняУтром`. Это условности необязательны, но мы рекомендуем вам придерживаться их, как общепринятых.

Заключение

Понимание основ использования строк и функций позволит вам создавать программы, которые взаимодействуют с пользователями. Это важно, потому что текст — это основной способ взаимодействия пользователя с компьютером. Пользователь вводит текст посредством клавиатуры с помощью функции `input()`, а компьютер отображает текст на экране с помощью функции `print()`.

Строки — это еще один пример значений нового типа данных. Все значения имеют тип данных, который оказывает влияние на работу оператора `+`.

Функции используются для выполнения сложных инструкций в программе. Язык Python содержит множество встроенных функций, о которых вы узнаете из этой книги позднее. Вызовы функций могут использоваться в выражениях где угодно, когда используется значение.

Инструкция или шаг в вашей программе, с которой работает Python, называется выполнением. В главе 3 вы узнаете больше о том, как выполнять инструкции в программе не по порядку. Узнав об этом, вы будете готовы создавать игры!

3

ИГРА «УГАДАЙ ЧИСЛО»

В этой главе мы напишем игру, которая называется «Угадай число». Компьютер загадает секретное число в диапазоне от 1 до 20 и попросит пользователя угадать это число. После каждой попытки угадать, компьютер будет сообщать пользователю, было ли его число больше или меньше загаданного. Пользователь выиграет, если угадает число за шесть попыток.

Написание кода для этой короткой игры полезно тем, что затрагивает много ключевых моментов программирования. Вы узнаете о том, как конвертировать значения в различные типы данных и когда к этому следует прибегать. Так как эта программа — игра, то мы будем называть пользователя *игроком*.

В ЭТОЙ ГЛАВЕ РАССМАТРИВАЮТСЯ СЛЕДУЮЩИЕ ТЕМЫ:

- Инструкции `import`
- Модули
- Функция `randint()`
- Инструкции `for`
- Блоки
- Функции `str()`, `int()` и `float()`
- Логические операторы
- Операторы сравнения
- Условия
- Разница между `=` и `==`
- Инструкции `if`
- Инструкции `break`

Пример запуска игры «Угадай число»

Вот как игрок видит вывод игры «Угадай число» во время ее прохождения. Текст, который вводит игрок, выделен полужирным шрифтом.

```
Привет! Как тебя зовут?
Михаил
Что ж, Михаил, я загадываю число от 1 до 20.
Попробуй угадать.
10
Твое число слишком большое.
Попробуй угадать.
2
Твое число слишком маленькое.
Попробуй угадать.
4
Отлично, Михаил! Ты справился за 3 попытки!
```

Исходный код игры «Угадай число»

В редакторе файлов создайте новый файл, выбрав команду меню **File ⇒ New File** (Файл ⇒ Новый файл). В открывшемся окне введите приведенный ниже исходный код и сохраните файл под именем *guess.py*. Затем запустите программу, нажав клавишу **F5**.

Когда вы вводите этот код в редактор файлов, обязательно обратите внимание на отступы в начале строк. Некоторые строки должны иметь отступы в четыре или восемь пробелов.

Если при выполнении программы возникают ошибки, сравните код, который вы набрали, с оригинальным кодом с помощью онлайн-инструмента на сайте **inventwithpython.com/diff/**.

guess.py

```
1. # Это игра по угадыванию чисел.
2. import random
3.
4. guessesTaken = 0
5.
6. print('Привет! Как тебя зовут?')
```

```
7. myName = input()
8.
9. number = random.randint(1, 20)
10. print('Что ж, ' + myName + ', я загадываю число от 1 до 20.')
11.
12. for guessesTaken in range(6):
13.     print('Попробуй угадать.') # Четыре пробела перед именем функции print
14.     guess = input()
15.     guess = int(guess)
16.
17.     if guess < number:
18.         print('Твое число слишком маленькое.') # Восемь пробелов перед именем функции print
19.
20.     if guess > number:
21.         print('Твое число слишком большое.')
22.
23.     if guess == number:
24.         break
25.
26. if guess == number:
27.     guessesTaken = str(guessesTaken + 1)
28.     print('Отлично, ' + myName + '! Ты справился за ' + guessesTaken + ' попытки!')
29.
30. if guess != number:
31.     number = str(number)
32.     print('Увы. Я загадала число ' + number + '.')
```

Импорт модуля `random`

Давайте взглянем на первые две строки этой программы.

```
1. # Это игра по угадыванию чисел.
2. import random
```

Первая строка — это комментарий, который вы уже встречали в главе 2. Помните, что интерпретатор Python будет игнорировать все, что указано после символа #. Комментарий только напоминает людям, что делает данная программа.

Вторая строка — это инструкция `import`. Помните, инструкции — это предписания, которые выполняют какое-либо действие, но не вычисляют значения, как это делают выражения. Вы уже встречали инструкцию присваивания, которая сохраняет значение переменной.

Хотя Python включает в себя множество встроенных функций, некоторые функции написаны в отдельных программах, называемых *модулями*. Вы можете использовать эти функции, импортируя соответствующие модули в вашу программу с помощью инструкции `import`.

Код в строке 2 импортирует модуль `random`. Таким образом, программа может вызвать функцию `randint()`. Эта функция будет генерировать случайное число, которое должен будет угадать игрок.

После того как вы импортировали модуль `random`, необходимо определить некоторые переменные для хранения значений, которые ваша программа будет использовать позже.

В строке 4 создается новая переменная с именем `guessesTaken`.

```
4. guessesTaken = 0
```

Эта переменная будет хранить количество попыток игрока угадать число. Так как на этом этапе программы игрок не сделал ни одной попытки, мы присваиваем ей целое значение `0`.

```
6. print('Привет! Как тебя зовут?')
7. myName = input()
```

Строки 6 и 7 аналогичны строкам из программы «Привет, мир!», описанной в главе 2. Программисты часто заимствуют код из других программ, чтобы сэкономить время на наборе кода.

Код в строке 6 — это вызов функции `print()`. Помните, что функция — это как бы мини-программа внутри вашей программы. Когда ваша программа вызывает функцию, она запускает эту мини-программу. Код внутри скобок функции `print()` — это строковый аргумент, который вы увидите на экране.

Код в строке 7 позволяет игроку ввести имя и сохраняет его в переменной `myName`. Помните, это значение необязательно может быть именем игрока. Это может быть что угодно, что введет игрок. Компьютер не имеет интеллекта и следует вашим инструкциям, вне зависимости от вводимого текста.

Генерация случайных чисел при помощи функции `random.randint()`

Теперь, когда переменные определены, вы можете использовать функцию модуля `random` для установки секретного числа, которое будет генерировать компьютер.

```
9. number = random.randint(1, 20)
```

Код в строке 9 вызывает новую функцию, называемую `randint()`, и сохраняет возвращаемое значение в качестве значения переменной `number`. Помните, что вызовы функций могут быть частью выражений, поскольку они вычисляют значение.

Функция `randint()` содержится в модуле `random`, поэтому вызывать ее нужно с помощью кода `random.randint()` (не забудьте указать точку!), чтобы Python учитывал, что функция `randint()` находится внутри модуля `random`.

Функция `randint()` вернет случайное целое число между (и включительно) двумя указанными вами целочисленными аргументами. Код в строке 9 передает числа `1` и `20`, функции — в скобках, разделенные запятой — после имени функции. Случайное целое число, которое возвращает функция `randint()`, сохраняется в качестве значения переменной с именем `number`. Именно это секретное число игрок будет пытаться угадать.

На минутку вернитесь к IDLE и введите команду **`import random`**, чтобы импортировать модуль `random`. Затем введите **`random.randint(1, 20)`**, чтобы увидеть, как происходит вычисление после вызова функции. В качестве возвращаемого значения вы получите целое число в диапазоне между `1` и `20`. Вновь введите команду и функция вернет другое целое число. Функция `randint()` каждый раз возвращает случайное целое число, подобно тому, как в результате броска игральных костей вы каждый раз получаете случайное значение. Например, введите указанные ниже команды в интерактивной оболочке. Числа, которые вы будете получать при вызове функции `randint()`, будут, скорее всего, разные (в конце концов, они ведь случайные!)

```
>>> import random
>>> random.randint(1, 20)
12
>>> random.randint(1, 20)
18
>>> random.randint(1, 20)
3
```

```
>>> random.randint(1, 20)
18
>>> random.randint(1, 20)
7
```

Вы можете попробовать разные диапазоны чисел, изменяя аргументы. Например, введите команду **random.randint(1, 4)**, чтобы получить целое число от 1 до 4 (включая 1 и 4). Или команду **random.randint(1000, 2000)**, чтобы получить число в диапазоне от 1000 до 2000.

Введите следующий код в интерактивной оболочке и посмотрите, какие числа выпадут у вас.

```
>>> random.randint(1, 4)
3
>>> random.randint(1000, 2000)
1294
```

Вы можете слегка изменить код игры, чтобы игра вела себя немного по-другому. В нашем примере мы используем целые числа в диапазоне от 1 до 20.

```
9. number = random.randint(1, 20)
10. print('Что ж, ' + myName + ', я загадываю число от 1 до 20.')
```

Попробуйте изменить диапазон целых чисел на (1, 100).

```
9. number = random.randint(1, 100)
10. print('Что ж, ' + myName + ', я загадываю число от 1 до 100.')
```

Теперь компьютер будет загадывать число между 1 и 100 вместо диапазона от 1 до 20. Изменение строки 9 повлечет изменение диапазона случайных чисел, но не забудьте также изменить текст в строке 10, чтобы игра сообщала игроку новый диапазон вместо старого.

Вы можете использовать функцию randint() каждый раз, когда вам захочется привнести в вашу игру «элемент случайности». Вы будете использовать «случайности» во многих играх (подумайте только, для какого количества настольных игр необходимо использовать игральные кости!).

Приветствие игрока

После того как компьютер присвоил случайное целое число переменной number, он приветствует игрока.

```
10. print('Что ж, ' + myName + ', я загадываю число от 1 and 20.')
```

В строке 10 функция print() приветствует игрока по имени и сообщает, что компьютер загадывает случайное число.

На первый взгляд может показаться, что в строке 10 используется больше чем один строковый аргумент, но взгляните на эту строку внимательнее. Операторы + между тремя строками связывают их в одно целое. И именно эта единая строка является аргументом функции print(). Если вы приглядитесь, то увидите, что запятые находятся внутри кавычек и являются частью строки.

Инструкции управления потоком

В предыдущих главах выполнение программ начиналось с первой инструкции программы и происходило сверху вниз, каждая инструкция по порядку. Но с помощью инструкций for, if, else и break вы можете создавать циклы или пропускать инструкции, которые соответствуют определенным условиям. Такой вид команд называется *инструкциями управления потоком*, так как они изменяют порядок выполнения кода в вашей программе.

Использование циклов для повторения кода

Код в строке 12 — это инструкция for, которая обозначает начало цикла for.

```
12. for guessesTaken in range(6):
```

Циклы позволяют выполнять фрагмент кода несколько раз. Строка 12 повторит свой код шесть раз. Инструкция for начинается с ключевого слова for, за которым следует имя новой переменной, затем ключевое слово in и вызов функции range(), для которой указано количество циклов, которое она должна выполнить, и двоеточие. Давайте рассмотрим несколько дополнительных концепций, чтобы вы могли работать с циклами.

Группировка в блоки

Несколько строк кода могут быть объединены в *блок*. Каждая строка в блоке кода начинается, по крайней мере, с того количества пробелов, что и первая строка блока. Вы можете определить начало и конец блока по количеству пробелов в начале строк. Это *отступы* строк.

Программисты на языке Python обычно вводят четыре *дополнительных* пробела в отступах, чтобы начать блок. Каждая следующая строка с таким же количеством пробелов в начале входит в состав блока. Блок заканчивается там, где находится строка кода с *тем же (меньшим) отступом, что и до* начала блока. Существуют также блоки внутри блоков — *вложенные*. Рисунок 3.1 демонстрирует схему кода с выделенными и пронумерованными блоками.

```
    12. for guessesTaken in range(6):
❶   13. ····print('Попробуй угадать.')
    14. ····guess = input()
    15. ····guess = int(guess)
    16.
    17. ····if guess < number:
❷   18. ····· ····print('Твое число слишком маленькое.')
    19.
    20. ····if guess > number:
❸   21. ····· ····print('Твое число слишком большое.')
    22.
    23. ····if guess == number:
❹   24. ····· ····break
    25.
    26. if guess == number:
```

Рис. 3.1. Пример блоков и их отступов. Серые точки отражают количество пробелов

На рис. 3.1 строка 12 не имеет отступа и находится вне блока. Строка 13 содержит отступ в четыре пробела. Это указывает на начало блока, так как у этой строки отступ больше, чем у предыдущей. Каждая последующая строка с таким же или бóльшим количеством пробелов в отступе считается частью блока ❶. Если Python встречает строку с количеством пробелов в отступе меньшим, чем в первой строке блока, это означает что блок закончен. Пустые строки при этом не учитываются.

Строка 18 содержит в начале отступ из восьми пробелов ❷. Этот блок находится *внутри* блока ❶. Но следующая строка, под номером 20, содержит отступ, в котором всего четыре пробела. Так как отступ уменьшился, становится понятно, что блок кода, который начался в строке 18 ❷, закончился, но

так как строка 20 имеет такой же отступ, что и строка 13, это означает, что соответствующий блок продолжается ❶.

В строке 21 отступ снова увеличивается до восьми пробелов, таким образом, начинается другой новый блок внутри блока: блок ❸. В строке 23 завершается блок ❸, а в строке 24 начинается последний вложенный блок — под номером ❹. Оба блока ❶ и ❹ заканчиваются строкой 24.

Создание циклов с инструкцией `for`

Инструкция `for` обозначает начало цикла. Циклы позволяют многократно выполнить один и тот же код. Когда интерпретатор достигает инструкции `for`, выполняется вход в блок, который следует за инструкцией `for`. После выполнения кода в этом блоке интерпретатор возвращается в начало блока, чтобы выполнить этот код снова.

Введите следующие команды в интерактивной оболочке.

```
>>> for i in range(3):
    print('Привет! Переменной i присвоено значение', i)

Привет! Переменной i присвоено значение 0
Привет! Переменной i присвоено значение 1
Привет! Переменной i присвоено значение 2
```

Обратите внимание, что после того, как вы ввели строку `for i in range(3):` и нажали клавишу **Enter**, в интерактивной оболочке не отобразилось новое приглашение >>>, так как среда ожидает ввод блока кода. Нажмите клавишу **Enter** снова после последней инструкции, чтобы сообщить IDLE, что вы закончили вводить блок кода. (Это применимо только в том случае, если вы работаете в интерактивной оболочке . Когда файлы *.py* создаются в редакторе файлов, вам нет необходимости оставлять пустую строку.)

Давайте взглянем на цикл `for` в строке 12 файла *guess.py*.

```
12. for guessesTaken in range(6):
13.     print('Попробуй угадать.') # Четыре пробела перед именем функции print
14.     guess = input()
15.     guess = int(guess)
16.
17.     if guess < number:
18.         print('Твое число слишком маленькое.') # Восемь пробелов перед именем функции print
```

```
19.
20.     if guess > number:
21.         print('Твое число слишком большое.')
22.
23.     if guess == number:
24.         break
25.
26. if guess == number:
```

В игре «Угадай число» блок `for` начинается с инструкции `for` в строке 12, а первой строкой после блока `for` будет строка 26.

Инструкция `for` всегда содержит двоеточие (`:`) после условия. Предполагается, что если вы закончили инструкцию двоеточием, то со следующей строки начинается новый блок. Это показано на рис. 3.2.

```
12. for guessesTaken in range(6):
13.     print('Попробуй угадать.') # Четыре пробела перед именем функции print
14.     guess = input()
15.     guess = int(guess)
16.
17.     if guess < number:
18.         print('Твое число слишком маленькое.') # Восемь пробелов перед именем функции print
19.
20.     if guess > number:
21.         print('Твое число слишком большое.')
22.
23.     if guess == number:
24.         break
25.
26. if guess == number:
```

Цикл выполняется 6 раз

Рис. 3.2. Ход выполнение цикла `for`

Рисунок 3.2 демонстрирует выполнение кода. Интерпретатор входит в блок `for` в строке 13 и продолжает выполнение сверху вниз. Когда программа достигает конца блока, вместо того, чтобы спускаться дальше к следующей строке, начинается второе выполнение цикла со строки 13. Так повторяется 6 раз, поскольку в вызове функции инструкции `for` указано `range(6)`. Каждое выполнение цикла называется *итерацией*.

Можно представить себе, что инструкция `for` сообщает: «Выполнить код в следующем блоке указанное количество раз».

Игрок угадывает число

В строках 13 и 14 компьютер предлагает игроку угадать число и предоставляет возможность ввести свой вариант.

```
13.     print('Попробуй угадать.') # Четыре пробела перед именем функции print
14.     guess = input()
```

Число, которое вводит игрок, сохраняется в переменной с именем `guess`.

Преобразование значений при помощи функций `int()`, `float()` и `str()`

В строке 15 вызывается новая функция, которая называется `int()`.

```
15.     guess = int(guess)
```

Функция `int()` принимает аргумент и возвращает значение этого аргумента в виде целого числа.

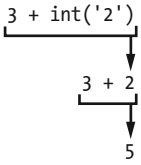
Введите следующий код в интерактивной оболочке, чтобы посмотреть, как работает функция `int()`:

```
>>> int('42')
42
```

Вызов `int('42')` возвратит целочисленное значение `42`.

```
>>> 3 + int('2')
5
```

Вызов `3 + int('2')` является примером, в котором возвращаемое значение функции `int()` используется как часть выражения. Вычисление производится до целочисленного значения `5`.



Несмотря на то что вы можете конвертировать строку в целое число, вы не можете сделать это с любой строкой.

Попытка конвертации строки `'forty-two'` (сорок два) в целое число приведет к ошибке.

```
>>> int('сорок-два')
Traceback (most recent call last):
    File "<pyshell#5>", line 1, in <module>
        int('сорок-два')
ValueError: invalid literal for int() with base 10: 'сорок-два'
```

Строка, которую вы передаете функции `int()`, должна содержать только цифры.

В игре «Угадай число» мы получаем ответ игрока при помощи функции `input()`. Помните, функция `input()` всегда возвращает *строку* текста, вводимую игроком. Если игрок ввел `5`, функция `input()` вернет строковое значение `'5'`, а не целочисленное значение `5`. Но в будущем нам понадобится сравнить ввод игрока с целочисленным значением, а Python не сможет использовать операторы < и > для сравнения строкового и целочисленного значений.

```
>>> 4 < '5'
Traceback (most recent call last):
    File "<pyshell#0>", line 1, in <module>
        4 < '5'
TypeError: unorderable types: int() < str()
```

Поэтому нам необходимо преобразовать строку в целое число.

```
14.     guess = input()
15.     guess = int(guess)
```

В строке 14 мы присваиваем переменной `guess` то строковое значение, которое ввел игрок. В строке 15 строковое значение переменной `guess` меняется на целочисленное, возвращаемое функцией `int()`. Код `int(guess)` возвращает новое целочисленное значение, которое было указано в виде строки, и оператор = присваивает это новое значение переменной `guess`. В дальнейшем это позволит программе сравнить значение переменной `guess` и определить его отношение к секретному числу в переменной `number`.

Функции `float()` и `str()` будут возвращать версии переданных им аргументов в виде числа с плавающей запятой и строки, соответственно. Введите следующий код в интерактивной оболочке:

```
>>> float('42')
42.0
>>> float(42)
42.0
```

Если функции `float()` передать строку `'42'` или целое число `42`, будет возвращено число с плавающей запятой — `42.0`.

Теперь попробуйте использовать функцию `str()`.

```
>>> str(42)
'42'
>>> str(42.0)
'42.0'
```

Если передать целое число `42` функции `str()`, она возвратит строковое значение `'42'`. Если передать ей то же число с плавающей запятой, вернется строковое значение `'42.0'`.

Используя функции `int()`, `float()` и `str()`, вы можете брать значения одного типа данных и превращать их в значения другого типа.

Логический тип данных

В Python каждое значение имеет свой тип данных. Вы уже познакомились с такими типами, как целое число, число с плавающей запятой и строка. Новый тип называется логическим или булевым. *Логический* (*булев*) тип данных может принимать одно из двух значений: `True` (Истина) или `False` (Ложь). Имена логических значений следует писать строго с соблюдением регистра: True или False.

Логические значения могут храниться в переменных, как и другие типы данных.

```
>>> spam = True
>>> eggs = False
```

В этом примере вы присваиваете переменной spam значение True, а переменной eggs — значение False. Строго соблюдайте регистр при вводе логических значений.

Вы будете использовать логические значения с операторами сравнения для создания условий. Сначала давайте разберемся, что такое операторы сравнения, а затем перейдем к условиям.

Операторы сравнения

Операторы сравнения сравнивают два значения и возвращают результат в виде логического значения True или False. В таблице 3.1 приведен перечень всех операторов сравнения в Python.

Таблица 3.1. Операторы сравнения

Оператор	Значение
<	Меньше
>	Больше
<=	Меньше или равно
>=	Больше или равно
==	Равно
!=	Не равно

Вы уже знакомы с арифметическими операторами +, -, * и /. Как и любые другие операторы, операторы сравнения объединяются со значениями для формирования выражений, таких как guessesTaken < 6.

В строке 17 программы «Угадай число» используется оператор сравнения «меньше».

```
17.     if guess < number:
```

Позднее мы рассмотрим инструкцию if более детально. Сейчас давайте взглянем на выражение, которое следует за ключевым словом if (guess < number). Это выражение содержит два значения (значения переменных guess и number), связанных между собой оператором < (меньше).

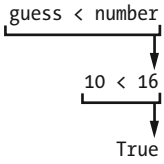
Проверка условий и определение истинности/ложности

Условие — это выражение, которое объединяет два значения при помощи оператора сравнения (такого, как < или >), выдавая в результате логическое значение. Условие — это всего лишь иное название выражения, кото-

рое определяется как истинное (`True`) или ложное (`False`). Например, условия почти всегда встречаются в инструкциях `if`.

Условие в строке 17 — `guess < number` — это вопрос: «Меньше ли значение переменной `guess`, чем значение переменной `number`?». Если да, то условие определяется истинное. Если нет — то ложное.

Предположим, переменная `guess` содержит целое число `10`, а переменная `number` — `16`. Так как `10` меньше, чем `16`, это условие будет определено, как истинное. Вычисление будет производиться следующим образом:



Эксперименты с логическими операторами, операторами сравнения и условиями

Введите следующие выражения в интерактивной оболочке, чтобы увидеть логические результаты их вычисления:

```
>>> 0 < 6
True
>>> 6 < 0
False
```

Условие `0 < 6` возвращает логическое значение `True`, так как число `0` меньше, чем число `6`. А так как `6` больше `0`, то условие `6 < 0` возвращает значение `False`.

Обратите внимание, что выражение `10 < 10` возвращает значение `False`, потому что число `10` не меньше числа `10`.

```
>>> 10 < 10
False
```

Значения одинаковые. Если абстрактные Анна и Борис — одного роста, то нельзя сказать, что один из них выше или ниже другого. Оба эти заявления были бы ложными.

Введите следующие выражения в интерактивной оболочке:

```
>>> 10 == 10
True
>>> 10 == 11
False
>>> 11 == 10
False
>>> 10 != 10
False
```

В этом примере 10 равно 10, поэтому выражение 10==10 истинно. Но 10 не равняется 11, поэтому выражение 10==11 ложно. И наконец, 10 равно 10, поэтому выражение 10!=10 ложно.

При помощи операторов сравнения вы также можете сравнивать строковые значения.

```
>>> 'Привет' == 'Привет'
True
>>> 'Прощай' != 'Привет'
True
>>> 'Привет' == 'ПРИВЕТ'
False
```

'Привет' — то же самое, что и 'Привет', поэтому выражение 'Привет' == 'Привет' истинно. 'Прощай' не равняется 'Привет', поэтому выражение 'Прощай' != 'Привет' — также истинно. Обратите внимание, что последнее выражение ложно. Вспомним, Python различает строчные и прописные буквы. Поэтому 'Привет' не равняется 'ПРИВЕТ'.

Целочисленные значения и строковые значения никогда не будут равны друг другу, поскольку они принадлежат к разным типам. Например, введите следующий код в интерактивной оболочке:

```
>>> 42 == 'Привет'
False
>>> 42 != '42'
True
```

В первом примере 42 — целое число, а 'Привет' — строка, поэтому значения не одинаковы и выражение ложно. Во втором примере строковое значе-

ние '42' так же не является целым числом, поэтому выражение «целое число 42 не равняется строковому значению '42'» истинно.

Разница между операторами = и ==

Будьте внимательны, чтобы не перепутать оператор присваивания = и оператор сравнения ==.

Знак равно (=) используется в инструкциях присваивания, чтобы сохранять (присваивать) значение переменной, а двойной знак равно (==) используется в выражениях сравнения, чтобы определить, равны ли два значения. Их очень легко случайно перепутать.

Чтобы запомнить разницу, помните, что обозначение оператора «Равно» (==), как и оператора «Не равно» (!=), состоит из двух символов.

Инструкции if

Код в строке 17 — это инструкция if.

```
17.     if guess < number:
18.         print('Твое число слишком маленькое.') # Восемь пробелов перед именем функции print
```

Блок кода, который следует за инструкцией if, будет выполнен, если условие инструкции будет определено как истинное. Если условие окажется ложным, код в блоке if будет пропущен.

Используя инструкцию if, вы можете заставить программу запускать код только тогда, когда вам это необходимо.

В строке 17 проверяется, меньше ли введенное игроком число, чем секретное число, которое загадал компьютер. Если да, интерпретатор переходит внутрь блока if к строке 18, и игрок получает сообщение, что его число меньше загаданного.

В строке 20 проверяется, что больше: введенное игроком или загаданное число.

```
20.     if guess > number:
21.         print('Твое число слишком большое.')
```

Если это условие истинно, вызов функции print() сообщает игроку, что его число больше загаданного компьютером.

Выход из цикла до его завершения при помощи инструкции `break`

Инструкция `if` в строке 23 проверяет, равно ли число, введенное игроком, числу, которое загадал компьютер. Если да, программа запускает инструкцию `break` в строке 24.

```
23.     if guess == number:
24.         break
```

Инструкция `break` незамедлительно переносит выполнение из блока `for` в первую строку после окончания этого блока. Инструкцию `break` следует вызывать только из циклов, таких как блок `for`.

Проверка, победил ли игрок

Блок `for` заканчивается на следующей строкой кода без отступов, которой является строка 26.

```
26. if guess == number:
```

Выполнение блока `for` прекращается в двух случаях: либо когда было пройдено 6 циклов (и игрок исчерпал свои попытки угадать число), либо когда выполняется инструкция `break` в строке 24 (т.е. игрок победил).

В строке 26 проверяется, угадал ли игрок число. Если да, происходит выполнение кода блока `if` в строке 27.

```
27.     guessesTaken = str(guessesTaken + 1)
28.     print('Отлично, ' + myName + '! Ты справился за ' + guessesTaken + ' попытки!')
```

Строки 27 и 28 выполняются только в том случае, если условие в инструкции `if` в строке 26 истинно (то есть если игрок угадал число).

В строке 27 вызывается функция `str()`, которая возвращает строковое значение `guessesTaken + 1` (так как диапазон функции изменяется от `0` до `5`, а не от `1` до `6`). Код в строке 28 конкатенирует строки, чтобы сообщить игроку о его выигрыше и о том, с какой попытки ему это удалось. Только строковые значения могут конкатенироваться с другими строками. Вот почему нам необходимо в строке 27 преобразовать значение `guessesTaken + 1` в строковую

форму. В противном случае при попытке объединить строку и целое число Python выдаст ошибку.

Проверка, проиграл ли игрок

Когда игрок использовал все свои попытки угадать число, интерпретатор переходит к следующей строке кода.

```
30. if guess != number:
```

В строке 30 используется оператор сравнения «Не равно», `!=`, чтобы проверить условие «последняя попытка игрока угадать не равняется числу, загаданному компьютером». Если это условие истинно, интерпретатор переходит к блоку `if` в строке 31.

Строки 31 и 32 внутри блока `if` выполняются только в том случае, если условие в строке 30 истинно.

```
31.     number = str(number)
32.     print('Увы. Я загадала число ' + number + '.')
```

В этом блоке программа сообщает игроку загаданное число. Для этого необходимо конкатенировать строки, но переменная `number` содержит целочисленное значение. В строке 31 мы перезаписываем значение переменной `number` в виде строки, чтобы можно было конкатенировать строки и получить результат — `'Увы. Я загадала число ' + number + '.'` в строке 32.

В этой точке достигается конец кода, и выполнение программы завершается. Поздравляю! Вы только что написали свою первую игру!

Вы можете изменять сложность игры, настраивая максимальное количество попыток угадать число. Чтобы предоставить игроку всего четыре попытки, измените код в строке 12 следующим образом:

```
12. for guessesTaken in range(4):
```

Передав функции `range()` аргумент `4`, вы определяете, что код внутри цикла будет выполняться четыре раза вместо шести. Это существенно усложняет игру. Чтобы упростить игру, введите большее целое число в вызов функции `range()`. Это приведет к тому, что цикл будет выполняться *большее* количество раз, и у игрока будет *больше* попыток угадать число.

Заключение

Программирование — написание кода программ — это создание программ, которые будут выполняться компьютером.

Когда вы видите, что кто-то работает с программой (например, играет в вашу игру «Угадай число»), все что вы видите, — это текст на экране компьютера. Программа решает, какой текст выводить на экран (программный *вывод*), основываясь на указанных вами инструкциях и на том, что игрок вводит с клавиатуры (программный *ввод*). Программа — это всего лишь набор инструкций, которые реагируют на ввод пользователя.

В Python существует несколько видов инструкций:

- **Выражения** — значения, связанные операторами. Все выражения вычисляются в одно значение. Например, в результате вычисления `2 + 2` получается `4`, а результатом выражения `'Привет' + ', ' + 'мир!'` будет строка `'Привет, мир!'`. Когда выражения следуют за ключевыми словами `if` и `for`, их еще называют *условиями*.
- **Инструкции присваивания** сохраняют значения в переменных для их дальнейшего использования в программе.
- **Инструкции `if`, `for` и `break`** управляют потоком и позволяют пропускать инструкции, зацикливать выполнение инструкций или прекращать выполнение циклов. Вызов функций позволяет изменить ход выполнения, осуществив переход к набору инструкций внутри функции.
- **Функции `print()` и `input()`** отображают текст на экране и получают текст, набранный с клавиатуры. Инструкции, которые имеют дело с программным вводом и выводом, называются *инструкциями ввода/вывода*.

Вот и все — только эти четыре пункта. Конечно же, еще предстоит изучить множество деталей об этих четырех типах инструкций. В следующих главах вы узнаете больше типов данных и операторов, изучите больше инструкций управления потоком и много других функций Python. Существуют также разные способы ввода/вывода помимо текстового, такие как ввод с помощью мыши и звуковой/графический вывод.

4

ПРОГРАММА-ШУТНИК

Программа из этой главы рассказывает пользователю несколько анекдотов и демонстрирует более продвинутые способы использования функции `print()`. В большинстве игр из этой книги для ввода и вывода применяется обычный текст. Ввод производит пользователь путем набора на клавиатуре, а вывод — это текст, который отображается на экране.

В ЭТОЙ ГЛАВЕ РАССМАТРИВАЮТСЯ СЛЕДУЮЩИЕ ТЕМЫ:

- Экранируемые символы
- Использование одинарных и двойных кавычек в строках
- Использование ключевого слова `end` в функции `print()` для пропуска символа новой строки

Вы уже знаете, как отображать обычный текстовый вывод с помощью функции `print()`. Теперь давайте ближе рассмотрим, как в Python работают строки и функция `print()`.

Пример запуска программы «Шутки»

Вот что видит пользователь, когда выполняется программа «Шутки»:

```
Что получится, если скрестить снеговика с вампиром?
Обморожение!

Что дантисты называют 'черной дырой'?
```

```
Кариес!

Тук-тук.
Кто там?
Невежливая корова.
Невежливая корова?-МУУУ!
```

Исходный код программы «Шутки»

В редакторе файлов создайте новый файл, выбрав команду меню **File** ⇒ **New File** (Файл ⇒ Новый файл). В открывшемся окне введите приведенный ниже исходный код и сохраните файл под именем *jokes.py*. Затем нажмите клавишу **F5** и запустите программу. Если при выполнении программы возникают ошибки, сравните код, который вы набрали, с оригинальным кодом с помощью онлайн-инструмента на сайте **inventwithpython.com/diff/**.

jokes.py

```
print('Что получится, если скрестить снеговика с вампиром?')
input()
print('Обморожение!')
print()
print('Что дантисты называют \'черной дырой\'?')
input()
print('Кариес!')
print()
print('Тук-тук.')
input()
print("Кто там?")
input()
print('Невежливая корова.')
input()
print('Невежливая корова?', end='')
print('-МУУУ!')
```

Как работает код

Для начала взглянем на первые четыре строки кода.

```
1. print('Что получится, если скрестить снеговика с вампиром?')
2. input()
3. print('Обморожение!')
4. print()
```

В строках 1 и 3 вызов функции `print()` используется для того, чтобы задать вопрос и дать на него ответ в пределах первой шутки. Вы ведь не хотите, чтобы пользователь сразу же прочел ответ на шутливый вопрос, поэтому требуется вызов функции `input()` после первого вызова `print()`. Пользователь прочитает шутку, нажмет клавишу **Enter**, а затем прочитает окончание.

Пользователь может ввести некую строку и нажать клавишу **Enter**, но эта возвращаемая строка не сохранится ни в одной переменной. Программа ее попросту проигнорирует и перейдет к следующей строке кода.

Последний вызов функции `print()` не содержит строкового аргумента. Так программе передается инструкция вывести пустую строку. Пустые строки полезны для того, чтобы избежать нагромождения текста.

Экранируемые символы

Строки с 5 по 8 выводят на экран текст (вопрос и ответ) следующей шутки.

```
5. print('Что дантисты называют \'черной дырой\'?')
6. input()
7. print('Кариес!')
8. print()
```

Обратите внимание, что символ \ — это обратный слеш (или обратная косая черта), а / — прямой слеш (или просто косая черта). В строке 5 два раза указан обратный слеш (косая черта) прямо перед одиночной кавычкой: `\'`. Этот обратный слеш означает, что символ, указанный следом, экранируемый . *Экранируемые символы,* позволяют выводить специальные символы, которые трудно или невозможно обычным образом вставить в исходный код. К примеру, к таким символам относится одиночная кавычка в строковом значении, которое само по себе начинается и заканчивается одинарными кавычками.

В этом случае, если мы не укажем символы обратного слеша, одинарные кавычки в тексте `\'черной дырой\'` будут расценены как конец строки, и мы получим ошибку синтаксиса. Но эти кавычки должны быть частью стро-

ки. Если их заэкранировать, они будут выведены в составе строкового значения.

Что же делать, если вы захотите отобразить обратный слеш?

Завершите работу вашей программы *jokes.py* и в интерактивной оболочке введите следующую инструкцию `print()`:

```
>>> print('Фото в папке d:\photo\tennis.')
Фото в папке d:\photo ennis.
```

Здесь не отобразился второй обратный слеш, потому что буква `t` после него была определена как экранированный символ. Символы `\t` распознаются как символ табуляции (нажатие клавиши **Tab** на клавиатуре).

Следующая строка кода правильно отобразит текст:

```
>>> print('Фото в папке d:\photo\\tennis.')
Фото в папке d:\photo\tennis.
```

В этом случае последовательность `\\` определяется как символ обратного слеша и сочетание `\t` перестает выводиться как символ табуляции.

В таблице 4.1 представлен список некоторых экранированных символов в Python, включая `\n`, который обозначает символ перехода на новую строку, ранее использованный нами.

Таблица 4.1. Экранированные символы

Экранированный символ	На экран выводится
`\\`	Обратный слеш (\)
`\'`	Одинарная кавычка (')
`\"`	Двойные кавычки (")
`\n`	Новая строка
`\t`	Табуляция

В Python существует еще несколько экранированных символов, но указанные выше вы будете чаще всего использовать во время разработки своих игр.

Одинарные и двойные кавычки

Пока мы еще находимся в интерактивной оболочке, давайте подробнее рассмотрим кавычки. Строки в Python не всегда обрамляются одинарными

кавычками. Вы можете использовать и двойные кавычки. Следующие две строки выводятся одинаково:

```
>>> print('Привет, мир!')
Привет, мир!
>>> print("Привет, мир!")
Привет, мир!
```

Но вы не можете использовать и те и другие кавычки одновременно, иначе получите сообщение об ошибке (достигнут конец строки во время исполнения строки с одной кавычкой) .

```
>>> print('Привет, мир!")
SyntaxError: EOL while scanning single-quoted string
```

Я предпочитаю использовать одинарные кавычки, потому что нет необходимости удерживать клавишу **Shift** во время их ввода. Их проще набрать с клавиатуры, а для Python нет никакой разницы.

Также обратите внимание, что по аналогии с символами \' в строке, обрамленной одинарными кавычками, следует использовать символы \" в строке, обрамленной двойными кавычками.

Взгляните на пример:

```
>>> print('Я попросил у Артура \'Мерс\' на неделю. Он сказал, "Конечно!"')
print('Я попросил у Артура \'Мерс\' на неделю. Он сказал, "Конечно!"')
```

Вы используете одинарные кавычки в начале и в конце строки, поэтому необходимо добавить обратный слеш перед одинарными кавычками, окружающими слово «Мерс». Но при использовании кавычек в слове «Конечно!» обратный слеш не нужен. Интерпретатор Python достаточно сообразителен, чтобы понять, что строка, которая начинается с одного типа кавычек, не может заканчиваться другим типом.

Теперь взгляните на другой пример:

```
>>> print("Она сказала, \"Не могу поверить, что ты одолжил им 'Мерс'.\"")
print("Она сказала, \"Не могу поверить, что ты одолжил им 'Мерс'.\"")
```

Мы заключили строку в двойные кавычки, поэтому потребовалось добавить обратный слеш ко всем двойным кавычкам внутри строки. Нам нет необходимости экранировать одинарные кавычки вокруг слова «Мерс».

Итак, в строках, обрамленных одинарными кавычками, вам не нужно экранировать двойные кавычки, но вы должны экранировать одинарные. Аналогично в строках, обрамленных двойными кавычками, вам не нужно экранировать одинарные и нужно экранировать двойные.

Параметр `end` функции `print()`

Теперь вернемся к файлу *jokes.py* и взглянем на строки 9–16.

```
9. print('Тук-тук.')
10. input()
11. print("Кто там?")
12. input()
13. print('Невежливая корова.')
14. input()
15. print('Невежливая корова?', end='')
16. print('-МУУУ!')
```

Обратили ли вы внимание на второй аргумент функции `print()` в строке 15? Обычно функция `print()` добавляет символ новой строки в конце строки, которую печатает. Поэтому функция `print()` без аргументов просто выводит пустую строку. Но у `print()` есть еще один необязательный параметр — `end`.

Помните, что аргумент — это значение, которое передается в вызове функции. Пустая строка, которая передается функции `print()`, называется *позиционным аргументом*. А `end=''` — именованным аргументом. Чтобы передать именованный аргумент функции, вы должны ввести перед значением аргумента его имя — как, например, `end=`.

Когда мы запустим эту часть кода, то получим результат:

```
Тук-тук.
Кто там?
Невежливая корова.
Невежливая корова?-МУУУ!
```

Так как мы передали пустую строку параметру `end`, функция `print()` отобразит пропуск вместо перехода на новую строку. Вот почему текст `'-МУУУ!'`

отображается в той же строке, а не новой. Символ новой строки после строки `'Невежливая корова?'` не выводится.

Заключение

В этой главе мы рассмотрели различные способы использования функции `print()`. Экранированные символы используются для вывода специальных символов. Если в строке вам нужно использовать специальные символы, укажите обратный слеш `\` вместе с буквой специального символа. Например, `\n` — символ новой строки. Если же в строке нужно вывести обратный слеш, используйте сочетание символов `\\`.

Функция `print()` автоматически добавляет символ новой строки в конец каждой строки. Чаще всего это полезно. Но иногда переход на новую строку не требуется. Для этого вы можете добавить в функцию `print()` именованный аргумент `end`. Например, чтобы вывести на экран слово `спам` без перехода на новую строку, следует использовать код `print('спам', end='')`.

5

ИГРА «ЦАРСТВО ДРАКОНОВ»

В этой главе вы создадите игру, которая называется «Царство драконов». В ней игрок выбирает между двумя пещерами, в одной из которых его ждет сокровище, а в другой — гибель.

Как играть в «Царство драконов»

Согласно условиям, игрок находится в землях, заселенных драконами. Все драконы живут в пещерах с кучами сокровищ, которые они насобирали. Некоторые драконы дружелюбны и делятся своими сокровищами. Другие — злы и съедают любого, кто попадает в их пещеру.

Игрок приближается к двум пещерам: одна — с дружелюбным драконом, а вторая — со злым. Игрок должен сделать выбор, не зная, какой из драконов ему встретится.

В ЭТОЙ ГЛАВЕ РАССМАТРИВАЮТСЯ СЛЕДУЮЩИЕ ТЕМЫ:

- Блок-схемы
- Создание собственных функций с помощью ключевого слова `def`
- Многострочный текст
- Инструкции `while`
- Логические операторы `and`, `or` и `not`
- Таблицы истинности
- Ключевое слово `return`
- Глобальная и локальная области видимости переменных
- Параметры и аргументы
- Функция `sleep()`

Пример запуска игры «Царство драконов»

Вот как выглядит эта игра при запуске. Текст, который вводит игрок, выделен полужирным шрифтом.

```
Вы находитесь в землях, заселенных драконами.
Перед собой вы видите две пещеры.
В одной из них — дружелюбный дракон,
который готов поделиться с вами своими сокровищами.
Во второй — `жадный и голодный дракон, который мигом вас съест.
В какую пещеру вы войдете? (нажмите клавишу 1 или 2)
1
Вы приближаетесь к пещере...
Ее темнота заставляет вас дрожать от страха...
Большой дракон выпрыгивает перед вами! Он раскрывает свою пасть и...
...моментально вас съедает!
Попытаете удачу еще раз? (да или нет)
нет
```

Блок-схема игры «Царство драконов»

Чаще всего стоит записать все, что должна выполнять ваша игра или программа, до начала работы над кодом. Этот этап называется *проектированием программы*.

В этом вам поможет создание блок-схемы. Блок-схема — это диаграмма, показывающая все возможные действия, которые могут происходить в игре, и их связи между собой. На рис. 5.1 показана блок-схема для игры «Царство драконов».

Чтобы узнать, что произойдет во время игры, поместите свой палец на поле «Старт».

Далее следуйте по стрелкам от этого блока к другим. Ваш палец играет роль интерпретатора программы. Выполнение программы заканчивается, когда ваш палец достигнет блока «Конец».

Когда вы доберетесь до блока «Проверка дракона (дружелюбный или голодный)», далее вы можете попасть как в блок «Игрок побеждает», так и в блок «Игрок проигрывает».

В этой точке ветвления выполнение программы может пойти в разных направлениях. В любом случае оба пути в конечном итоге попадут в блок «Спрашиваем, хочет ли сыграть снова».

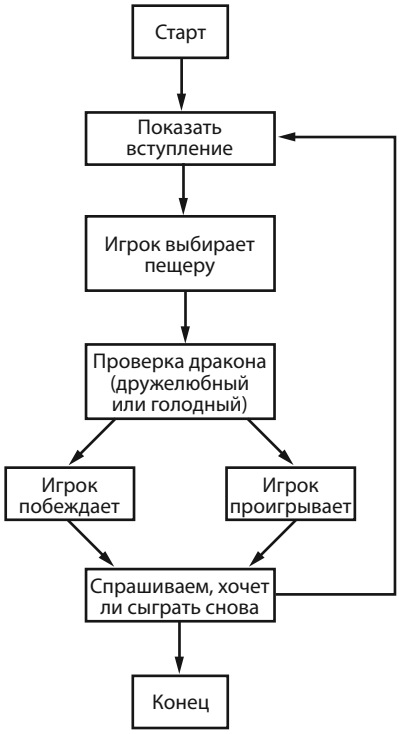


Рис. 5.1. Блок-схема игры «Царство драконов»

Исходный код игры «Царство драконов»

В редакторе файлов создайте новый файл, выбрав команду меню **File** ⇒ **New File** (Файл ⇒ Новый файл). В открывшемся окне введите приведенный ниже исходный код и сохраните файл под именем *dragon.py.* Нажмите клавишу **F5** для запуска программы. Если при ее выполнении возникают ошибки, сравните код, который вы набрали, с оригинальным кодом с помощью онлайн-инструмента на сайте **inventwithpython.com/diff/**.

dragon.py

```
import random
import time

def displayIntro():
```

```
5.     print('''Вы находитесь в землях, заселенных драконами.
6. Перед собой вы видите две пещеры. В одной из них — дружелюбный дракон,
7. который готов поделиться с вами своими сокровищами. Во второй —
8. жадный и голодный дракон, который мигом вас съест.''')
9.     print()
10.
11. def chooseCave():
12.     cave = ''
13.     while cave != '1' and cave != '2':
14.         print('В какую пещеру вы войдете? (нажмите клавишу 1 или 2)')
15.         cave = input()
16.
17.     return cave
18.
19. def checkCave(chosenCave):
20.     print('Вы приближаетесь к пещере...')
21.     time.sleep(2)
22.     print('Ее темнота заставляет вас дрожать от страха...')
23.     time.sleep(2)
24.     print('Большой дракон выпрыгивает перед вами! Он раскрывает свою пасть и...')
25.     print()
26.     time.sleep(2)
27.
28.     friendlyCave = random.randint(1, 2)
29.
30.     if chosenCave == str(friendlyCave):
31.         print('...делится с вами своими сокровищами!')
32.     else:
33.         print('...моментально вас съедает!')
34.
35. playAgain = 'да'
36. while playAgain == 'да' or playAgain == 'д':
37.     displayIntro()
38.     caveNumber = chooseCave()
39.     checkCave(caveNumber)
40.
41.     print('Попытаете удачу еще раз? (да или нет)')
42.     playAgain = input()
```

Давайте проанализируем исходный код.

Импорт модулей random и time

Эта программа импортирует два модуля:

```
1. import random
2. import time
```

Модуль random предоставляет функцию randint(), которую мы использовали в игре «Угадай число» в главе 3. Код в строке 2 импортирует модуль time, содержащий функции для работы со временем.

Функции в игре «Царство драконов»

Функции позволяют запускать один и тот же код несколько раз без необходимости повторного копирования и вставки этого кода. Вы помещаете код внутрь функции, которую вызываете при необходимости. И, поскольку вы пишете этот фрагмент кода только один, то в случае ошибки функции вам нужно будет внести изменения только в одном месте кода программы.

Вы уже использовали несколько функций, такие как print(), input(), randint(), str() и int(). Ваши программы вызывали эти функции для выполнения их кода. В игре «Царство драконов» вы будете писать собственные функции с помощью инструкции def.

Инструкции def

Код в строке 4 — это как раз и есть инструкция def.

```
4. def displayIntro():
5.     print('''Вы находитесь в землях, заселенных драконами.
6. Перед собой вы видите две пещеры. В одной из них — дружелюбный дракон,
7. который готов поделиться с вами своими сокровищами. Во второй —
8. жадный и голодный дракон, который мигом вас съест.''')
9.     print()
```

Инструкция def определяет новую функцию (в данном случае — функцию displayIntro()), которую вы можете вызвать позже в программе.

Рисунок 5.2 демонстрирует часть инструкции def. Она содержит ключевое слово def, за которым следует имя функции с круглыми скобками, а затем двоеточие (:). Блок после инструкции def называется блоком def.

Рис. 5.2. Компоненты инструкции `def`

Вызов функции

Когда вы *определяете* функцию, вы указываете инструкции для ее запуска в следующем блоке. Когда вы *вызываете* функцию, выполняется код внутри блока `def`. Пока вы не вызовете эту функцию, инструкции в блоке `def` не будут выполняться.

Другими словами, когда интерпретатор доходит до инструкции `def`, он пропускает следующей за ней блок `def` целиком и переходит к первой строке после него. При вызове функции интерпретатор переходит внутрь этой функции к первой строке кода в блоке `def`.

Например, взгляните на вызов функции `displayIntro()` в строке 37:

```
37.     displayIntro()
```

Вызов этой функции запускает функцию `print()`, которая выводит на экран вступление с текстом «Вы находитесь в землях, заселенных драконами…».

Где указывать определение функций?

Определение функции и блок `def` должны *предшествовать* вызову функции. Точно так же, как вы сначала определяете значение переменной, а затем уже используете эту переменную. Если вы укажете вызов функции перед ее определением, при выполнении программы отобразится ошибка. Давайте рассмотрим маленькую программу в качестве примера. Откройте новое окно редактора файлов, введите указанный ниже код, сохраните его под именем *example.py* и запустите.

```
sayGoodbye()

def sayGoodbye():
    print('Пока!')
```

При попытке запустить эту программу вы получите следующее сообщение об ошибке:

```
Traceback (most recent call last):
  File "C:/Users/Al/AppData/Local/Programs/Python/Python36/example.py",
    line 1, in <module>
    sayGoodbye()
NameError: name 'sayGoodbye' is not defined
```

Чтобы исправить эту ошибку, поменяйте местами определение функции и ее вызов.

```
def sayGoodbye():
    print('Пока!')

sayGoodbye()
```

Теперь функция `sayGoodbye()` определена прежде, чем будет вызвана, таким образом Python будет знать, как ее интерпретировать. В противном случае Python не будет иметь инструкций для функции `sayGoodbye()` во время ее вызова и, таким образом, не сможет ее обработать и вывести текст.

Многострочный текст

До сих пор весь текст в наших вызовах функции `print()` помещался на одной строке и ограничивался кавычками. Но если указать по три кавычки в начале и в конце строки, тогда можно ввести несколько строк текста. Это так называемый *многострочный текст*.

Введите следующий код в интерактивной оболочке, чтобы увидеть пример работы с многострочным текстом:

```
>>> fizz = '''Дорогая Алиса!
В конце месяца я возвращаюсь к Кате.
Твой друг,
Борис'''
>>> print(fizz)
Дорогая Алиса!
В конце месяца я возвращаюсь к Кате.
Твой друг,
Борис
```

Обратите внимание на переносы строк во время вывода текста на экран. В многострочных блоках текста символы перехода на новую строку учитываются в составе строк. Так как вы указываете три кавычки подряд, вам не нужно использовать экранирующие символы `\n` или кавычки другого типа. Такие переносы строк упрощают чтение, когда речь идет о больших объемах текста.

Выполнение циклов с помощью инструкций `while`

В строке 11 определяется другая функция, `chooseCave()`

```
11. def chooseCave():
```

Код этой функции предназначен для того, чтобы запросить у игрока, в какую пещеру он хочет войти, `1` или `2`. Для осуществления выбора мы воспользовались инструкцией `while` — новым типом цикла, с которым вы еще не работали.

В отличие от цикла `for`, который выполняет определенное количество раз (или итераций), цикл `while` повторяется до тех пор, пока выполняется определенное условие. Когда интерпретатор доходит до инструкции `while`, учитываются условия, указанные после ключевого слова `while`. Если условие истинно, интерпретатор переходит к блоку `while`. Если условие ложно, блок `while` игнорируется.

Вам может показаться, что инструкция `while` почти то же самое, что и инструкция `if`. Выполнение блоков кода обеих инструкций осуществляется в случае, если условие истинно. Отличие в том, что когда интерпретатор достигает конца блока в цикле `while`, он возвращается к инструкции `while`, чтобы перепроверить условие.

Взгляните на блок `def` функции `chooseCave()`, чтобы увидеть цикл `while` в действии:

```
12.     cave = ''
13.     while cave != '1' and cave != '2':
```

В строке 12 создается новая переменная с именем `cave`, и ей присваивается пустая строка. Затем в строке 13 начинается цикл `while`. Функция `chooseCave()` должна убедиться, что игрок входит в пещеру `1` или `2`, а не куда-либо еще. Цикл здесь служит для того, чтобы программа продолжала спрашивать игро-

ка, пока он не выберет один из двух допустимых вариантов. Этот процесс называется *проверкой ввода*.

Условие также содержит новый оператор, с которым вы еще не знакомы — `and`. Так же, как — и `*` — это арифметические операторы, а `==` и `!=` — операторы сравнения, оператор `and` — это логический оператор. Давайте более детально рассмотрим логические операторы.

Логические операторы

Логические операторы имеют дело с выражениями, которые могут быть либо истинными, либо ложными. Эти операторы сравнивают значения и возвращают логическое значение.

Задумайтесь над предложением «У кошек есть усы, а у собак есть хвосты». Выражение «У кошек есть усы» истинно, и «У собак есть хвосты» также истинно, таким образом, в целом предложение «у кошек есть усы, а у собак есть хвосты» — истинно.

Но предложение «у кошек есть усы, а у собак есть крылья» будет ложно. Даже несмотря на истинность выражения, что «у кошек есть усы», у собак быть не может крыльев, поэтому выражение «у кошек есть усы, а у собак есть крылья» в целом — ложно. Согласно булевой логике, выражение может быть либо полностью истинным, либо абсолютно ложным. Из-за слова `and` (как союза «и») все предложение истинно только в том случае, если истинны *обе* его части. Если одна или обе части ложны, то и все предложение ложно.

Оператор `and`

Для оператора `and` в Python также необходимо, чтобы все логическое выражение было истинно или ложно. Если логические значения по обеим сторонам от ключевого слова `and` истинны, выражение определяется как истинное. Если же одно или оба значения ложны, то и все выражение определяется как ложное.

Введите следующие выражения с оператором `and` в интерактивной оболочке:

```
>>> True and True
True
>>> True and False
False
>>> False and True
False
>>> False and False
```

```
False
>>> spam = 'привет'
>>> 10 < 20 and spam == 'привет'
True
```

Оператор `and` может быть использован для вычисления любых двух логических выражений. В последнем примере выражение `10 < 20` истинно и выражение `spam == 'привет'` также истинно, таким образом, два логических выражения, объединенных оператором `and`, истинны.

Если вы вдруг забудете принцип работы логических операторов, можете подсмотреть в следующую *таблицу истинности*, которая отражает все варианты комбинаций логических значений. Таблица 5.1 содержит все комбинации для оператора `and`.

Таблица 5.1. Таблица истинности оператора `and`

A and Б	Результат
True and True	True
True and False	False
False and True	False
False and False	False

Оператор `or`

Оператор `or` похож на оператор `and`, за исключением того, что он принимает решение об истинности выражения, если *одна* (любая) из двух его частей истинна, либо если истинны обе части. Единственный раз, когда оператор `or` определяет выражение как ложное, — если ложны *обе* части выражения.

Введите следующий код в интерактивной оболочке:

```
>>> True or True
True
>>> True or False
True
>>> False or True
True
>>> False or False
False
>>> 10 > 20 or 20 > 10
True
```

В последнем примере 10 не превышает 20, но 20 больше 10, поэтому первая часть выражения определяется как ложная, а вторая — как истинная. Так как вторая часть выражения истинна, все выражение целиком становится истинным. В таблице 5.2 приведена таблица истинности оператора or.

Таблице 5.2. Таблица истинности оператора or

A or Б	Результат
True or True	True
True or False	True
False or True	True
False or False	False

Оператор not

Вместо объединения двух значений оператор not работает только с одним значением. Оператор not вычисляет противоположное логическое значение: истинные выражения определяются как ложные, а ложные — как истинные. Введите следующий код в интерактивной оболочке:

```
>>> not True
False
>>> not False
True
>>> not ('черное' == 'белое')
True
```

Оператор not также может использоваться для любого логического выражения. В последнем примере выражение 'черное' == 'белое' определяется как ложное. Вот почему выражение not ('black' == 'white') в результате истинно. Таблица истинности оператора not приведена в таблице 5.3.

Таблице 5.3. Таблица истинности оператора not

not A	Результат
not True	False
not False	True

Вычисление логических операций

Взгляните снова на строку 13 игры «Царство драконов»:

```
13.     while cave != '1' and cave != '2':
```

Условие состоит из двух частей, связанных логическим оператором `and`. Условие истинно только в случае, если истинны обе его части.

В первый раз, когда условие выполнения инструкций `while` было проверено, строка `cave` оставалась пустой, `''`. Пустая строка не равна строке `'1'`, поэтому левая часть выражения истинна. Пустая строка также не равна строке `'2'`, поэтому правая часть выражения также истинна.

Таким образом, условие выглядит как `True and True`. Поскольку оба значения `True`, все условие определяется как истинное, поэтому интерпретатор переходит к выполнению блока `while`, в котором программа будет пытаться присвоить значение переменной `cave` (вместо пустой строки). В строке 14 код запрашивает у игрока выбор пещеры.

```
13.     while cave != '1' and cave != '2':
14.         print('В какую пещеру вы войдете? (нажмите клавишу 1 или 2)')
15.         cave = input()
```

Код в строке 15 позволяет игроку ввести свой ответ и нажать клавишу **Enter**. Этот ответ сохраняется в переменной `cave`. После выполнения этого кода интерпретатор возвращается обратно к инструкции `while` и перепроверяет условие в строке 13.

Если игрок ввел `1` или `2`, тогда переменной `cave` будет присвоено значение либо `'1'`, либо `'2'` (так как функция `input()` всегда возвращает строковое значение). После этого условие становится ложным, и выполнение программы будет продолжаться после цикла `while`. Например, если пользователь ввел `'1'`, тогда процесс вычисления будет выглядеть так:

```
while cave != '1' and cave != '2':
      ↓
while  '1' != '1' and cave != '2':
           ↓
while     False   and cave != '2':
                      ↓
while     False   and  '1' != '2':
                           ↓
while     False   and     True   :
           ↓
while     False                  :
```

Но если игрок ввел `3`, `4` или `ПРИВЕТ`, такой ответ будет недействительным. Тогда условие останется истинным, и интерпретатор будет запускать код

в блоке while, чтобы снова спросить игрока. Программа продолжит просить игрока выбрать пещеру, пока он не введет 1 или 2. Это гарантирует, что после выполнения кода переменная cave будет содержать допустимый ответ.

Возвращаемые значения функций

Код в строке 17 содержит новую инструкцию return.

```
17.     return cave
```

Инструкция return размещается только внутри определяющих функцию блоков def, в нашем случае selectCave(). Помните, как функция input() считывает строку и возвращает строковое значение, введенное игроком? Функция selectCave() также вернет значение. Код в строке 17 возвращает значение, которое хранится в переменной cave, — либо '1', либо '2'.

После выполнения инструкции return интерпретатор сразу же выходит из блока def (так же, как после выполнения инструкции break программа завершает выполнение блока while). Интерпретатор возвращается к строке с вызовом функции. Сам вызов функции будет определять возвращаемое значение функции.

Перейдите на мгновение к строке 38, где вызывается функция selectCave().

```
38.     caveNumber = chooseCave()
```

В строке 38, когда программа вызывает функцию chooseCave(), определенную в строке 11, вызов функции определяет строковое значение переменной cave, которое затем сохраняется в переменной caveNumber. Цикл while внутри функции chooseCave() гарантирует, что функция chooseCave() вернет только '1' или '2' в качестве возвращаемого значения. Таким образом, переменной caveNumber может быть присвоено только одно из этих двух значений.

Глобальная и локальная области видимости переменных

У переменных, которые создаются внутри функций, есть свои особенности, как и в случае с переменной cave в функции chooseCave() (строка 12).

```
12.     cave = ''
```

Локальная область видимости создается каждый раз, когда вызывается функция. Любые переменные, которые она создаст, будут существовать в локальной области. Представьте, что *область видимости* — это контейнер для переменных. Особенность локальных переменных в том, что они стираются из памяти, когда функция возвращается, и создаются заново при повторном вызове функции. Значение локальной переменной между вызовами функций не сохраняется.

Переменные, которые создаются вне функций, существуют в *глобальной области видимости*. Может быть только одна глобальная область, и она создается при запуске программы. Когда программа завершает работу, глобальная область уничтожается, и все переменные в ней стираются.

Переменная, которая существует в локальной области видимости, называется *локальной переменной*, а переменная, которая существует в глобальной области видимости, — соответственно, *глобальной переменной*. Переменная может быть или локальной, или глобальной; она не может быть и той и другой одновременно.

Переменная `cave` создается внутри функции `chooseCave()`. Это означает, что она создана в локальной области видимости функции `chooseCave()`. Она будет стерта при возврате функции `chooseCave()` и при повторном вызове функции `chooseCave()` будет создана заново.

Локальные и глобальные переменные могут иметь одинаковые имена, так как это разные переменные, находящиеся в разных областях видимости. Чтобы это продемонстрировать, давайте напишем новую программу.

```
  def bacon():
❶     spam = 99  # Создает локальную переменную с именем spam
❷     print(spam) # Выводит 99

❸ spam = 42    # Создает глобальную переменную с именем spam
❹ print(spam)   # Выводит 42
❺ bacon()     # Вызывает функцию bacon() и выводит 99
❻ print(spam)   # Выводит 42
```

Сначала мы создаем функцию с именем `bacon()`. Внутри функции `bacon()` мы создаем переменную `spam` и присваиваем ей значение `99` ❶. В пункте ❷ мы вызываем функцию `print()` для вывода значения этой локальной переменной `spam`, которое равно `99`. В пункте ❸ также присутствует глобальная переменная с именем `spam` с присвоенным ей значением `42`. Эта переменная глобальная, так как находится за пределами функции. Когда глобальная переменная `spam` вызывается в функции `print()` (❹), выводится значение `42`. Когда вызы-

вается функция bacon() в пункте ❺, выполняются пункты ❶ и ❷, создавая локальную переменную spam и присваивая, а затем выводя ее значение. Таким образом, вызов функции bacon() выводит значение переменной, равное 99. После возврата вызова функции bacon() локальная переменная spam уничтожается из памяти. Если мы выведем значение переменной spam в пункте ❻, то увидим значение глобальной переменной, равное 42. Если запустить этот код, получим следующее:

```
42
99
42
```

Позиция создания переменной определяет ее область видимости. Помните об этом при разработке своих программ.

Параметры функции

Следующая функция, определенная в программе «Царство драконов», называется checkCave().

```
19. def checkCave(chosenCave):
```

Обратите внимание на текст chosenCave в круглых скобках. Это *параметр* — локальная переменная, которую использует код функции. Когда вызывается функция, аргументы вызова — это значения, присвоенные параметрам. Давайте на мгновение вернемся к IDLE. Помните, что для некоторых вызовов функций, таких как str() или randint(), вы должны передать одно или несколько значений в круглых скобках.

```
>>> str(5)
'5'
>>> random.randint(1, 20)
14
>>> len('привет')
6
```

Этот пример содержит функцию Python, которую вы еще не встречали — len(). Функция len() возвращает количество символов в строке в виде целого

числа. В данном случае функция определила, что строка 'привет' содержит 6 символов.

Когда вы вызываете функцию checkCave(), вы также передаете аргумент. Этот аргумент сохраняется в виде новой переменной с именем chosenCave, которая является параметром функции checkCave().

Ниже представлена короткая программа, которая демонстрирует определение функции (sayHello) с параметром (name):

```
def sayHello(name):
    print('Привет, ' + name + '. Твое имя состоит из ' + str(len(name)) + ' букв.')
sayHello('Алиса')
sayHello('Борис')
spam = 'Катя'
sayHello(spam)
```

Когда вы вызовете функцию sayHello() с аргументом в круглых скобках, аргумент присваивается параметру name, и код в функции выполняется. В функции sayHello() есть только одна строка кода, которая является вызовом функции print(). Внутри вызова функции print() находится несколько строк и переменная name, наряду с вызовом функции len(). В этом примере функция len() используется для подсчета количества символов в имени. Если вы запустите программу, то получите следующий результат:

```
Привет, Алиса. Твое имя состоит из 5 букв.
Привет, Борис. Твое имя состоит из 5 букв.
Привет, Катя. Твое имя состоит из 4 букв.
```

В случае каждого вызова функции sayHello() выводится приветствие и длина аргумента name. Обратите внимание, что поскольку строка 'Катя' присвоена переменной spam, код sayHello(spam) равносилен коду sayHello('Катя').

Отображение результатов игры

Вернемся к исходному коду игры «Царство драконов».

```
20.     print('Вы приближаетесь к пещере...')
21.     time.sleep(2)
```

Модуль time содержит функцию sleep(), которая приостанавливает выполнение программы.

В строке 21 передается целое значение 2, поэтому функция time.sleep() приостановит выполнение программы на 2 секунды.

```
22.     print('Ее темнота заставляет вас дрожать от страха...')
23.     time.sleep(2)
```

Здесь программа выводит еще одну строку текста и ждет еще 2 секунды. Эти короткие паузы добавляют интриги в игру, что делает ее гораздо интереснее. В программе-шутнике из главы 4 вы вызывали функцию input() для приостановки программы, пока игрок не нажал клавишу **Enter**. Здесь игроку не требуется ничего делать, только подождать пару секунд.

```
24.     print('Большой дракон выпрыгивает перед вами! Он раскрывает свою пасть и...')
25.     print()
26.     time.sleep(2)
```

После небольшого ожидания программа определяет, в какой же пещере дружелюбный дракон.

Определение пещеры с дружелюбным драконом

В строке 28 вызывается функция randint(), которая в случайном порядке возвращает 1 или 2.

```
28.     friendlyCave = random.randint(1, 2)
```

Это целочисленное значение сохраняется в переменной friendlyCave и указывает на пещеру с дружелюбным драконом.

```
30.     if chosenCave == str(friendlyCave):
31.         print('...делится с вами своими сокровищами!')
```

Код в строке 30 проверяет, является ли выбранная игроком пещера в переменной chosenCave ('1' или '2') пещерой с дружелюбным драконом.

Но значение переменной friendlyCave целочисленное, потому что функция randint() возвращает целые числа. Вы не можете сравнивать строки и целые числа с оператором ==, потому что они *никогда* не будут равнозначны друг другу: '1' не равно 1, а '2' не равно 2.

Таким образом, значение переменной friendlyCave передается функции str(), которая возвращает строковое значение переменной friendlyCave. Теперь значения будут одного и того же типа данных и могут быть сопоставлены друг с другом. Как вариант, мы могли бы также преобразовать значение переменной chosenCave в целочисленное. Тогда код в строке 30 выглядел бы так:

```
if int(chosenCave) == friendlyCave:
```

Если значение переменной chosenCave равно значению переменной friendlyCave, условие определяется как истинное, а код в строке 31 сообщает игроку, что он выиграл сокровище.

Теперь нам следует добавить код, который будет выполняться в случае, если условие ложно. Код в строке 32 — это инструкция else.

```
32.     else:
33.         print('...моментально вас съедает!')
```

Инструкция else может находиться только после блока if. Блок else выполняется, когда условие инструкции if ложно. Это можно представить так: «Если это условие истинно, тогда выполнять блок if, а если ложно, то выполнить блок else».

В нашем случае инструкция else запускается, когда значение переменной chosenCave не равняется значению переменной friendlyCave. Тогда выполняется функция print() в строке 33 и сообщает игроку, что он съеден драконом.

Игровой цикл

В первой части программы определяется несколько функций, но не запускается код внутри них. Строка 35 — позиция, где начинается основная часть программы, так как это первая строка, в которой происходит какое-либо выполнение.

```
35. playAgain = 'да'
36. while playAgain == 'да' or playAgain == 'д':
```

С этой строки начинается основная часть программы. Предыдущие инструкции `def` просто определяли функции. Они не запускали код внутри этих функций. В строках 35 и 36 расположен цикл, в котором содержится оставшийся код игры. В конце игрок может сообщить программе, хочет ли он сыграть снова. Получив ответ 'да', интерпретатор переходит к циклу `while`, чтобы снова запустить всю игру с самого начала. В противном случае условие инструкции `while` будет ложным, интерпретатор перейдет в конец программы, и ее работа завершится.

При первом переходе к инструкции `while`, в строке 35, переменной `playAgain` присваивается значение `'да'`. Это означает, что при старте программы условие в строке 36 будет истинно. Таким образом мы обеспечиваем, что цикл `while` будет выполнен хотя бы один раз.

Вызов функций в программе

В строке 37 вызывается функция `displayIntro()`.

```
37.     displayIntro()
```

Это не встроенная функция Python, а определенная вами ранее, в строке 4. Когда эта функция вызывается, интерпретатор переходит к первой строке в функции `displayIntro()` в строке 5. Когда все строки кода в функции выполнены, интерпретатор переходит к строке 37 и продолжает движение вниз.

В строке 38 также вызывается определенная вами функция.

```
38.     caveNumber = chooseCave()
```

Помните, что функция `chooseCave()` позволяет игроку выбрать пещеру, в которую он хочет войти. Когда выполняется код `return cave` в строке 17, интерпретатор возвращается к строке 38. Затем вызов функции `selectCave()` вернет целочисленное значение номера пещеры, выбранной игроком. Это возвращаемое значение сохраняется в новой переменной с именем `caveNumber`.

Затем интерпретатор переходит к строке 39.

```
39.     checkCave(caveNumber)
```

В строке 39 вызывается функция `checkCave()` с переданным ей значением переменной `caveNumber` в качестве аргумента. Интерпретатор не только переходит к строке 20, но и копирует значение переменной `caveNumber` в параметр

chosenCave внутри функции checkCave(). Это функция, которая выводит текст: либо '...делится с вами своими сокровищами!', либо '...моментально вас съедает!', в зависимости от номера пещеры, который ввел игрок.

Запрос «сыграть снова»

Несмотря на то, победил игрок или проиграл, мы спрашиваем его, хочет ли он сыграть еще раз.

```
41.     print('Попытаете удачу еще раз? (да или нет)')
42.     playAgain = input()
```

Переменная playAgain сохраняет ввод игрока. Строка 42 — последняя строка в блоке while, поэтому интерпретатор возвращается к строке 36, чтобы проверить выполнение условия цикла while: playAgain == 'да' or playAgain == 'д'.

Если игрок вводит «да» или «д», интерпретатор начинает цикл сначала со строки 37.

Если игрок вводит «нет», «н» или что-то глупое вроде «Здесь был Вася», условие определяется как ложное, выполнение программы продолжается, и интерпретатор переходит к строке после блока while. Но так как после блока while нет никакого кода, программа завершает работу.

Одна особенность: строка 'ДА' не идентична строке 'да', так как компьютер не считает прописные и строчные буквы одинаковыми. Если игрок введет 'ДА', условие инструкции while будет определено как ложное, и программа все же завершит работу.

Позже, во время написания кода программы «Виселица», вы научитесь избегать этой проблемы. (См. раздел «Строковые методы lower() и upper()» в главе 8.)

Вы только что завершили свою вторую игру! В игре «Царство драконов» вы использовали многое из того, что уже выучили, когда создавали приложение «Угадай число», и взяли на заметку несколько новых трюков. Если что-то в этой программе осталось для вас непонятным, просмотрите еще раз каждую строку исходного кода и попробуйте изменить код, чтобы проанализировать, как в итоге меняется программа.

В главе 6 мы на время отложим игры. Вместо этого вы узнаете, как использовать *отладчик*.

Заключение

В игре «Царство драконов», вы создали свои собственные функции. Функция — это мини-программа внутри вашей программы. Код внутри функции

запускается при ее вызове. Используя функции, вы можете строить свой код более понятным образом.

Аргументы — это значения, копируемые в параметры функции при ее вызове. Вызов функции возвращает значение, которое она вычисляет.

Вы также узнали об областях видимости переменных. Переменные, созданные внутри функции, находятся в локальной области видимости, а переменные, созданные вне какой-либо функции, — в глобальной области видимости. Код из глобальной области видимости не может использовать локальные переменные. Если локальные переменные имеют такие же имена, что и глобальные, Python определяет их как разные переменные. Присвоение нового значения локальной переменной не влияет на значение глобальной переменной.

Области видимости переменных могут показаться сложным моментом программирования, но они полезны для организации функций как отдельных фрагментов кода программы.

Поскольку каждая функция имеет свою собственную локальную область видимости, вы можете быть уверены, что код одной функции не приведет к появлению ошибок в других функциях.

Так как функции полезны, их содержат почти все программы. Поняв принцип работы функций, вы можете избежать ручного набора огромного количества кода и упростите процесс исправления ошибок.

6

ИСПОЛЬЗОВАНИЕ ОТЛАДЧИКА

Если вы введете код с ошибками, компьютер не сможет правильно запустить вашу программу. Компьютерная программа всегда будет выполнять то, что вы ей скажете, хотя это иногда различается с тем, что вы *хотите* получить в результате. Ошибки в компьютерных программах называются *багами*. Баги возникают, если программист невнимательно подходит к тому, что именно программа должна выполнять.

В ЭТОЙ ГЛАВЕ РАССМАТРИВАЮТСЯ СЛЕДУЮЩИЕ ТЕМЫ:

- Три типа багов
- Отладчик IDLE
- Управление отладкой
- Навигация по коду
- Точки останова

Типы багов

Существуют три типа багов, с которыми вы можете столкнуться при написании кода:

- **Ошибки синтаксиса.** Этот тип багов возникает из-за опечаток. Интерпретатор определяет ошибку синтаксиса, когда ваш код написан не на корректном языке Python. Python не будет выполнять программу при наличии хотя бы одной ошибки синтаксиса.

- **Ошибки выполнения.** Это баги, которые возникают во время работы программы. Программа будет работать, пока не достигнет строки кода с ошибкой, после чего завершит работу и выдаст сообщение об ошибке (эта ситуация называется *сбоем* программы). В интерпретаторе Python будет отображаться *трассировка* — сообщение об ошибке, показывающее строку, в которой возникла проблема.
- **Семантические ошибки.** Эти баги — самые сложные для исправления. Они не приводят к сбою программы, но и не позволяют программе выполнять задуманное. Например, если программист хочет, чтобы переменная `total` была суммой значений переменных `a`, `b` и `c`, но записал `total = a * b * c`, тогда значение переменной `total` будет неправильным. Это может привести к сбою программы в будущем, но не сразу станет очевидным, где была допущена семантическая ошибка.

Поиск багов в программах может быть сложным занятием, если вы вообще заметите эти ошибки! При запуске программы вы можете обнаружить, что некоторые функции не вызываются, когда должны, либо они могут быть вызваны чрезмерное количество раз. Вы можете неправильно прописать условие для цикла `while` так, что оно будет выполняться неправильное количество раз. Вы можете написать цикл, из которого нет выхода, — семантическая ошибка, известная как *бесконечный цикл*. Чтобы остановить программу, застрявшую в бесконечном цикле, вы можете нажать в интерактивной оболочке сочетание клавиш **Ctrl+C**.

Для примера создайте бесконечный цикл, введя данный код в окне IDLE (не забудьте дважды нажать клавишу **Enter**, чтобы интерактивная оболочка определила, что вы закончили ввод в блоке `while`).

```
>>> while True:
        print('Нажмите Ctrl+C, чтобы остановить бесконечный цикл!!!')
```

Теперь нажмите и удерживайте сочетание клавиш **Ctrl+C**, чтобы остановить выполнение программы. Интерактивная оболочка будет выглядеть следующим образом:

```
Нажмите Ctrl+C, чтобы остановить бесконечный цикл!!!
Нажмите Ctrl+C, чтобы остановить бесконечный цикл!!!
Нажмите Ctrl+C, чтобы остановить бесконечный цикл!!!
Нажмите Ctrl+C, чтобы остановить бесконечный цикл!!!
Нажмите Ctrl+C, чтобы остановить бесконечный цикл!!!
Traceback (most recent call last):
```

```
    File "<pyshell#1>", line 2, in <module>
        print('Нажмите Ctrl+C, чтобы остановить бесконечный цикл!!!')
    File "C:\Program Files\Python 3.6\lib\idlelib\PyShell.py", line 1347, in
write
        return self.shell.write(s, self.tags)
KeyboardInterrupt
```

В нашем случае цикл `while` всегда истинен, поэтому программа будет продолжать выводить одну и ту же строку целую вечность, пока пользователь не прекратит выполнение. В данном примере мы нажали сочетание клавиш **Ctrl+C**, чтобы остановить бесконечный цикл после того, как цикл `while` выполнился 5 раз.

Отладка

Бывает трудно понять источник ошибки, так как строки кода выполняются быстро, а значения переменных изменяются часто. *Отладчик* — это программа, которая позволяет вам выполнять код по одной строке за раз в том же порядке, что и Python выполняет каждую инструкцию. Отладчик также показывает вам, какие значения хранятся в переменных на каждом шаге.

Запуск отладчика

Откройте файл игры «Царство драконов», которую вы сделали в главе 5, в IDLE. Выберите команду меню **Run** ⇒ **Python Shell** (Запустить ⇒ Оболочка Python), чтобы открыть окно интерактивной оболочки Python. В открывшемся окне выберите команду меню **Debug** ⇒ **Debugger** (Отладка ⇒ Отладчик), чтобы отобразить окно управления отладкой, показанное на рис. 6.1. Установите в нем флажки **Stack** (Стек), **Locals** (Локальные), **Source** (Код) и **Globals** (Глобальные).

Теперь, когда отладчик запущен, запустите игру «Царство драконов», нажав в интерактивной оболочке клавишу **F5**. Окно **Debug Control** (Управление отладкой) будет выглядеть примерно так, как показано на рис. 6.2.

Вы произвели так называемый запуск программы *под отладчиком*. Когда вы запускаете программу Python под отладчиком, программа приостанавливает выполнение перед первой командой. Если вы перейдете к окну файла в IDLE, то увидите, что первая команда выделена серым цветом (при условии, что в диалоговом окне **Debug Control** (Управление отладкой) установлен флажок **Source** (Код)). В диалоговом окне **Debug Control** (Управление отладкой) будет указано, что выполняется код в строке 1, в которой импортируется модуль `random` (рис. 6.2).

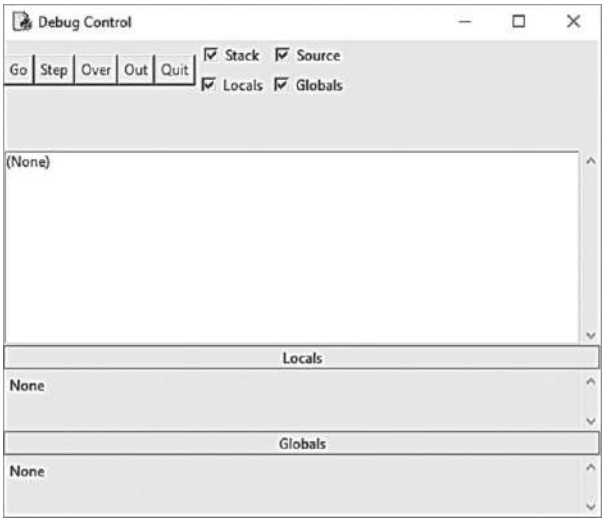


Рис. 6.1. Окно **Debug Control**

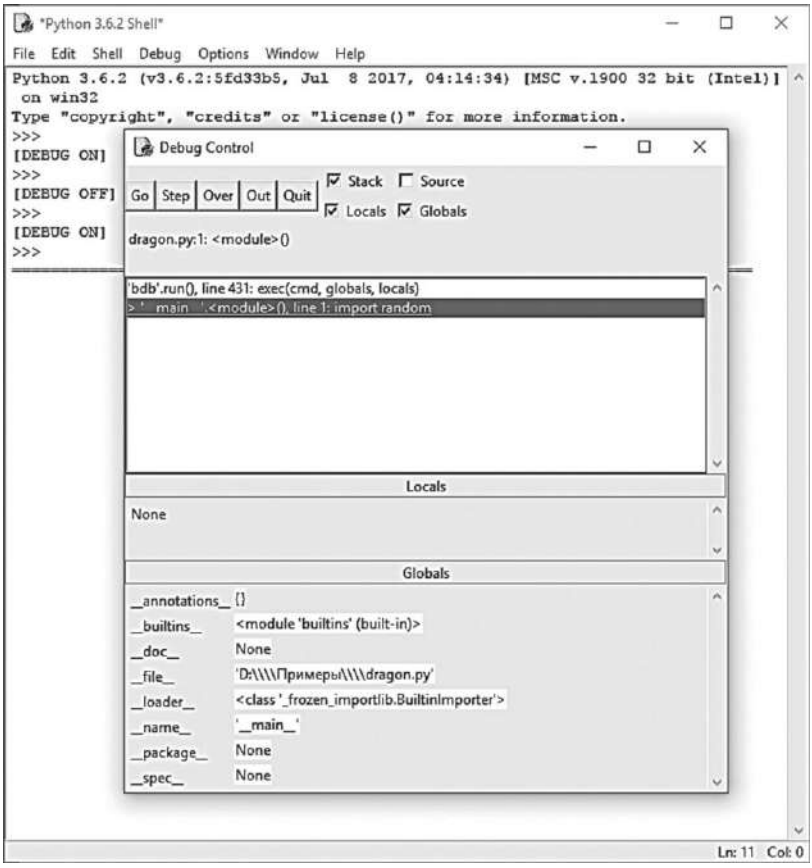


Рис. 6.2. Запуск игры «Царство драконов» под отладчиком

Пошаговое выполнение программы с помощью отладчика

Отладчик позволяет выполнять инструкции по очереди — этот процесс называется *пошаговым выполнением*. Выполните одну команду, нажав кнопку **Step** (Шаг) в окне **Debug Control** (Управление отладкой). Интерпретатор Python выполнит инструкцию `import random`, а затем остановится, прежде чем начнет выполнять следующую. Окно **Debug Control** (Управление отладкой) отобразит, как будет выглядеть выполненная строка при нажатии кнопки **Step** (Шаг). На данном этапе интерпретатор должен находиться на строке 2 — `import time`. Нажмите кнопку **Quit** (Выход), чтобы завершить программу.

Это краткое описание того, что происходит, когда вы нажимаете кнопку **Step** (Шаг) при запуске игры «Царство драконов» под отладчиком. Нажмите клавишу **F5**, чтобы снова запустить игру «Царство драконов», а затем следуйте этим инструкциям:

1. Дважды нажмите кнопку **Step** (Шаг), чтобы выполнить две инструкции импорта.
2. Нажмите кнопку **Step** (Шаг) еще три раза, чтобы выполнить три инструкции `def`.
3. Нажмите кнопку **Step** (Шаг) еще раз, чтобы определить переменную `playAgain`.
4. Нажмите кнопку **Go** (Вперед), чтобы запустить оставшуюся часть программы, или кнопку **Quit** (Выход), чтобы завершить работу.

Отладчик пропустил строку 3, потому что она пуста. Обратите внимание, что с отладчиком вы можете только делать шаг вперед; вы не можете возвращаться по коду назад.

Область глобальных переменных

Область **Globals** (Глобальные) в диалоговом окне **Debug Control** (Управление отладкой) — это место, в котором отражены все глобальные переменные. Помните, глобальными называются переменные, созданные вне каких-либо функций (то есть в глобальной области видимости).

Текст, рядом с именами функций в области **Globals** (Глобальные) выглядит так: `<function checkCave at 0x012859B0>`. Имена модулей также содержат путающий текст рядом с ними, например `<module 'random' from 'C:\\...thon36\\\\lib\\\\random.py'>`. Для отладки программы вам необязательно знать, что все это означает. Просто увидев, находятся ли функции и модули в области **Globals** (Глобальные), вы поймете, определена ли так или иная функция или импортирован ли модуль.

Также вы можете не обращать внимание настройки типа `__builtins__`, `__doc__`, `__name__` и прочие в области **Globals** (Глобальные). (Это переменные, которые используются в каждой программе Python.)

В программе «Царство драконов» в окне **Debug Control** (Управление отладкой) в области **Globals** (Глобальные) появятся три инструкции `def`, которые выполняют и определяют функции. Когда будет создана переменная `playAgain`, она также отобразится в области **Globals** (Глобальные). Рядом с именем переменной будет указана строка `'да'`. Окно **Debug Control** (Управление отладкой) позволяет видеть значения всех переменных в программе по мере запуска программы. Это помогает при исправлении багов.

Область локальных переменных

Вдобавок к области **Globals** (Глобальные) есть область **Locals** (Локальные), которая отображает локальные переменные и их значения. Переменные будут видны в области **Locals** (Локальные) только в том случае, если интерпретатор находится внутри той или иной функции. Когда интерпретатор находится в глобальной области видимости, область **Locals** (Локальные) пуста.

Обычное выполнение и завершение работы

Если вы устали от постоянного нажимания кнопки **Step** (Шаг) и хотите, чтобы программа была запущена в обычном режиме, нажмите кнопку **Go** (Вперед) в верхней части окна **Debug Control** (Управление отладкой). Программа будет выполнена в обычном режиме вместо пошагового выполнения.

Чтобы полностью завершить работу программы, нажмите кнопку **Quit** (Выход) в верхней части окна **Debug Control** (Управление отладкой). Программа тут же завершит работу. Это удобно, если вам необходимо начать отладку заново с самого начала.

Навигация по коду

Запустите программу «Царство драконов» под отладчиком. Продолжайте пошаговое выполнение, пока не достигнете строки 37. Как показано на рис. 6.3, это строка с функцией `displayIntro()`. Когда вы снова нажмете кнопку **Step** (Шаг), отладчик начнет выполнение функции и окажется в строке 5 — первой строке функции `displayIntro()`. Этот так называемый *шаг с входом*.

Когда выполнение приостановилось в строке 5, вам может понадобиться прекратить пошаговую отладку. Если вы нажмете кнопку **Step** (Шаг) еще раз, отладчик войдет в функцию `print()`. Она относится к встроенным функциям Python, поэтому нецелесообразно пропускать ее через отладчик. Функции Python, такие как `print()`, `input()`, `str()` и `randint()`, уже тщательно проверены на отсутствие багов. Можно предположить, что эти фрагменты кода не содержат багов.

Вы же не хотите терять время, проверяя каждую функцию `print()`? Поэтому вместо нажатия кнопки **Step** (Шаг), которая приведет вас к проверке кода функции `print()`, нажмите кнопку **Over** (Шаг с обходом). Она позволит *обойти* код внутри функции `print()`. Код внутри функции `print()` будет выполнен

в обычном режиме, а затем отладчик приостановит работу, когда выполнение функции `print()` будет завершено.

Рис. 6.3. Пошаговая проверка кода программы, строка 37

Обход — удобный способ пропустить код внутри функции. Отладчик теперь будет приостановлен на строке 38 — `caveNumber = chooseCave()`.

Нажмите кнопку **Step** (Шаг) снова, чтобы перейти к коду внутри функции `chooseCave()`. Продолжайте пошаговую проверку кода, пока не достигнете строки 15, отвечающей за вызов функции `input()`. Программа будет ждать, пока вы введете ответ в интерактивной оболочке, точно так же, как при обычном выполнении программы. Если вы сейчас попытаетесь нажать кнопку **Step** (Шаг), то ничего не произойдет, потому что программа ждет ответа с клавиатуры.

Вернитесь обратно к IDLE и введите номер пещеры. Прежде чем вы сможете ввести число, в нижней части интерактивной оболочки должен появиться мигающий курсор. Иначе текст, который вы введете, не появится.

Как только вы нажмете клавишу **Enter**, отладчик продолжит пошаговую проверку кода.

Теперь нажмите кнопку **Out** (Шаг с выходом) в окне **Debug Control** (Управление отладкой). Это так называемый *шаг с выходом*, так как эта кнопка инструктирует отладчик «перешагнуть» столько строк, сколько потребуется, пока интерпретатор не вернется из функции, в которой находится на данный момент. После выхода из функции интерпретатор перейдет к первой строке, расположенной после текущей функции.

Если интерпретатор находится вне функции, то нажатие кнопки **Out** (Шаг с выходом) приведет к тому, что отладчик выполнит все остальные строки в программе. Это действие аналогично нажатию кнопки **Go** (Вперед). Ниже представлено краткое описание каждой кнопки:

- **Go** (Вперед) выполняет оставшуюся часть кода в обычном режиме до конца, или пока не достигнет точки останова (см. раздел «Установка точек останова» далее в этой главе).
- **Step** (Шаг) выполняет одну инструкцию или один шаг. Если строка содержит вызов функции, отладчик перейдет внутрь функции.
- **Over** (Шаг с обходом) выполняет одну инструкцию или один шаг. Если код в строке содержит вызов функции, отладчик не *перейдет в* нее, но *пропустит* этот вызов.
- **Out** (Шаг с выходом) продолжает переходить по строкам кода, пока отладчик не покинет текущую функцию. Это *выход* из функции.
- **Quit** (Выход) немедленно завершает программу.

Теперь, когда вы умеете использовать отладчик, давайте попробуем найти баги в коде программы.

Поиск багов

Отладчик может помочь вам найти источник ошибки в вашей программе. Ниже, в качестве примера, представлен код программы с ошибкой. Программа по сложению случайных чисел содержит баг, который следует найти.

В редакторе файлов создайте новый файл, выбрав команду меню **File** ⇒ **New File** (Файл ⇒ Новый файл). Введите следующий код в открывшееся окно и сохраните файл под именем *buggy.py*.

buggy.py

```
1. import random
2. number1 = random.randint(1, 10)
3. number2 = random.randint(1, 10)
4. print('Сколько будет: ' + str(number1) + ' + ' + str(number2) + '?')
```

```
5. answer = input()
6. if answer == number1 + number2:
7.     print('Верно!')
8. else:
9.     print('Нет! Правильный ответ - ' + str(number1 + number2))
```

Введите код программы именно так, как показано выше, даже если вы уже увидели ошибку. Далее нажмите клавишу **F5** и запустите программу. Вот как она могла бы выглядеть при запуске:

```
Сколько будет: 9 + 9?
18
Нет! Правильный ответ - 18
```

Вот и баг! Сбоя в программе нет, но она и не работает должным образом.

Программа сообщает пользователю, что он ввел неправильное число, даже если он дал правильный ответ.

Запуск программы под отладчиком позволит найти причину ошибки.

В верхней части окна интерактивной оболочки нажмите **Debug** ⇒ **Debugger** (Отладка ⇒ Отладчик) для вывода на экран диалогового окна **Debug Control** (Управление отладкой). (Убедитесь что в окне установлены флажки **Stack** (Стек), **Locals** (Локальные), **Source** (Код) и **Globals** (Глобальные).) Далее нажмите клавишу **F5** в редакторе файлов для запуска программы. Теперь программа запустится под отладчиком.

Отладчик начнет со строки `import random`.

```
1. import random
```

Здесь нет ничего особенного, поэтому просто нажмите кнопку **Step** (Шаг), чтобы выполнить указанную команду. Вы увидите, что модуль `random` добавлен в область **Globals** (Глобальные). Снова нажмите кнопку **Step** (Шаг), чтобы выполнить инструкцию в строке 2.

```
2. number1 = random.randint(1, 10)
```

Появится новое окно редактора файлов с файлом *random.py*. Вы перешли к коду внутри функции `randint()` модуля `random`. Вы ведь помните, что

встроенные функции Python не станут причиной ошибки, поэтому нажмите кнопку **Out** (Шаг с выходом), чтобы выйти из функции `randint()` и вернутся к нашей программе. Затем закройте окно *random.py*. В следующий раз вы можете нажать кнопку **Over** (Шаг с обходом), чтобы обойти функцию `randint()` вместо того, чтобы заходить в нее.

Код в строке 3 также содержит вызов функции `randint()`.

```
3. number2 = random.randint(1, 10)
```

Обойдите ее, нажав кнопку **Over** (Шаг с обходом).

Код в строке 4 содержит вызов функции `print()` для отображения игроку случайных чисел.

```
4. print('Сколько будет: ' + str(number1) + ' + ' + str(number2) + '?')
```

Вы знаете, какие числа программа выведет, еще до того, как она их напечатает! Взгляните на область **Globals** (Глобальные) в диалоговом окне **Debug Control** (Управление отладкой). Вы увидите переменные `number1` и `number2`, а рядом с ними целочисленные значения, которые они содержат.

Переменная `number1` содержит значение `2`, а переменная `number2` — значение `5`. (Ваши случайные числа, скорее всего, будут отличаться.) Когда вы нажмете кнопку **Step** (Шаг), функция `str()` конкатенирует строковые версии этих целых чисел, и программа отобразит строку в вызове `print()` с этими значениями. Окно **Debug Control** (Управление отладкой) при этом выглядит так, как показано на рис. 6.4.

Нажмите клавишу **Step** (Шаг) в строке 5, чтобы выполнить функцию `input()`.

```
5. answer = input()
```

Отладчик ожидает, пока игрок введет ответ в программу. Введите корректный ответ (в моем случае `7`) в окне IDLE.

Отладчик сохранит ответ и перейдет далее к строке 6.

```
6. if answer == number1 + number2:
7.     print('Верно!')
```

Рис. 6.4. Переменной `number1` присвоено значение `2`, а `number2` — значение `5`

Код в строке 6 — это инструкция `if`. Условие заключается в том, что значение в ответе должно совпадать с суммой чисел `number1` и `number2`. Если условие истинно, отладчик перейдет к строке 7.

Если ложно — к строке 9. Нажмите клавишу **Step** (Шаг) еще раз, чтобы выяснить, куда он перейдет.

```
8. else:
9.     print('Нет! Правильный ответ - ' + str(number1 + number2))
```

Отладчик в строке 9! Что происходит? Должно быть, условие в инструкции `if` ложно. Взгляните на значения переменных `number1` и `number2` и на ответ. Обратите внимание, что значения переменных `number1` и `number2` — целые числа, поэтому их сумма также должна быть целым числом. Но ответ — это строка.

Это означает, что выражение `answer == number1 + number2` будет вычисляться по формуле `'7' == 7`.

Строковое значение и значение целого числа никогда не будут равны друг другу, поэтому условие ложно.

Это и есть ошибка в коде: в нем используется значение `answer`, в то время когда следует использовать `int(answer)`. Измените строку 6 на `if int(answer) == number1 + number2` и запустите программу снова.

```
Сколько будет: 2 + 5?
7
Верно!
```

Теперь программа работает правильно. Запустите ее еще раз и намеренно введите неправильный ответ.

Вы отладили программу! Помните, компьютер будет выполнять ваши программы в точности так, как вы их напишете, даже если вы написали совсем не то, что намеревались.

Установка точек останова

Пошаговая отладка кода, строка за строкой, может занять слишком много времени. Часто вам бы хотелось, чтобы программа выполнялась в обычном режиме, пока не достигнет определенной строки. Вы можете установить *точку останова* на строке, с которой хотите начать отладку. Если вы предполагаете, что проблема в вашем коде, скажем, в строке 17, просто установите точку останова на этой строке (либо на пару строк выше).

Когда интерпретатор достигнет этой строки, программа перейдет в отладчик. Тогда вы сможете выполнять код построчно, чтобы найти ошибку. Нажатие кнопки **Go** (Вперед) запустит выполнение программы в обычном режиме, пока не будет достигнута следующая точка останова или конец программы.

Чтобы установить точку останова в операционной системе Windows, щелкните правой кнопкой мыши по строке в редакторе файлов и выберите в появившемся меню команду **Set Breakpoint** (Установить точку останова). В операционной системе macOS щелкните по строке мышью, удерживая клавишу **Ctrl**, чтобы перейти в меню, и выберите команду **Set Breakpoint** (Установить точку останова). Вы можете установить столько точек останова, сколько захотите. Редактор файлов окрасит строки с точками останова в желтый цвет.

На рис. 6.5 приведен пример кода с установленными точками останова.

Чтобы удалить точку останова, щелкните мышью по строке и выберите пункт **Clear Breakpoint** (Очистить точку останова) в появившемся меню.

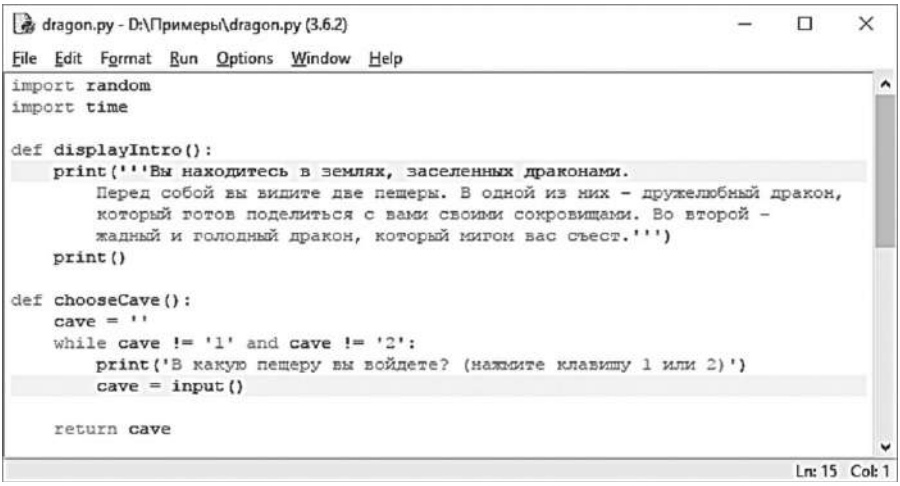


Рис. 6.5. Код с двумя точками останова

Использование точек останова

Далее мы рассмотрим на программу, которая вызывает функцию `random.randint(0, 1)`, имитирующую подкидывание монетки. Если функция вернет целое число `1` — это будет «орел», а если `0` — «решка». Переменная `flips` будет подсчитывать, сколько раз «подбрасывалась монетка». Переменная `heads` будет подсчитывать, сколько раз «выпал» «орел».

Программа «подбросит монетку» 1000 раз. Человеку для этого понадобится более часа, но компьютер сможет сделать это за секунду! В этой программе нет ошибок, но отладчик позволит нам взглянуть на то, как эта программа работает. Введите указанный ниже код в редактор файлов и сохраните в файл под именем *coinFlips.py*. Если у вас возникают ошибки после ввода этого кода, сравните код, который вы ввели, с исходным кодом на сайте **inventwithpython.com/diff/**.

coinFlips.py

```
1. import random
2. print('Я подброшу монетку 1000 раз. Угадай, сколько раз выпадет "Орел"? (Нажми клавишу Enter,
чтобы начать)')
3. input()
4. flips = 0
5. heads = 0
6. while flips < 1000:
7.     if random.randint(0, 1) == 1:
8.         heads = heads + 1
```

```
9.     flips = flips + 1
10.
11.     if flips == 900:
12.         print('900 подкидываний и "Орел" выпал ' + str(heads) + ' раз.')
13.     if flips == 100:
14.         print('При 100 бросках, "Орел" выпал ' + str(heads) + ' раз.')
15.     if flips == 500:
16.         print('Полпути пройдено и "Орел" выпал  ' + str(heads) + ' раз.')
17.
18. print()
19. print('Из 1000 подбрасываний монетки "Орел" выпал ' + str(heads) + ' раз!')
20. print('Насколько вы близки к правильному ответу?')
```

Программа работает довольно быстро. Больше времени занимает ожидание, пока пользователь нажмет клавишу **Enter**, чем виртуальное подбрасывание монетки. Давайте предположим, что вы бы хотели увидеть как программа «подбрасывает монетку» раз за разом. В окне интерактивной оболочки выберите команду меню **Debug** ⇒ **Debugger** (Отладка ⇒ Отладчик), чтобы открыть диалоговое окно **Debug Control** (Управление отладкой). Затем нажмите клавишу **F5**, чтобы запустить программу.

Программа в отладчике начнется со строки 1. Нажмите кнопку **Step** (Шаг) в окне **Debug Control** (Управление отладкой) три раза для выполнения первых трех строк кода (это строки 1, 2 и 3). Вы заметите, что кнопки становятся неактивными, потому что вызывается функция `input()`, и интерактивная оболочка ждет, когда пользователь что-то введет. Щелкните мышью по окну IDLE и нажмите клавишу **Enter**. (Убедитесь, что нажали клавишу в окне IDLE; в противном случае нажатие может быть не распознано.)

Вы можете нажать кнопку **Step** (Шаг) еще несколько раз, но вскоре поймете, что для обхода всей программы вам понадобится достаточно много времени. Вместо этого установите точки останова на строках 12, 14 и 16, чтобы отладчик остановился, когда значение переменной `flips` (число «подкидываний») будет равняться `900`, `100` и `500`, соответственно. Эти строки будут выделены, как показано на рис. 6.6.

После установки точек останова нажмите кнопку **Go** (Вперед) в диалоговом окне **Debug Control** (Управление отладкой). Программа будет выполняться в обычном режиме, пока не достигнет следующей точки останова. Когда значение переменной `flip` станет равно `100`, условие для инструкции `if` в строке 13 станет истинным. Это приведет к тому, что начнет выполняться код в строке 14 (где установлена точка останова), а значит, отладчик остановит программу и возьмет выполнение программы на себя. Взгляните на об-

ласть **Globals** (Глобальные) в окне отладчика, чтобы увидеть значения переменных `flips` («подкидывания») и `heads` («орел»).

```
coinFlips.py - D:\Примеры\coinFlips.py (3.6.2)
File  Edit  Format  Run  Options  Window  Help
import random
print('Я подброшу монетку 1000 раз. Угадай, сколько раз выпадет "Орел"? (Нажми к
input()
flips = 0
heads = 0
while flips < 1000:
    if random.randint(0, 1) == 1:
        heads = heads + 1
    flips = flips + 1

    if flips == 900:
        print('900 подкидываний и "Орел" выпал ' + str(heads) + ' раз.')
    if flips == 100:
        print('При 100 бросках, "Орел" выпал ' + str(heads) + ' раз.')
    if flips == 500:
        print('Полпути пройдено и "Орел" выпал  ' + str(heads) + ' раз.')

print()
print('Из 1000 подбрасываний монетки "Орел" выпал ' + str(heads) + ' раз!')
print('Насколько вы близки к правильному ответу?')
Ln: 16  Col: 1
```

Рис. 6.6. Установлены три точки останова в коде файла *coinflips.py*

Нажмите кнопку **Go** (Вперед) еще раз и программа продолжит выполняться, пока интерпретатор не достигнет следующей точки останова в строке 16. Снова взгляните, как изменились значения переменных `flips` и `heads`.

Нажмите кнопку **Go** (Вперед) вновь, чтобы продолжить выполнение до следующей точки останова в строке 12.

Заключение

Написание программы — это только половина работы. Затем необходимо удостовериться, что ваша программа работает должным образом. Отладчик помогает выполнить код пошагово, строка за строкой. Вы можете проверить, какие строки выполняются в каком порядке, и какие значения содержат переменные.

Если пошаговая проверка программы занимает слишком много времени, вы можете установить точки останова, чтобы отладчик останавливался только на нужных строках кода.

Использование отладчика — прекрасный способ понять, что делает программа. Так как эта книга содержит объяснения кода всех приведенных игр, отладчик поможет получить дополнительные сведения о ваших собственных программах.

7

ПРОЕКТИРОВАНИЕ ИГРЫ «ВИСЕЛИЦА» С ПОМОЩЬЮ БЛОК-СХЕМ

В этой главе вы спроектируете игру «Виселица». Эта игра существенно сложнее, чем предыдущие игры, но и более интересная. Из-за сложности игры мы сначала тщательно ее спланируем, построив в этой главе блок-схему. К написанию кода игры «Виселица» мы приступим в 8 главе.

В ЭТОЙ ГЛАВЕ РАССМАТРИВАЮТСЯ СЛЕДУЮЩИЕ ТЕМЫ:

- ASCII-графика
- Проектирование игры с помощью блок-схем

Правила игры «Виселица»

Виселица — это игра для двоих, в которой один игрок загадывает слово и рисует на странице отдельные пустые клетки для каждой буквы. А второй игрок пытается угадать буквы, которые могут быть в данном слове, а затем и все слово целиком.

Если второй игрок правильно угадывает букву, первый игрок вписывает ее в соответствующую пустую клетку. А если ошибается, первый игрок рисует одну из частей тела повешенного человечка. Чтобы победить, второй игрок должен угадать все буквы в слове до того, как повешенный человечек будет полностью нарисован.

Пример запуска игры «Виселица»

Ниже представлен пример, что будет видеть пользователь, играя в написанную вами в главе 8 игру «Виселица». Текст, который вводит игрок, выделен полужирным шрифтом.

```
В И С Е Л И Ц А

  +---+
      |
      |
      |
     ===

Ошибочные буквы:

_ _ _ _
Введите букву.
а

  +---+
      |
      |
      |
     ===

Ошибочные буквы:

_ _ _ а
Введите букву.
и

  +---+
  O   |
      |
      |
     ===

Ошибочные буквы: и

_ _ _ а
Введите букву.
о

  +---+
```

```
  O   |
  |   |
      |
     ===

Ошибочные буквы: и о
_ _ _ а
Введите букву.
у

  +---+
  O   |
  |   |
      |
     ===

Ошибочные буквы: и о
у _ _ а
Введите букву.
т

  +---+
  O   |
  |   |
      |
     ===

Ошибочные буквы: и о
у т _ а
Введите букву.
к
ДА! Секретное слово - "утка"! Вы угадали!
Хотите сыграть еще? (да или нет)
нет
```

ASCII-графика

Графика для игры «Виселица» — это символы клавиатуры, напечатанные на экране. Такой вид графики называется *ASCII-графика* (произносится как аски). Она была предшественником популярных сейчас «Эмодзи». Ниже показан кот, нарисованный при помощи ASCII-графики:

```
            _________________________
         __/xxx     xx     xxx    xx\
        /xxxxxxx     xx     xx    xx \_
       /       xx                    xx\
      /             /\                x  \
      |   /\       /  \___                 \_
      |  /  \____/        \_            Z    \
      | /                   \      z          \
       /               ___/  |  z              |
    __|       \___            |              xxx|
   /  |             ___  ___-----         _   x|
  /   |            |   |  ______      ___/    |
  |   \   ------   \_/      /________/ xx     /
  \      \_   ----        __/                 /
   \       \____________/                    /
    \_________________________________________/
```

ASCII-графика для игры «Виселица» будет выглядеть так:

```
+---+   +---+   +---+   +---+   +---+   +---+   +---+
    |   O   |   O   |   O   |   O   |   O   |   O   |
    |       |   |   |  /|   |  /|\   |  /|\   |  /|\   |
    |       |       |       |       |  /    |  / \   |
   ===     ===     ===     ===     ===     ===     ===
```

Проектирование игры с помощью блок-схем

Эта игра немного сложнее, чем те, которые вы видели до сих пор, поэтому давайте подумаем об этапах ее программирования. Для начала вы создаете блок-схему (типа той, что мы рисовали для игры «Царство драконов» на рис. 5.1 в главе 5), чтобы наглядно представить то, что будет делать программа.

Как упоминалось в главе 5, блок-схема представляет собой диаграмму, отражающую серию шагов в виде полей (блоков), связанных друг с другом стрелками. Каждое поле — это шаг, а стрелки показывают возможные следующие шаги. Поместите палец на поле «Старт» блок-схемы и проследите за ходом программы, передвигаясь по стрелкам на другие поля, пока не дойдете до поля «Конец». Вы можете двигаться от поля к полю только по стрелкам. Вы не можете двигаться назад до тех пор, пока стрелка вас туда не перенесет,

как в случае с полем «Игрок уже называл эту букву». На рис. 7.1 показана полная блок-схема игры «Виселица».

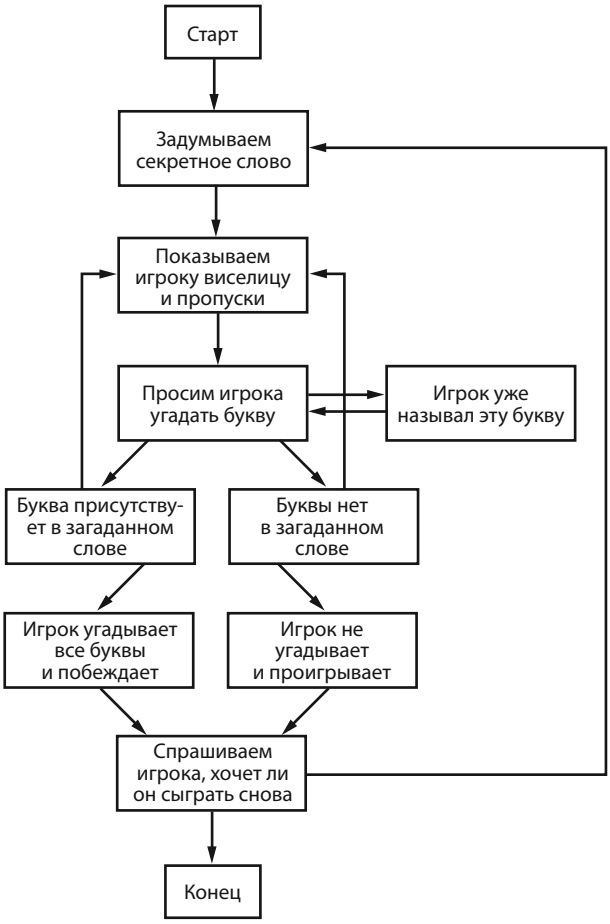


Рис. 7.1. Полная блок-схема игры «Виселица»

Конечно же, вы *не обязаны* составлять блок-схему; вы можете сразу начать писать код. Но часто бывает так, что, как только вы начинаете писать программу, вам на ум приходят вещи, которые вы хотите добавить или изменить. Все может закончиться тем, что вам придется удалить значительный фрагмент вашего кода, что будет означать трату времени. Самый лучший способ этого избежать — спланировать, как должна работать программа до того, как начинать писать ее код.

Создание блок-схем

Ваша блок-схема необязательно должна выглядеть так, как показано на рис. 7.1. Главное, чтобы вы понимали свою блок-схему, — это вам поможет, когда вы начнете писать код. Вы можете спроектировать свою блок-схему, на-

чав с полей «Старт» и «Конец», как показано на рис. 7.2. Теперь подумайте о том, что происходит, когда вы играете в игру «Виселица». Для начала компьютер задумывает секретное слово. Затем игрок угадывает буквы. Добавьте поля для этих действий, как показано на рис. 7.3. Новые поля в каждой блок-схеме обведены пунктиром.

Рис. 7.2. Начинаем проектировать блок-схему с полей «Старт» и «Конец»

Рис. 7.3. Добавляем дополнительные два поля с описанием шагов в игре «Виселица»

Но игра не заканчивается после того, как игрок угадывает букву. Программе необходимо проверить, содержит ли секретное слово данную букву.

Ветвление в блок-схемах

Существуют два сценария: буква присутствует в слове, или ее в нем нет. Вы добавляете два новых блока в схему — по одному для каждого из сценариев развития событий. Так создается разветвление в блок-схеме, как показано на рис. 7.4.

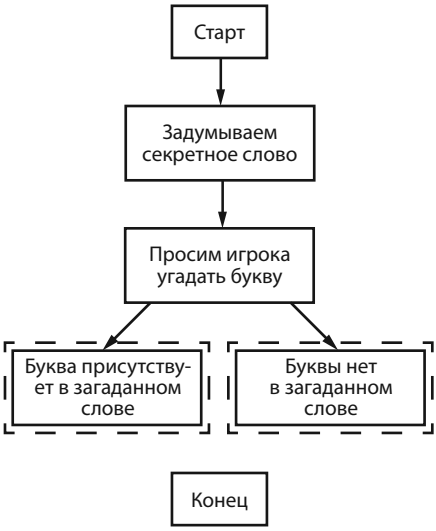


Рис. 7.4. Разветвление на две отдельные ветви

Если буква присутствует в секретном слове, следует проверить, возможно, игрок угадал все буквы и выиграл. Если буквы в слове нет, нужно проверить, возможно, повешенный человек нарисован полностью, и игрок проиграл. Добавьте поля для этих действий.

Блок-схема теперь выглядит так, как показано на рис. 7.5.

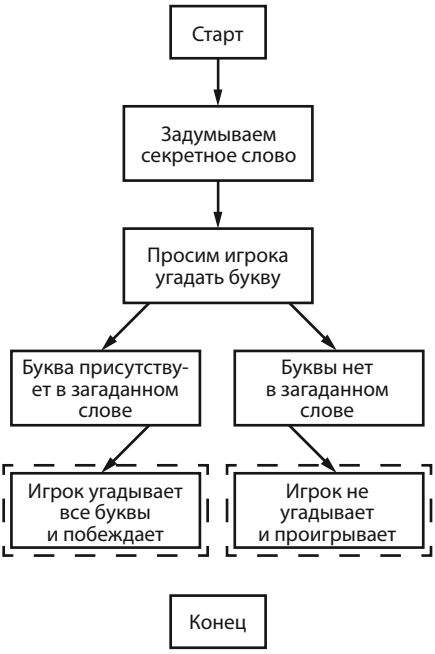


Рис. 7.5. После разветвления шаги продолжаются в разных направлениях

Нет необходимости рисовать стрелку из поля «Буква присутствует в загаданном слове» к полю «Игрок не угадывает и проигрывает», так как невозможно проиграть, правильно угадав букву. Как и невозможно выиграть, не угадав букву, поэтому и для этого действия рисовать стрелку не нужно.

Заканчиваем или начинаем игру сначала

Когда игрок выиграл или проиграл, спрашиваем его, хочет ли он сыграть заново, чтобы отгадать другое слово. Если игрок не хочет играть, программа завершает работу; в противном случае программа продолжает выполнение и загадывает новое секретное слово. Это показано на рис. 7.6.

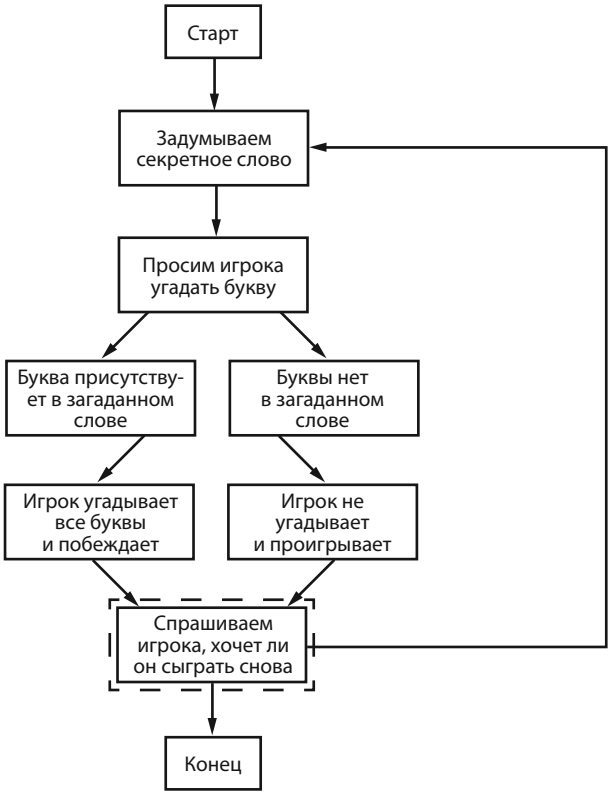


Рис. 7.6. Ветки блок-схемы с запросом, хочет ли игрок сыграть снова

Следующая попытка

Теперь блок-схема по большей части завершена, но нам все еще кое-чего недостает. Во-первых, игрок не угадывает букву только один раз; он продолжает угадывать буквы до тех пор, пока не выиграет или не проиграет. Дорисуйте две новые стрелки, как показано на рис. 7.7.

А как быть, если игрок назовет ту же самую букву снова? Вместо того чтобы проверять одну и ту же букву повторно, позвольте игроку загадать другую букву. Это новое поле показано на рис. 7.8.

Если игрок дважды называет одну и ту же букву, блок-схема возвращается к полю «Просим игрока угадать букву».

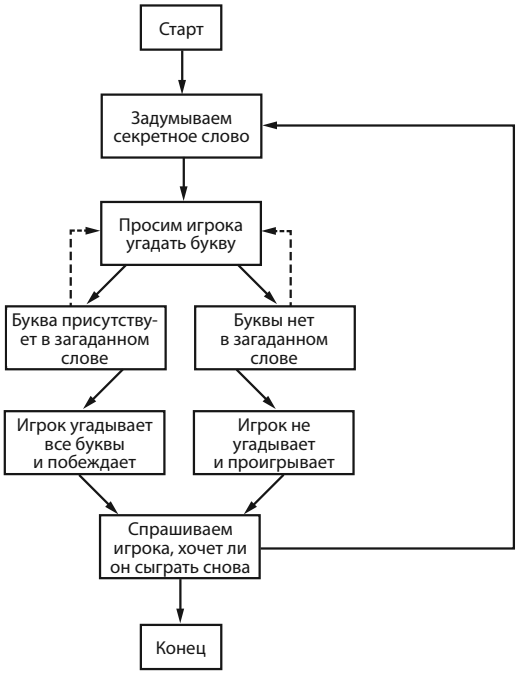


Рис. 7.7. Пунктирные линии показывают, что игрок может угадывать снова

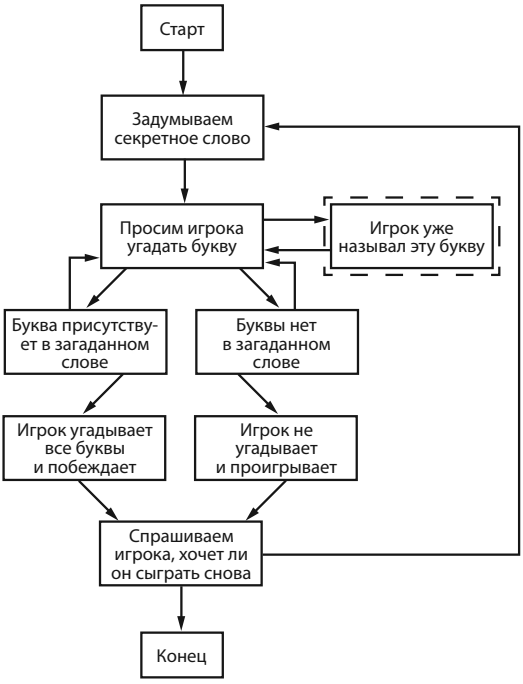


Рис. 7.8. Добавляем шаг на тот случай, если игрок назвал букву, которую уже произносил ранее

Обратная связь с игроком

Игроку необходимо знать, как он справляется. Программа должна отобразить рисунок повешенного человека и секретное слово (с пропусками вместо букв, которые игрок еще не угадал). Этот интерфейс позволит игроку понять, насколько он близок к победе или к провалу.

Эта информация обновляется каждый раз, когда игрок пытается угадать букву. Добавьте блок «Показываем игроку виселицу и пропуски» в схему между блоками «Задумываем секретное слово» и «Просим игрока угадать букву», как показано на рис. 7.9.

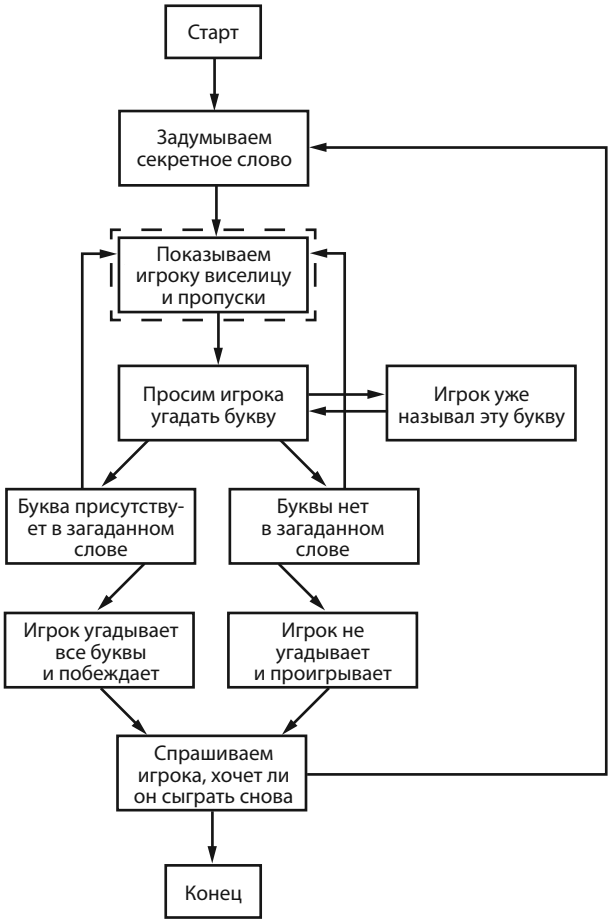


Рис. 7.9. Добавляем блок «Показываем игроку виселицу и пропуски» для обратной связи с игроком

Выглядит неплохо! Эта блок-схема полностью описывает порядок игрового процесса приложения «Виселица». Блок-схема поможет вам не упустить ничего из того, что следует учесть при разработке игры.

Заключение

Вам может показаться скучным рисование блок-схем перед написанием программы. В конце концов, люди желают играть в игры, а не рассматривать блок-схемы! Но гораздо легче вносить изменения и выявлять проблемы, предполагая, как будет работать программа, *прежде* чем писать ее код.

Если вы сразу приступите к написанию кода, то можете обнаружить проблемы, требующие изменения кода, который уже написан, что будет пустой тратой времени и усилий. И каждый раз, когда вы изменяете свой код, вы рискуете наделать новых ошибок, внося изменения. Намного эффективней знать, что вы хотите построить в итоге, прежде чем приступать к строительству. Теперь, когда у нас есть блок-схема, давайте создадим игру «Виселица» в главе 8!

8

НАПИСАНИЕ КОДА ИГРЫ «ВИСЕЛИЦА»

В этой главе вы узнаете много нового. Но не волнуйтесь, — прежде чем приступить к написанию кода игры, вы познакомитесь с новыми понятиями в интерактивной оболочке. Вы изучите *методы* — функции, назначенные объектам. А также изучите новый тип данных — *список*. После того как эти концепции будут понятны, написать рабочий код для игры «Виселица» станет значительно проще.

В ЭТОЙ ГЛАВЕ РАССМАТРИВАЮТСЯ СЛЕДУЮЩИЕ ТЕМЫ:

- Списки
- Оператор `in`
- Методы
- Строковые методы `split()`, `lower()`, `upper()`, `startswith()` и `endswith()`
- Инструкции `elif`

Исходный код игры «Виселица»

Код этой игры немного длиннее предыдущих примеров, но большая его часть — это ASCII-код для прорисовки «повешенного».

В редакторе файлов создайте новый файл, выбрав команду меню **File** ⇒ **New File** (Файл ⇒ Новый файл). В открывшемся окне введите приведенный ниже исходный код и сохраните файл под именем *hangman.py*. Затем запустите программу, нажав кла-

вишу **F5**. Если при выполнении программы возникают ошибки, сравните код, который вы набрали, с оригинальным кодом с помощью онлайн-инструмента на сайте **inventwithpython.com/diff/**.

hangman.py

```
1. import random
2. HANGMAN_PICS = ['''
3.   +---+
4.       |
5.       |
6.       |
7.      ===''', '''
8.   +---+
9.       |
10.       |
11.       |
12.      ===''', '''
13.   +---+
14.   0   |
15.   |   |
16.       |
17.      ===''', '''
18.   +---+
19.   0   |
20.  /|   |
21.       |
22.      ===''', '''
23.   +---+
24.   0   |
25.  /|\  |
26.       |
27.      ===''', '''
28.   +---+
29.   0   |
30.  /|\  |
31.  /    |
32.      ===''', '''
33.   +---+
34.   0   |
35.  /|\  |
36.  / \  |
```

```
37.      ===''']
38. words = 'аист акула бабуин баран барсук бобр бык верблюд волк воробей ворон выдра голубь гусь жаба зебра змея
    индюк кит кобра коза козел койот корова кошка кролик крыса курица лама ласка лебедь лев лиса лосось лось ля-
    гушка медведь моллюск моль мул муравей мышь норка носорог обезьяна овца окунь олень орел осел панда паук питон
    попугай пума семга скунс собака сова тигр тритон тюлень утка форель хорек черепаха ястреб ящерица'.split()
39.
40. def getRandomWord(wordList):
41.     # Эта функция возвращает случайную строку из переданного списка.
42.     wordIndex = random.randint(0, len(wordList) - 1)
43.     return wordList[wordIndex]
44.
45. def displayBoard(missedLetters, correctLetters, secretWord):
46.     print(HANGMAN_PICS[len(missedLetters)])
47.     print()
48.
49.     print('Ошибочные буквы:', end=' ')
50.     for letter in missedLetters:
51.         print(letter, end=' ')
52.     print()
53.
54.     blanks = '_' * len(secretWord)
55.
56.     for i in range(len(secretWord)): # заменяет пропуски отгаданными буквами
57.         if secretWord[i] in correctLetters:
58.             blanks = blanks[:i] + secretWord[i] + blanks[i+1:]
59.
60.     for letter in blanks: # Показывает секретное слово с пробелами между буквами
61.         print(letter, end=' ')
62.     print()
63.
64. def getGuess(alreadyGuessed):
65.   # Возвращает букву, введенную игроком. Эта функция проверяет, что игрок ввел только одну букву и ничего больше.
66.     while True:
67.         print('Введите букву.')
68.         guess = input()
69.         guess = guess.lower()
70.         if len(guess) != 1:
71.             print('Пожалуйста, введите одну букву.')
72.         elif guess in alreadyGuessed:
73.             print('Вы уже называли эту букву. Назовите другую.')
74.         elif guess not in 'абвгдеежзийклмнопрстуфхцчшщъыьэюя':
75.             print('Пожалуйста, введите БУКВУ.')
```

```
76.         else:
77.             return guess
78.
79. def playAgain():
80.   # Эта функция возвращает значение True, если игрок хочет сыграть заново; в противном случае возвращает False.
81.     print('Хотите сыграть еще? (да или нет)')
82.     return input().lower().startswith('д')
83.
84.
85. print('В И С Е Л И Ц А')
86. missedLetters = ''
87. correctLetters = ''
88. secretWord = getRandomWord(words)
89. gameIsDone = False
90.
91. while True:
92.     displayBoard(missedLetters, correctLetters, secretWord)
93.
94.     # Позволяет игроку ввести букву.
95.     guess = getGuess(missedLetters + correctLetters)
96.
97.     if guess in secretWord:
98.         correctLetters = correctLetters + guess
99.
100.         # Проверяет, выиграл ли игрок.
101.         foundAllLetters = True
102.         for i in range(len(secretWord)):
103.             if secretWord[i] not in correctLetters:
104.                 foundAllLetters = False
105.                 break
106.         if foundAllLetters:
107.             print('ДА! Секретное слово - "' + secretWord + '"! Вы угадали!')
108.             gameIsDone = True
109.     else:
110.         missedLetters = missedLetters + guess
111.
112.          # Проверяет, превысил ли игрок лимит попыток и проиграл.
113.         if len(missedLetters) == len(HANGMAN_PICS) - 1:
114.              displayBoard(missedLetters, correctLetters, secretWord)
115.  print('Выисчерпаливсепопытки!\nНеугаданобукв:'+str(len(missedLetters))+'иугаданобукв:'+str(len(correctLetters))+'.Былозагаданослово"'+secretWord+'".')
116.             gameIsDone = True
117.
```

```
118.     # Запрашивает, хочет ли игрок сыграть заново (только если игра завершена).
119.    if gameIsDone:
120.        if playAgain():
121.            missedLetters = ''
122.            correctLetters = ''
123.            gameIsDone = False
124.            secretWord = getRandomWord(words)
125.        else:
126.            break
```

Импорт модуля random

Программа «Виселица» случайным образом выбирает слово из заранее определенного списка и предлагает угадать это слово игроку. Возможность такого случайного выбора обеспечивает модуль random, который импортируется в первой строке кода.

```
1. import random
```

Переменная HANGMAN_PICS во второй строке немного отличается от переменных, с которыми вы уже знакомы. Для понимания следующего кода вы должны познакомиться с несколькими дополнительными концепциями.

Константы

Строки с 2 по 37 — это одна длинная инструкция присваивания для переменной HANGMAN_PICS.

```
2. HANGMAN_PICS = ['''
3.   +---+
4.       |
5.       |
6.       |
7.      ===''', '''

--пропущено--

37.      ===''']
```

Имя переменной состоит из прописных букв. Таким образом, по соглашению, обозначаются константы. *Константы* — это переменные, значения которых не изменяются в процессе работы программы с момента их объявления. Хотя вы можете изменить значение переменной HANGMAN_PICS также просто, как и любой другой переменной, имя прописными буквами напоминает вам, чтобы вы этого не делали.

Как и в случае со всеми остальными соглашениями по оформлению кода, вы вовсе *не обязаны* им следовать. Но, соблюдая их, вы сделаете код более понятным для других программистов.

Списки

Значение переменной HANGMAN_PICS состоит из нескольких строк. Это называется списком. В списках, в качестве элементов, могут содержаться различные значения.

Введите в интерактивную оболочку следующий код:

```
>>> animals = ['аксолотль', 'аргонавт', 'астрильд', 'альберт']
>>> animals
['аксолотль', 'аргонавт', 'астрильд', 'альберт']
```

Список animals содержит четыре значения. Список значений начинается и заканчивается квадратными скобками — []. Строковые значения (слова) помещаются в одинарные кавычки — '' и разделяются запятыми.

Эти слова называются *элементами* списка. Каждый элемент HANGMAN_PICS — это многострочное строковое значение.

Списки позволяют хранить несколько значений, не назначая отдельную переменную для каждого из них.

Без списков код выглядел бы примерно так:

```
>>> animals1 = 'аксолотль'
>>> animals2 = 'аргонавт'
>>> animals3 = 'астрильд'
>>> animals4 = 'альберт'
```

Такой код был бы слишком сложен для восприятия и обслуживания, так как содержал бы сотни, а то и тысячи строк. А один список может содержать целый ряд значений.

Доступ к элементам по их индексам

Можно получить доступ к любому элементу списка, указав номер элемента в квадратных скобках в конце имени переменной. Номер в квадратных скобках называется *индексом* элемента. В Python первому элементу присваивается индекс `0`. Второй элемент имеет индекс `1`, третий — `2` и так далее. Так как индексы в списках начинаются с `0`, а не с `1`, их называют списками *нулевого индекса*.

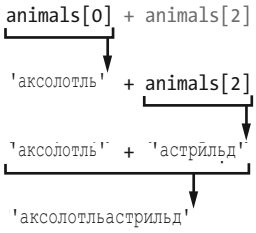
Пока вы находитесь в интерактивной оболочке и работаете со списком `animals`, введите по очереди: **`animals[0]`**, **`animals[1]`**, **`animals[2]`** и **`animals[3]`**, чтобы посмотреть, что произойдет.

```
>>> animals[0]
'аксолотль'
>>> animals[1]
'аргонавт'
>>> animals[2]
'астрильд'
>>> animals[3]
'альберт'
```

Напомню, что первый элемент списка `'аксолотль'` хранится под индексом `0`, а не `1`. Каждому элементу списка присваивается порядковый номер, начиная с `0`. С помощью квадратных скобок можно использовать элементы списка как любые другие переменные. Например, если в интерактивной оболочке ввести **`animals[0] + animals[2]`**, мы получим:

```
>>> animals[0] + animals[2]
'аксолотльастрильд'
```

Обе переменные с индексами `0` и `2` — строковые, поэтому они просто объединятся. Схематично это преобразование выглядит так:



Индекс за пределами диапазона и ошибка `IndexError`

Если указать индекс, значение которого превышает количество записей в списке, выполнение программы будет прервано и появится сообщение об ошибке `IndexError`. Чтобы увидеть пример такой ошибки, укажите в интерактивной оболочке следующий код:

```
>>> animals = ['аксолотль', 'аргонавт', 'астрильд', 'альберт']
>>> animals[9999]
Traceback (most recent call last):
  File "", line 1, in
    animals[9999]
IndexError: list index out of range
```

Ошибка возникла, потому что элемента с индексом `99999` не существует.

Присваивание индексов элементам

Значения элементов списка можно изменять путем *присваивания индексов*. В интерактивной оболочке наберите следующее:

```
>>> animals = ['аксолотль', 'аргонавт', 'астрильд', 'альберт']
>>> animals[1] = 'АРГОНАВТ'
>>> animals
['аксолотль', 'АРГОНАВТ', 'астрильд', 'альберт']
```

Строка, прописанная в списке под вторым индексом, примет новое значение `'АРГОНАВТ'`. Таким образом, выражение `animals[1]` само по себе преобразуется в значение списка с индексом два. В данном случае это выражение находится в левой части оператора присваивания, и вся инструкция заносит новое значение в этот элемент списка.

Конкатенация списков

Можно соединить несколько списков в один, как и обычные строки, используя оператор `+`. Такая операция называется *конкатенацией списков*. Чтобы понаблюдать за этой операцией, укажите следующий код в интерактивной оболочке:

```
>>> [1, 2, 3, 4] + ['апельсины', 'яблоки'] + ['Алиса', 'Борис']
[1, 2, 3, 4, 'апельсины', 'яблоки', 'Алиса', 'Борис']
```

Выражение `['апельсины'] + ['яблоки']` будет преобразовано в `['апельсины', 'яблоки']`. Но выражение `['апельсины'] + 'яблоки'` приведет к возникновению ошибки. При помощи оператора + нельзя складывать элементы списка и строки. Если необходимо добавить элементы в конец списка, не используя конкатенацию, следует применить метод `append()`, описанный в разделе «Методы `reverse()` и `append()`» далее в этой главе.

Оператор `in`

Оператор `in` позволяет определить наличие в списке элемента с определенным значением. Выражения с оператором `in` возвращают значение `True`, если искомое значение содержится в списке, или `False` – если нет. В интерактивной оболочке наберите следующее:

```
>>> animals = ['аксолотль', 'аргонавт', 'астрильд', 'альберт']
>>> 'астрильд' in animals
True
>>> 'муравей' in animals
False
```

Выражение `'астрильд' in animals` вернет значение `True`, так как строка `'астрильд'` — элемент списка `animals`. В списке она находится с индексом `2`. А вот выражение `'муравей' in animals` вернет значение `False`, потому что строки `'муравей'` в списке нет.

Оператор `in` также применим и к строкам. В этом случае он проверяет, является ли одна строка частью другой. Для наглядности наберите в интерактивной оболочке следующее:

```
>>> 'привет' in 'Алиса сказала привет Борису.'
True
```

Мы получили истинный результат, так как подстрока 'привет' содержится в строке в правой стороне инструкции.

Вызов методов

Метод — это функция, привязанная к объекту (списку или строке). Для вызова метода необходимо привязать его к конкретному объекту. В языке Python встроено много полезных методов, некоторые из них мы используем

в игре «Виселица». Но сначала рассмотрим несколько методов для работы со списками и строками.

Методы списков `reverse()` и `append()`

Списки поддерживают два встроенных метода, которые используются наиболее часто, — `reverse()` и `append()`. Метод `reverse()` разворачивает (обращает) список. Попробуйте ввести `spam = [1, 2, 3, 4, 5, 6, 'мяу', 'гав']`, а затем выполнить `spam.reverse()` для обращения списка. Далее введите `spam`, чтобы просмотреть содержимое переменной.

```
>>> spam = [1, 2, 3, 4, 5, 6, 'мяу', 'гав']
>>> spam.reverse()
>>> spam
['гав', 'мяу', 6, 5, 4, 3, 2, 1]
```

Но наиболее часто в списках используется метод `append()`. Он добавляет новые значения в конец списка. Наберите в интерактивной оболочке следующее:

```
>>> eggs = []
>>> eggs.append('самолет')
>>> eggs
['самолет']
>>> eggs.append('теплоход')
>>> eggs
['самолет', 'теплоход']
```

Эти методы изменяют списки, которые их вызывают. Они не создают новые списки. Это называется изменением списков *по месту*.

Строковый метод `split()`

Строки имеют встроенный метод `split()`, который возвращает список, сформированный из строковых переменных, на которые разбивается строка. Попробуйте использовать метод `split()`, введя следующий код:

```
>>> sentence = input()
Моя неутомимая мать только что приготовила нам буррито.
>>> sentence.split()
['Моя', 'неутомимая', 'мать', 'только', 'что', 'приготовила', 'нам', 'буррито.']
```

В результате получится список из восьми элементов, по одному элементу на каждое слово в исходной строке. Указателем для разделения строки на слова (элементы) является пробел между словами. Пробелы не включаются ни в один элемент списка.

В 38-й строке кода игры «Виселица» тоже используется метод `split()`, как показано ниже. Код выглядит длинным, но, на самом деле, это просто длинная строка слов, разделенных пробелами, с вызовом метода `split()` в конце. Метод `split()` создает список, в котором каждое слово из строки становится отдельным элементом списка.

```
38. words = 'аист акула бабуин баран барсук бобр бык верблюд волк воробей ворон выдра голубь
    гусь жаба зебра змея индюк кит кобра коза козел койот корова кошка кролик крыса курица лама
    ласка лебедь лев лиса лосось лось лягушка медведь моллюск моль мул муравей мышь норка но-
    сорог обезьяна овца окунь олень орел осел панда паук питон попугай пума семга скунс собака
    сова тигр тритон тюлень утка форель хорек черепаха ястреб ящерица'.split()
```

Метод `split()` облегчает программирование. В самом деле, чтобы создать список, надо записать слова в одинарных кавычках через запятую и заключить их в квадратные скобки. Например, так: `['аист', 'акула', 'бабуин'` и так далее. Правда, утомительно?

Вы можете добавить собственные слова в код строки 38 или удалить какие-нибудь из них, если не хотите, чтобы они присутствовали в игре. Главное, убедитесь, что слова разделены пробелами.

Получение секретного слова из списка

В строке 40 определяется функция `getRandomWord()`. Значения элементов списка передаются аргументу `wordlist` в качестве параметров. Эта функция возвращает единичное секретное слово из списка.

```
40. def getRandomWord(wordList):
41.     # Эта функция возвращает случайную строку из переданного списка.
42.     wordIndex = random.randint(0, len(wordList) - 1)
43.     return wordList[wordIndex]
```

В строке 42 из этого списка в качестве значения переменной `wordIndex` сохраняется элемент со случайным индексом. Случайный выбор элемента производится функцией `randint()` с двумя аргументами. Первый аргумент —

это 0 (для первого возможного индекса), а второй — значение выражения len(wordList) - 1, определяющего последний возможный индекс.

Напомним, индексация элементов списка начинается с 0, а не с 1. Если имеется список из трех элементов, то индекс первого элемента — 0, второго — 1, а третьего — 2. Длина такого списка равна 3, но индекса 3 в списке нет. Вот почему в строке 42 из длины списка вычитается единица. Код в строке 42 работает вне зависимости от размеров списка wordList. Теперь вы можете спокойно добавлять и удалять строки из списка wordList.

Переменная wordIndex хранит случайный индекс из списка, переданного с помощью параметра wordList. Код в строке 43 возвращает из списка wordList значение элемента с соответствующим индексом, который сохраняется как целое число в wordIndex.

Предположим, что ['яблоко', 'апельсин', 'виноград'] передается как аргумент в функцию getRandomWord(), тогда randint(0, 2) возвращает 2. Это значит, что строка 43, в которой возвращается значение wordList[2], вернет значение 'виноград'.

Вот так функция getRandomWord() возвращает случайную строку из списка wordList.

То есть входными данными для функции getRandomWord() служит список строк, а выходными — случайно выбранная строка. В игре «Виселица», таким образом, выбирается секретное слово, которое будет угадывать игрок.

Отображение игрового поля для игрока

Далее необходима функция прорисовки игрового поля игры «Виселица». На нем должно отображаться количество введенных игроком букв, угаданных как верно, так и ошибочно.

```
45. def displayBoard(missedLetters, correctLetters, secretWord):
46.     print(HANGMAN_PICS[len(missedLetters)])
47.     print()
```

В следующем коде определяется функция с именем displayBoard(). У этой функции есть три параметра:

- **missedLetters**. Строка букв, которые игрок назвал, но их нет в загаданном слове;
- **correctLetters**. Строка букв, которые игрок угадал в загаданном слове;
- **secretWord**. Строка с загаданным словом, которое игрок пытается угадать.

Сначала функция print() вызывает отображение игрового поля. Глобальная переменная HANGMAN_PICS содержит список строковых переменных для прорисовки всех возможных этапов игры. (Напомню, что глобальные переменные доступны из функции.) Код HANGMAN_PICS[0] отображает пустую «виселицу», код HANGMAN_PICS[1] показывает голову (когда игрок назвал неправильно одну букву), код HANGMAN_PICS[2] показывает голову и тело (когда игрок неправильно назвал две буквы) и так далее до HANGMAN_PICS[6], которая показывает «висельника» целиком.

Число букв, сохраняемое в переменной missedLetters, отражает количество неправильных предположений, сделанных игроком. Для определения этого числа вызывается функция len(missedLetters). То есть если значение переменной missedLetters равно, к примеру, 'аист', то код len('аист') вернет 4. Код HANGMAN_PICS[4] отобразит повешенного, соответствующего четырем промахам. Это то, во что преобразуется код HANGMAN_PICS[len(missedLetters)] в строке 46.

Код в строке 49 выводит сообщение 'Ошибочные буквы:' с символом пробела в конце, вместо символа новой строки.

```
49.     print('Ошибочные буквы:', end=' ')
50.     for letter in missedLetters:
51.         print(letter, end=' ')
52.     print()
```

В строке 50 начинается цикл for, в котором происходит перебор всех символов из строчного значения переменной missedLetters и вывод их на экран. Напомню, что выражение end=' ' замещает символ новой строки, который помещается в конце каждой строки, — единичным пробелом. Например, если значение missedLetters равно 'аызх', то такой цикл for выведет на экран а ы з х.

Остальная часть кода функции displayBoard() (строки 54–62) выводит на экран буквы и формирует строку — секретное слово, в котором еще не угаданные буквы замещены пробелами. Это достигается с помощью функции range() и среза списка.

Функции list() и range()

Функция range(), вызываемая с одним аргументом, возвращает последовательность чисел от 0 до величины аргумента, сам аргумент в последовательность не включается. Такую последовательность можно использовать в цикле for, но можно и преобразовать в список с помощью функции list(). Введите в интерактивную оболочку код list(range(10)):

```
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list('Привет')
['П', 'р', 'и', 'в', 'е', 'т']
```

Функция `list()` очень похожа на функции `str()` или `int()`. Она принимает последовательность величин и возвращает их в виде списка. С помощью функций `list()` и `range()` легко генерировать огромные списки. Например, наберите в интерактивной оболочке код **`list(range(10000))`**:

```
>>> list(range(10000))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,...
--пропуск--
...9990, 9991, 9992, 9993, 9994, 9995, 9996, 9997, 9998, 9999]
```

Получившийся список настолько огромен, что не помещается на экране. Но этот список можно сохранить в переменной.

```
>>> spam = list(range(10000))
```

Если функция `range()` вызывается с двумя целочисленными аргументами, она возвращает последовательность чисел, начиная с первого аргумента до (не включая) второго. Попробуйте набрать код **`list(range(10, 20))`** и вы получите следующее:

```
>>> list(range(10, 20))
[10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
```

Как видим, список начинается с `10` и доходит только до `19`, не включая `20`.

Срезы списков и строк

Срез списка создает новый список с подмножеством элементов родительского списка. Для создания среза списка используются два индекса (начальный и конечный), которые помещаются в конец имени, в квадратных скобках

и разделяются двоеточием. Для примера наберите в интерактивной оболочке следующее:

```
>>> spam = ['апельсины', 'бананы', 'морковка', 'халапеньо']
>>> spam[1:3]
['бананы', 'морковка']
```

Выражение `spam[1:3]` выводит список с элементами из списка `spam`, индексы которых от `1` до (не включая) `3`.

Если опустить первый индекс, то вместо него Python будет использовать индекс `0`.

```
>>> spam = ['апельсины', 'бананы', 'морковка', 'халапеньо']
>>> spam[:2]
['апельсины', 'бананы']
```

Если опустить второй индекс, то будет выведена оставшаяся часть списка, начиная с указанного индекса.

```
>>> spam = ['апельсины', 'бананы', 'морковка', 'халапеньо']
>>> spam[2:]
['морковка', 'халапеньо']
```

Такие же срезы можно производить и со строками. Символы строк будут аналогичны элементам списка. Наберите в интерактивной оболочке следующее:

```
>>> myName = 'Марсик объелся мышей'
>>> myName[4:12]
'ик объел'
>>> myName[:10]
'Марсик объ'
>>> myName[7:]
'объелся мышей'
```

В следующей части кода нашей игры используются срезы.

Вывод секретного слова с пробелами

При желании можно вывести секретное слово с пробелами вместо неугаданных букв. Можно также использовать символ нижнего подчеркивания (`_`). Сначала нужно создать строку из символов нижнего подчеркивания для всех букв секретного слова. Затем заменить пробелы этими символами. Например, для секретного слова `'ворон'` пустая строка-подчеркивание выглядит так: `'_ _ _ _ _'` (пять подчеркиваний). Если значение переменной `correctLeters` равно `'он'`, секретное слово надо напечатать в виде: `'_ о _ о н'`. Код в строках 54–58 делает именно это.

```
54.     blanks = '_' * len(secretWord)
55.
56.     for i in range(len(secretWord)): # заменяет пропуски отгаданными буквами
57.         if secretWord[i] in correctLetters:
58.             blanks = blanks[:i] + secretWord[i] + blanks[i+1:]
```

В строке 54 создается строковая переменная `blanks`, с подчеркиваниями по числу букв секретного слова. Напомню, что оператор `*` применим для переменных строкового и целого типов, поэтому выражение `'_'*5` будет преобразовано в строку `'_ _ _ _ _'`. Эта операция служит гарантом того, что переменная `blanks` содержит столько же подчеркиваний, сколько букв в секретном слове.

Код в строке 56 в цикле `for` производит последовательный перебор всех букв секретного слова и замещает подчеркивания буквами, содержащимися в переменной `correctLetters`.

Давайте рассмотрим цикл предыдущего примера с другими данными, пусть секретное слово `'ворон'`, а значение переменной `correcLetters` равно `'он'`. Нужно вывести на экран строку `'_ о _ о н'`. Давайте подумаем, как это сделать.

Код в строке 56 после вызова `len(secretWord)` вернет значение 5. Вызов `range(len(secretWord))` примет вид `range(5)`, тогда цикл `for` произведет итерации от 0 до 4 включительно.

Так как значение переменной `i` будет последовательно принимать значения [0, 1, 2, 3, 4], код цикла `for` будет выглядеть примерно так:

```
if secretWord[0] in correctLetters:
    blanks = blanks[:0] + secretWord[0] + blanks[1:]


if secretWord[1] in correctLetters:
    blanks = blanks[:1] + secretWord[1] + blanks[2:]
--пропуск--
```

Мы показали только две первые итерации цикла `for`, но, на самом деле, переменная итерации `i` поочередно будет принимать все значения последовательности, начиная с `0`.

В первой итерации `i` будет равна `0`, поэтому оператор `if` проверит, содержится ли буква секретного слова с индексом `0` в строке `correctLetters`. Такая проверка будет произведена в цикле для каждой буквы секретного слова, по одной итерации на каждую букву.

Если вы сомневаетесь в отношении, какого либо значения, например `secretWord[0]` или `blank[3:]`, обратитесь к рис. 8.1.

Там показаны значения `secretWord`, переменной `blanks` и индексы для каждой буквы в строке.

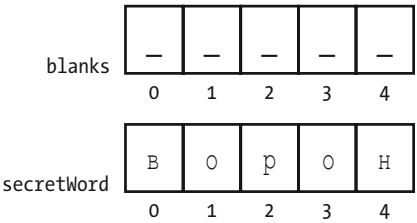


Рис. 8.1. Индексы строк `blanks` и `secretWord`

Если срез списка и список индексов заменить значениями их элементов, то код цикла выглядел бы примерно так:

```
if 'в' in 'он': # False
    blanks = '' + 'в' + '____' # Код в строке пропускается.
--пропуск--
if 'н' in 'он': # True
    blanks = '_о_о_' + 'н' + '' # Код в строке выполняется.

# переменной blanks присвоено значение _о_он'.
```

Пример кода, приведенный выше, делает все *то же самое*, когда значение `secretWord` равно `'ворон'`, а значение `correctLetters` равно `'он'`. Следующий фрагмент кода выводит на экран новое значение `blanks` с пробелами между буквами.

```
60.     for letter in blanks: # Показывает секретное слово с пробелами между буквами
61.         print(letter, end=' ')
62.     print()
```

Отметим, что цикл `for` в строке 60 не вызывает функцию `range()`. Вместо ранжирования операций последовательностью возвращаемой `range`, итерации производятся по значению строковой переменной `blanks`. При каждой итерации берется одна новая буква из строки `'ворон'` переменной `blanks`.

В результате после добавления пробелов будет выведено `'_о_о_н'`.

Получение предположений игрока

При вызове функции `getGuess()` игрок может ввести предполагаемую букву. Эта функция возвращает букву, предложенную игроком, в виде строки. Далее функция `getGuess()` проверяет допустимость введенного символа, прежде чем передать его в функцию.

```
64. def getGuess(alreadyGuessed):
65.     # Возвращает букву, введенную игроком. Эта функция проверяет, что игрок ввел только одну
букву и ничего больше.
```

Строка букв, предложенных игроком, передается в качестве аргумента в параметр `alreadyGuessed` функции. Затем функция `getGuess()` просит игрока ввести одну букву. Эту букву функция `getGuess()` вернет как свое значение. Так как Python чувствителен к регистру, следует убедиться, что введена строчная буква, чтобы ее можно было корректно сопоставить с буквами секретного слова. Для этого понадобится метод `lower()`.

Строковые методы `lower()` и `upper()`

Для иллюстрации действия, производимого методом, в интерактивной оболочке введите **`'Привет, мир!'.lower()`**.

```
>>> 'Привет, мир!'.lower()
'привет, мир!'
```

Метод `lower()` возвращает строку, в которой все буквы строчные. Существует также и обратный строковый метод `upper()`, который возвращает строку с прописными буквами. Попробуйте в интерактивной оболочке ввести **`'Привет, мир!'.upper()`**.

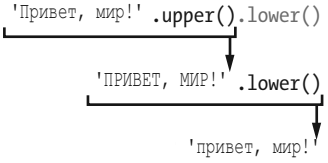
```
>>> 'Привет, мир!'.upper()
'ПРИВЕТ, МИР!'
```

Так как метод upper() возвращает строку, вызывать его следует строковой переменной.

Теперь В интерактивной оболочке введите следующие команды:

```
>>> 'Привет, мир!'.upper().lower()
'привет, мир!'
```

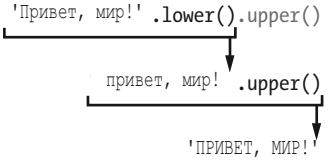
Код 'Привет, мир!'upper() преобразуется в строку 'ПРИВЕТ, МИР!', а затем вызывается метод lower(). Он вернет строку 'привет, мир!', что и станет финалом преобразования.



Порядок следования методов важен. 'Привет, мир!'.lower().upper(), это не то же самое, что 'Привет, мир!'.upper().lower().

```
>>> 'Привет, мир!'.lower().upper()
'ПРИВЕТ, МИР!'
```

Преобразование будет выглядеть так:



Если строка хранится в переменной, то строковые методы можно вызывать следующим способом:

```
>>> spam = 'Привет, мир!'
>>> spam.upper()
'ПРИВЕТ, МИР!'
```

Этот код не изменяет значение переменной spam. Сама переменная spam сохраняет строку 'Привет, мир!'.

Возвращаясь к программе «Виселица», метод `lower()` используется, когда игроку предлагается ввести предполагаемую букву.

```
66.     while True:
67.         print('Введите букву.')
68.         guess = input()
69.         guess = guess.lower()
```

Теперь, если игрок введет прописную букву, функция `getGuess()` вернет эту букву в виде строчного символа.

Завершение цикла `while`

В строке 66 цикл `while` будет предлагать игроку ввести букву до тех пор, пока не будет введена ранее не предлагавшаяся буква.

Условием цикла `while` выступает логическая переменная в значении `True`. Это значит, что единственным способом завершить цикл является выполнение инструкции `break` или выполнение инструкции `return`, которая обеспечивает выход не только из цикла, но и из функции.

Внутренний код цикла предлагает игроку ввести букву, которая будет сохранена в переменной `guess`. Если игрок ввел прописную букву, она будет преобразована в строчную с помощью кода в строке 69.

Инструкции `elif`

Следующая часть кода игры «Виселица» содержит инструкции `elif`. Инструкция `elif` означает, что «Если это истинно, сделать так. А если выполняется другое условие, то сделать так. Если не выполняется ни одно условие, сделать вот так». Взгляните на следующий код:

```
if catName == 'Пушок':
    print('Ваш котенок пушистенький.')
elif catName == 'Рыжик':
    print('Ваш котенок рыженький.')
else:
    print('Ваш котенок ни пушистенький и ни рыженький.')
```

Если значение переменной `catName` эквивалентно строке `'Пушок'`, то условие инструкции `if` истинно и блок `if` сообщает игроку, что его котенок пу-

шистенький. Но, если условие ложно, Python проверяет следующее условие. Если переменной catName присвоено значение 'Рыжик', на экран выводится строка 'Ваш котенок рыженький.'. Если оба условия ложны, игроку сообщается, что его котенок ни пушистенький и ни рыженький.

Инструкций elif может быть столько, сколько нужно.

```
if catName == 'Пушок':
    print('Ваш котенок пушистенький.')
elif catName == 'Рыжик':
    print('Ваш котенок рыженький.')
elif catName == 'Лохматик':
    print('Ваш котенок лохматенький.')
elif catName == 'Чудик':
    print('Ваш котенок чудной.')
else:
    print('Ваш котенок ни пушистенький и ни рыженький и ни лохматенький и ни чудной.')
```

Если условие одной инструкции elif истинно, выполняется код именно этой инструкции, затем происходит возврат к первой строке цикла. Таким образом, за один проход выполняется код *одного и только одного* программного блока конструкции if-elif-else. Инструкция else прекращает работу блока if, если больше нет необходимости проверять выполнение условий блока.

Проверка допустимости предположения игрока

Переменная guess содержит буквы, предложенные игроком. Программа должна удостовериться, что вводимые символы допустимы, это должна быть только одна буква, при этом не вводившаяся ранее. Если это условие не выполняется, цикл возвращается к началу и снова запрашивает букву.

```
70.         if len(guess) != 1:
71.             print('Пожалуйста, введите одну букву.')
72.         elif guess in alreadyGuessed:
73.             print('Вы уже называли эту букву. Назовите другую.')
74.         elif guess not in 'абвгдеежзийклмнопрстуфхцчшщъыьэюя':
75.             print('Пожалуйста, введите БУКВУ.')
76.         else:
77.             return guess
```

В строке 70 проверяется, что введено не более одного символа, код в строке 72 проверяет, что предложенная буква не содержится в переменной alreadyGuessed, код в строке 74 проверяет, что введен символ стандартного русского алфавита.

Если хотя бы одно условие не выполняется, игроку предлагается ввести другую букву.

Если выполнены все условия, программа переходит к исполнению кода блока и возвращает значение предложенной буквы в строке 77.

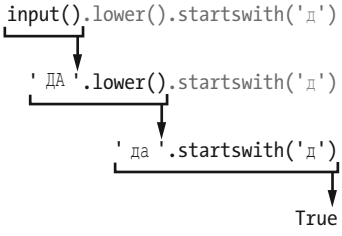
Напомню, что в конструкции if-elif-else исполняется только один программный блок.

Предложение игроку сыграть заново

Функция playAgain() содержит лишь вызов функции print() и инструкцию return.

```
79. def playAgain():
80.  # Эта функция возвращает значение True, если игрок хочет сыграть заново; в противном случае возвращает False.
81.     print('Хотите сыграть еще? (да или нет)')
82.     return input().lower().startswith('д')
```

Инструкция return завершает выполнение программы, но ее действие можно отменить. Рассмотрим подробнее, какие преобразования производит Python, если игрок ответил ДА.



Функция playAgain() позволяет игроку ответить «Да» или «Нет» на предложение еще одного раунда. Игрок должен ввести Да, да, Д или что-нибудь еще, начинающееся с буквы д, и это значение будет означать «Да». Если игрок ввел ДА, функция input() возвращает значение 'ДА'. Выражение 'ДА'.lower() возвращает строку в нижнем регистре (строчными буквами). То есть значение 'ДА' превращается в 'да'.

Но, кроме этого, производится еще и вызов метода startswith('д'). Эта функция возвращает True, если вызывающая строка начинается с ука-

занного параметра, или False, если это не так. Например, выражение 'да'.startwith('д') возвращает значение True.

Вот и весь смысл этого выражения. Оно позволяет игроку ввести ответ, переводит его в строчную форму, проверяет, что ответ начинается с буквы д, и затем возвращает True или False.

Замечу, что еще существует строковый метод endwith(*некаяСтрока*), который возвращает True, если вызывающая строка заканчивается строкой указанной в аргументе *некаяСтрока*, и False в противном случае. Метод endwith() в некотором смысле противоположен методу startwith().

Обзор функций игры

Это все функции, использованные в игре «Виселица». Вот их краткий обзор:

- **getRandomWord(wordList)** случайным образом выбирает одну строку из списка. Так выбирается слово, которое будет угадывать игрок;
- **displayBoard(missedLetters, correctLetters, secretWord)** отображает текущее состояние игры, показывает, сколько букв игрок уже предложил для определения слова, и какие из них оказались ошибочными. Для корректной работы этой функции необходимы три параметра. Строковые переменные correctLetters и missedLetter для хранения букв, которые были предложены игроком и тех, которых не оказалось в секретом слове, а также строковая переменная secretWord, содержащая секретное слово, которое пытается угадать игрок. Функция не возвращает каких-либо значений;
- **getGuess(alreadyGuessed)** проверяет, не содержится ли введенная игроком буква в строковой переменной alreadyGuessed. Функция возвращает букву, введенную игроком, если она допустима;
- **playAgain()** спрашивает игрока, не хочет ли он сыграть еще раз. Функция возвращает True, если игрок соглашается, или False, если отказывается.

После описания функций со строки 85 начинается основная часть (тело) программы. Все, что выше этой строки, «всего лишь» описание функций и большой код инструкции присваивания для переменной HANGMAN_PICS.

Игровой цикл

Основная часть кода программы «Виселица» отображает на экране название игры, содержит несколько переменных и запускает цикл while. В этом разделе рассмотрим пошаговое выполнение оставшейся части программного кода.

```
85. print('В И С Е Л И Ц А')
86. missedLetters = ''
87. correctLetters = ''
88. secretWord = getRandomWord(words)
89. gameIsDone = False
```

Код в строке 85 вызывает функцию `print()`, которая выводит на экран заголовок игры в момент ее запуска. Затем строковой переменной `missedLetters` присваивается пустое значение, переменной `correctLetters` тоже присваивается пустое значение, так как игрок не предложил еще никаких букв.

В строке 88 вызывается функция `getRandomWord(words)`, которая выбирает случайным образом секретное слово из списка.

Код в строке 89 присваивает переменной `gameIsDone` значение `False`. Код присвоит этой переменной значение `True`, когда поступит сигнал завершения игры, и предложит игроку сыграть заново.

Вызов функции `displayBoard()`

Оставшаяся часть программы состоит из цикла `while`. Условие цикла всегда истинно, а это значит, что он будет выполняться бесконечно долго, до тех пор, пока не будет инициировано выполнение инструкции `break`. (Это происходит в строке 126.)

```
91. while True:
92.     displayBoard(missedLetters, correctLetters, secretWord)
```

Код в строке 92 вызывает функцию `displayBoard()`, передавая ей значения трех переменных, установленных в строках 86, 87 и 88. В зависимости от того, сколько букв игрок угадал правильно и сколько раз ошибся, эта функция выводит на экран соответствующее изображение «повешенного».

Ввод игроком угадываемой буквы

Далее вызывается функция `getGuess()`, чтобы игрок мог ввести угадываемую букву.

```
94.     # Позволяет игроку ввести букву.
95.     guess = getGuess(missedLetters + correctLetters)
```

В функцию передается параметр alreadyGuessed для определения — вводил игрок такую букву раньше или еще нет. Код в строке 95 конкатенирует строковые переменные missedLetters и correctLetters, а затем передает результат в качестве аргумента параметру alreadyGuessed.

Проверка наличия буквы в секретном слове

Если предложенная буква есть в секретном слове, она добавляется в конец строковой переменной correctLetters.

```
97.     if guess in secretWord:
98.         correctLetters = correctLetters + guess
```

Код в последней строке добавляет новое значение в переменную correctLetters.

Проверка — не победил ли игрок

Как программе узнать, что игрок угадал все буквы, содержащиеся в секретном слове? Итак, переменная correctLetters содержит все буквы, угаданные игроком правильно, а переменная secretWord — секретное слово. Но простую проверку missedLetters == correctLetters сделать невозможно. Если, например, секретное слово 'ворон', а correctLetters равно 'внро', значение выражения correctLetters == secretWord будет ложным, хотя игрок *угадал* все буквы.

Единственно возможный путь — это произвести сравнение каждой буквы из correctLetters с буквами в переменной secretWord. Игрок побеждает тогда и только тогда, если каждая буква из переменной secretWord содержится в переменной correctLetters.

```
100.        # Проверяет, выиграл ли игрок.
101.        foundAllLetters = True
102.        for i in range(len(secretWord)):
103.            if secretWord[i] not in correctLetters:
104.                foundAllLetters = False
105.                break
```

Понятно, что если в переменной secretWord найдена буква, которой нет в переменной correctLetters, игрок *не* победил. Новая переменная foundAllLetters со значением True устанавливается в строке 101 до начала

цикла. Цикл начинается в предположении, что все буквы секретного слова угаданы. Но в строке 104, в процессе выполнения цикла, значение переменной foundAllLetters меняется на False, как только обнаружена первая буква из переменной secretWord, не содержащаяся в переменной correctLettrs.

Если все буквы секретного слова обнаружены, игроку сообщается о его победе, а переменная gameIsDone принимает значение True.

```
106.        if foundAllLetters:
107.            print('ДА! Секретное слово - "' + secretWord + '"! Вы угадали!')
108.            gameIsDone = True
```

Обработка ошибочных предположений

В строке 109 начинается блок else.

```
109.    else:
110.        missedLetters = missedLetters + guess
```

Напомним, что код этого блока выполняется, если условие цикла for ложно. Но какое условие? Чтобы узнать это, наведите указатель на начало ключевого слова else и переместите его вверх. Вы увидите, что ключевое слово else располагается точно в той же позиции, что и if, в строке 97.

```
97.    if guess in secretWord:
--пропуск--
109.   else:
110.       missedLetters = missedLetters + guess
```

Таким образом, если условие в строке 97 (guess in secretWord) ложно, выполняется блок else.

Неправильно угаданные буквы добавляются в строковую переменную missedLetters в строке 110. Это происходит также как в строке 98 с буквами, угаданными правильно.

Проверка — не проиграл ли игрок

Каждый раз, когда игрок вводит неправильную букву, она добавляется в переменную missedLetters. Таким образом, длина значения переменной

missedLetters (или в коде — len(missedLetters)) становится равна числу ошибочных предположений.

```
112.         # Проверяет, превысил ли игрок лимит попыток и проиграл.
113.        if len(missedLetters) == len(HANGMAN_PICS) - 1:
114.             displayBoard(missedLetters, correctLetters, secretWord)
115.           print('Вы исчерпали все попытки!\nНе угадано букв: ' + str(len(missedLetters)) + ' и угадано
                    букв: ' + str(len(correctLetters)) + '. Было загадано слово "' + secretWord + '".')
116.            gameIsDone = True
```

Переменная HANGMAN_PICS содержит семь строк с ASCII-символами рисунка. Таким образом, если длина строки missedLetters равна len(HANGMAN_PICS) – 1 (то есть 6), игрок исчерпал лимит предположений. Если рисунок «повешенного» завершен — игрок проиграл. Напомню, HANGMAN_PICS[0] — первый элемент списка, а HANGMAN_PICS[6] — последний.

Код в строке 115 выводит секретное слово, а строка 116 присваивает переменной gameIsDone значение True.

```
118.     # Запрашивает, хочет ли игрок сыграть заново (только если игра завершена).
119.    if gameIsDone:
120.        if playAgain():
121.            missedLetters = ''
122.            correctLetters = ''
123.            gameIsDone = False
124.            secretWord = getRandomWord(words)
```

Завершение или перезагрузка игры

Вне зависимости от победы или проигрыша игра должна спросить игрока — хочет ли он сыграть снова. Функция playAgain() обрабатывает полученный ответ («Да» или «Нет») в строке 120.

Если игрок хочет начать игру заново, значения переменных missedLetters и correctLetters надо обнулить, переменной gameIsDone присвоить значение False и выбрать новое секретное слово. После того как выполнение цикла while возвращается к началу, к строке 91, игровой интерфейс, отображаемый на экране, перезагружается и готов к новой игре.

Если игрок не ввел ничего, что начинается с буквы д, при запросе — хочет ли он сыграть заново, условие в строке 120 становится ложным и выполняется блок else.

```
125.        else:
126.            break
```

Инструкция `break` приводит к выполнению первой инструкции после цикла. Но, так как после цикла нет исполняемых инструкций, происходит завершение работы программы.

Заключение

«Виселица» — наиболее продвинутая игра из числа рассмотренных. В процессе ее создания вы изучили несколько новых понятий. По мере усложнения игр неплохо было бы составлять наброски блок-схем того, что должно выполняться в программе.

Список — это программный объект, который может содержать другие объекты. Методы — это функции, привязанные к какому-либо объекту. Списки содержат встроенный метод `append()`. Для строк существуют следующие методы: `lower()`, `upper()`, `split()`, `startswith()` и `endswith()`. Остальная часть книги познакомит вас со многими другими методами и типами данных.

Инструкция `elif` позволяет добавить условия выполнения «или-еще-если» в середину конструкции `if-else`.

9

ДОРАБОТКА ИГРЫ «ВИСЕЛИЦА»

Теперь, когда готова основа игры «Виселица», давайте рассмотрим некоторые возможности расширения ее функциональности. В этой главе вы создадите несколько групп секретных слов и добавите возможность изменения уровня сложности игры.

В ЭТОЙ ГЛАВЕ РАССМАТРИВАЮТСЯ СЛЕДУЮЩИЕ ТЕМЫ:

- Тип данных «Словарь»
- Пары ключ-значение
- Методы словаря `keys()` и `values()`
- Множественное назначение переменных

Увеличение числа угадываний

После того как вы сыграли в игру «Виселица» несколько раз, вы, возможно, подумали, что шесть попыток для угадывания недостаточно для разгадки многих слов.

Число попыток можно легко увеличить, добавив строки в список `HANGMAN_PICS`.

Сохраните файл *hangman.py* под именем *hangman2.py*. Для расширения списка, содержащего ASCII-символы для прорисовки «висельника», добавьте следующие строки, начиная со строки 37:

```
37.     ===''', '''
38.  +---+
39. [O   |
40. /|\  |
41. / \  |
42.     ===''', '''
43.  +---+
44. [O]  |
45. /|\  |
46. / \  |
47.     ===''']
```

Этот код добавляет два новых многострочных значения в список HANGMAN_PICS, одна строка служит для прорисовки левого уха «висельника», другая для прорисовки обоих ушей. Программа сообщает игроку о его проигрыше на основании значения выражения `len(missedLetters ) == len(HANGMAN_PICS) - 1`, поэтому в других изменениях необходимости нет. Остальная часть программы будет прекрасно работать с новым списком.

Словари

В первой версии программы «Виселица» мы использовали список с названиями животных, но вы можете изменить список слов в строке 48. Например, вместо животных можно использовать цвета.

```
48. words = 'красный оранжевый желтый зеленый синий голубой фиолетовый белый черный коричневый'.
    split()
```

Или геометрические фигуры.

```
48. words = 'квадрат треугольник прямоугольник круг эллипс ромб трапеция параллелограмм
    пятиугольник шестиугольник восьмиугольник'.split()
```

Или фрукты.

```
48. words = 'яблоко апельсин лимон лайм груша мандарин виноград грейпфрут персик банан абрикос
    манго банан нектарин'.split()
```

Внеся некоторые изменения, вы можете доработать код так, что игра «Виселица» будет использовать несколько наборов слов, таких как животные, цвета, фигуры или фрукты.

Программа будет сообщать игроку, какой набор использован для выбора секретного слова.

Чтобы произвести эти изменения, понадобится новый тип данных, который называется *словарем*. Словарь, также как и список, — это набор значений. Но вместо доступа к элементам по целочисленным индексам в словаре возможен доступ к элементам по индексам произвольного типа. В словаре эти индексы называются *ключами*.

При работе со словарем используются фигурные скобки `{}` вместо квадратных `[]`.

В качестве примера в интерактивной оболочке наберите следующее:

```
>>> spam = {'привет':'Привет всем, как дела?', 4:'бекон', 'курс':9999 }
```

Пара величин, разделенная двоеточием внутри фигурных скобок, называется *парой ключ-значение*. Левая часть пары (перед двоеточием) — это *ключ*, правая часть (после двоеточия) — *значение* ключа. Получить доступ к отдельным значениям можно с помощью ключей.

Чтобы попрактиковаться, выполните в интерактивной оболочке следующий пример:

```
>>> spam = {'привет':'Привет всем, как дела?', 4:'бекон', 'курс':9999 }
>>> spam['привет']
'Привет всем, как дела?'
>>> spam[4]
'бекон'
>>> spam['курс']
9999
```

Внутри квадратных скобок можно помещать не только целочисленные значения, но и строковые. В приведенном примере, в качестве ключей словаря `spam`, использованы величины целого типа — `4` и строкового — `'курс'`.

Определение размера словаря с помощью функции `len()`

С помощью функции `len()` можно определить количество пар ключ-значение, сохраненных в словаре.

Для примера наберите в интерактивной оболочке следующее:

```
>>> stuff = {'привет':'Привет всем, как дела?', 4:'бекон', 'курс':9999 }
>>> len(stuff)
3
```

Функция `len()` вернула целочисленное значение `3`, равное количеству пар ключ-значение в словаре.

Различия между списком и словарем

Одно отличие словаря от списка уже было рассмотрено — это возможность использования в качестве ключей значений произвольного типа, а не только целых чисел. Напомню, что `0` и `'0'` — разные величины, т.е. это будут разные ключи.

Введите в интерактивной оболочке следующий код:

```
>>> spam = {'0':'строка', 0:'число'}
>>> spam[0]
'число'
>>> spam['0']
spam['0']
```

Словари, подобно спискам, можно обрабатывать циклически, используя ключи в цикле `for`.

Чтобы посмотреть, как это работает, введите в интерактивной оболочке следующие команды:

```
>>> favorites = {'фрукты':'апельсины', 'животные':'коты', 'число':42}
>>> for k in favorites:
  print(k)
фрукты
животные
число
>>> for k in favorites:
  print(favorites[k])
апельсины
коты
42
```

Ключи и значения могут выводиться в различном порядке, потому что словари, в отличие от списков, не упорядочены (не ранжированы). Первым элементом списка с именем `listStuff`, будет `listStuf[0]`. Но в словаре нет первого элемента, потому что нет какого-нибудь порядка расположения элементов. В приведенном коде Python просто выбрал тот порядок, в котором элементы хранились в памяти, нет никаких гарантий, что в следующий раз порядок будет тем же.

В интерактивной оболочке наберите следующее:

```
>>> favorites1 = {'фрукт':'апельсины', 'число':42, 'животное':'кот'}
>>> favorites2 = {'животное':'кот', 'число':42, 'фрукт':'апельсины'}
>>> favorites1 == favorites2
True
```

Выражение `favorites1 == favorites2` истинно потому, что словари не упорядочены и считаются эквивалентными, если состоят из одинаковых пар ключ-значение. А списки с одинаковыми значениями элементов, но разным порядком их следования, равны не будут.

Для демонстрации этих различий наберите в интерактивной оболочке следующее:

```
>>> listFavs1 = ['апельсины', 'коты', 42]
>>> listFavs2 = ['коты', 42, 'апельсины']
>>> listFavs1 == listFavs2
False
```

Выражение `listFavs1 == listFavs2` ложно, так как содержимое списков расположено в разном порядке.

Методы словаря `keys()` и `values()`

Словари содержат два полезных метода — `keys()` и `values()`. Эти методы возвращают значения, тип которых называется `dict_keys` и `dict_values`, соответственно. Подобно большинству ранжированных объектов, данные этих типов возвращаются в форме списка функцией `list()`.

Наберите следующий код:

```
>>> favorites = {'фрукт':'апельсин', 'животное':'кот', 'число':42}
>>> list(favorites.keys())
['фрукт', 'животное', 'число']
```

```
>>> list(favorites.values())
['апельсин', 'кот', 42]
```

Используя методы keys() или values() в функции len(), можно получить список ключей или значений словаря.

Использование словаря слов в игре «Виселица»

Давайте изменим код в новой версии игры «Виселица», добавив поддержку разных наборов секретных слов. Во-первых, заменим значение переменной words, создав словарь, в котором ключи представлены строками, а значения — списками строк. Строковый метод split() вернет список строк, по одному слову в каждой строке.

```
48. words = {'Цвета':'красный оранжевый желтый зеленый синий голубой фиолетовый белый черный
    коричневый'.split(),
49. 'Фигуры':'квадрат треугольник прямоугольник круг эллипс ромб трапеция параллелограмм
    пятиугольник шестиугольник восьмиугольник'.split(),
50. 'Фрукты':'яблоко апельсин лимон лайм груша мандарин виноград грейпфрут персик банан абрикос
    манго банан нектарин'.split(),
51. 'Животные':'аист бабуин баран барсук бык волк зебра кит коза корова кошка кролик крыса лев
    лиса лось медведь мул мышь норка носорог обезьяна овца олень осел панда пума скунс собака
    сова тигр тюлень хорек ящерица'.split()}
```

Строки 48–51 — все еще одна инструкция присваивания. Инструкция заканчивается закрывающей фигурной скобкой в строке 51.

Случайный выбор из списка

Функция choice() модуля random принимает в качестве аргумента список и возвращает из него случайное значение. Это похоже на то, что ранее делала функция getRandomWord().

В новой версии функции getRandomWord() будет использована функция choice().

Чтобы увидеть, как работает функция choice(), В интерактивной оболочке введите следующие команды:

```
>>> import random
>>> random.choice(['кот', 'собака', 'мышь'])
```

```
'мышь'
>>> random.choice(['кот', 'собака', 'мышь'])
'кот'
```

Подобно тому, как функция `randint()` возвращает случайное целое число, функция `choice()` возвращает случайное значение из списка.

Изменения функции `getRandomWord()` таковы, что теперь ее параметр — словарь, состоящий из списков строк, а не просто список строк.

Вот как выглядела оригинальная функция:

```
40. def getRandomWord(wordList):
41.     # Эта функция возвращает случайную строку из переданного списка.
42.     wordIndex = random.randint(0, len(wordList) - 1)
43.     return wordList[wordIndex]
```

А вот как выглядит код этой функции после изменения:

```
53. def getRandomWord(wordDict):
54.    # Эта функция возвращает случайную строку из переданного словаря списков строк, а также ключ.
55.     # Во-первых, случайным образом выбираем ключ из словаря:
56.     wordKey = random.choice(list(wordDict.keys()))
57.
58.     # Во-вторых, случайным образом выбираем слово из списка ключей в словаре:
59.     wordIndex = random.randint(0, len(wordDict[wordKey]) - 1)
```

Имя `wordList` изменено на `wordDict` для большей наглядности.

Теперь вместо случайного выбора слова из списка строк сначала случайно выбирается ключ из словаря `wordDict` посредством вызова `random.choice()`.

И вместо возвращения строки `wordList[wordIndex]` функция возвращает список с двумя элементами.

Первый элемент — это `wordDict[wordKey][wordIndex]`.

Второй элемент — `wordKey`.

Выражение `wordDict[wordKey][wordIndex]` в строке 61 может выглядеть сложным, но его действие понять очень просто.

Во-первых, представим, что переменной `wordKey` присвоено значение `'Фрукты'`, а переменной `wordIndex` — значение `5`. Тогда преобразование `wordDict[wordKey][wordIndex]` будет выглядеть так:

```
wordDict[wordKey][wordIndex]
        ↓
wordDict['Фрукты'][wordIndex]
        ↓
['яблоко', 'апельсин', 'лимон', 'лайм', 'груша', 'мандарин', 'виноград', 'грейпфрут',
'персик', 'банан', 'абрикос', 'манго', 'банан', 'нектарин'][wordIndex]
        ↓
['яблоко', 'апельсин', 'лимон', 'лайм', 'груша', 'мандарин', 'виноград', 'грейпфрут',
'персик', 'банан', 'абрикос', 'манго', 'банан', 'нектарин'][5]
        ↓
        'мандарин'
```

В этом случае элемент списка, возвращенный функцией, — это строка `'мандарин'`. (Напомню, что индексы начинаются с `0`, поэтому индекс `[5]` отсылает к шестому элементу списка, а не к пятому.)

Так как функция `getRandomWord()` теперь возвращает список из двух элементов, а не строку, то и значение переменной `secretWord` будет списком, а не строкой. Чтобы сохранить эти значения в двух разных переменных, можно использовать множественное присваивание, которое мы рассмотрим далее в этой главе.

Удаление элементов списка

Инструкция `del` удаляет элементы с указанными индексами из списка. Так как `del` — инструкция, а не функция и не оператор, она указывается без круглых скобок и не имеет возвращаемого значения. Понаблюдайте, как работает эта инструкция, используя следующий код:

```
>>> animals = ['аксолотль', 'аргонавт', 'астрильд', 'альберт']
>>> del animals[1]
>>> animals
['аксолотль', 'астрильд', 'альберт']
```

Отмечу, что когда удаляется элемент с индексом `1`, элемент, имевший индекс `2`, становится элементом с индексом `1`, элемент, имевший индекс `3`, приобретает индекс `2` и так далее. Вышеописанная процедура удаляет только один элемент и, соответственно, уменьшает количество индексов.

Выражение `del animals[1]` можно вводить снова и снова, удаляя по одному элементу списка.

```
>>> animals = ['аксолотль', 'аргонавт', 'астрильд', 'альберт']
>>> del animals[1]
>>> animals
['аксолотль', 'астрильд', 'альберт']
>>> del animals[1]
>>> animals
['аксолотль', 'альберт']
>>> del animals[1]
>>> animals
['аксолотль']
```

Длина списка HANGMAN_PICS — это число сделанных игроком предположений. Удаляя строки из этого списка, можно сокращать допустимое количество предположений и, тем самым, усложнять игру.

Добавьте следующие строки в код вашей программы, между строкой print('В И С Е Л И Ц А') и строкой missedLetters = ''.

```
print('В И С Е Л И Ц А')

difficulty = ''
while difficulty not in 'ЛСТ':
     print('Выберите уровень сложности: Л - Легкий, С - Средний, Т - Тяжелый')
    difficulty = input().upper()
if difficulty == 'С':
    del HANGMAN_PICS[8]
    del HANGMAN_PICS[7]
if difficulty == 'Т':
    del HANGMAN_PICS[8]
    del HANGMAN_PICS[7]
    del HANGMAN_PICS[5]
    del HANGMAN_PICS[3]

missedLetters = ''
```

Этот код удаляет элементы списка HANGMAN_PICS, делая его короче, в зависимости от выбранного уровня сложности игры. Чем выше уровень сложности, тем больше элементов удаляется из списка HANGMAN_PICS, сокращая возможное количество попыток угадывания. Остальная часть кода игры «Виселица» использует длину этого списка для определения момента вывода сообщения игроку об исчерпании количества попыток.

Множественное присваивание

Множественное присваивание — это возможность назначения нескольким переменным различных значений в одной строке. Для множественного присваивания запишите переменные через запятую и присвойте им список значений. Ниже представлен пример записи, введите ее в интерактивной оболочке:

```
>>> fruit, animal, number = ['апельсин', 'кот', 42]
>>> fruit
'апельсины'
>>> animal
'кот'
>>> number
42
```

Исполнение кода, приведенного в примере, равноценно выполнению следующих операций присваивания:

```
>>> fruit = ['апельсин', 'кот', 42][0]
>>> animal = ['апельсин', 'кот', 42][1]
>>> number = ['апельсин', 'кот', 42][2]
```

Количество строк с именами переменных в левой части инструкции присваивания должно быть эквивалентно количеству элементов, указанных в правой части. Python автоматически присвоит значение первого элемента списка первой переменной, второй переменной — значение второго элемента и так далее. Если число переменных будет отлично от числа элементов списка, Python интерпретирует это как ошибку, вот так:

```
>>> fruit, animal, number, text = ['апельсин', 'кот', 42, 10, 'привет']
Traceback (most recent call last):
    File "<pyshell#8>", line 1, in <module>
        fruit, animal, number, text = ['апельсин', 'кот', 42, 10, 'привет']
ValueError: too many values to unpack
>>> fruit, animal, number, text = ['апельсин', 'кот']
Traceback (most recent call last):
    File "<pyshell#9>", line 1, in <module>
        fruit, animal, number, text = ['апельсин', 'кот']
ValueError: need more than 2 values to unpack
```

Изменение строк 120 и 157 в коде игры «Виселица» для использования множественного присваивания значений, возвращаемых функцией `getRandomWord()` выглядит следующим образом:

```
119. correctLetters = ''
120. secretWord, secretSet = getRandomWord(words)
121. gameIsDone = False
--пропуск--
156.            gameIsDone = False
157.           secretWord, secretSet = getRandomWord(words)
158.       else:
159.           break
```

В строке 120 возвращаемые функцией `getRandomWord(word)` значения присваиваются двум переменным — `secretWord` и `secretSet`. Код в строке 157 делает то же самое, когда игрок решает сыграть заново.

Выбор игроком категории слов

Последнее изменение, которое необходимо сделать — это сообщить игроку, к какому набору принадлежит слово для угадывания. Чтобы игрок знал, что секретное слово — это животное, цвет, фигура или фрукт. Ниже показан первоначальный код:

```
91. while True:
92.     displayBoard(missedLetters, correctLetters, secretWord)
```

В новой версии добавьте строку 124, чтобы код выглядел так:

```
123. while True:
124.     print('Секретное слово из набора: ' + secretSet)
125.     displayBoard(missedLetters, correctLetters, secretWord)
```

На этом изменения в программе «Виселица» заканчиваются. Вместо простого списка строковых переменных секретное слово выбирается из нескольких различных списков. Программа сообщает игроку, какой набор слов использован для выбора секретного слова. Попробуйте сыграть в новой версии. Вы можете легко изменить словарь слов, начинающийся в строке 48, чтобы еще увеличить число наборов слов.

Заключение

Написание кода игры «Виселица» завершено. В этой главе вы изучили несколько новых понятий, добавляя при этом в игру новые возможности. Даже написав программу, вы можете расширять ее функционал по мере изучения Python.

Словари похожи на списки, за исключением того, что словари могут использовать индексы произвольного типа, а не только целочисленные значения. Индексы словарей называются ключами. Многозначное присваивание — это рациональный способ присвоения различным переменным, значений, помещенных в список.

Игра «Виселица» стала значительно более продвинутой по сравнению с предыдущей версией. Теперь вам знакомы основные концептуальные понятия, необходимые при написании программ: переменные, циклы, функции, типы данных — список и словарь. Написание последующих программ этой книги по-прежнему будет сложной задачей, но вы завершили самую крутую часть восхождения!

10

ИГРА «КРЕСТИКИ-НОЛИКИ»

В этой главе представлена игра «Крестики-нолики». Обычно в нее играют два человека. Один игрок рисует крестики, другой — нолики. Игроки по очереди размещают свои знаки. Если игрок разместил на игровом поле три своих знака подряд — в один ряд, в один столбец или по диагонали, — он побеждает. Если поле заполнено, но ни один игрок не выиграл, игра заканчивается ничьей.

Эта глава не богата новыми концепциями. Пользователь будет играть против простого искусственного интеллекта, который мы создадим, используя уже имеющиеся навыки программирования. Искусственный интеллект (ИИ) — это компьютерная программа, способная разумно отвечать на действия игрока. ИИ, играющий в «Крестики-нолики», не слишком сложный: в действительности, это буквально несколько строк программного кода.

Давайте начнем с рассмотрения примера выполнения программы.

Игрок производит действие, вводя номер клетки, которую он хочет занять. Для облегчения запоминания порядка нумерации клеток пронумеруем их в том же порядке, что и цифровая клавиатура вашего компьютера, как показано на рис. 10.1.

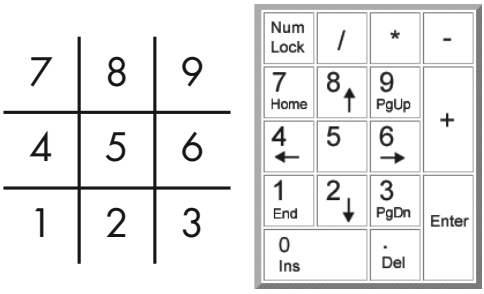


Рис. 10.1. Клетки игрового поля нумеруются так же, как цифровая клавиатура

В ЭТОЙ ГЛАВЕ РАССМАТРИВАЮТСЯ СЛЕДУЮЩИЕ ТЕМЫ:

- Искусственный интеллект
- Ссылки на список
- Вычисление по короткой схеме
- Значение `None`

Пример запуска игры «Крестики-нолики»

Вот что видит пользователь при запуске программы «Крестики-нолики». Текст, который вводит игрок, выделен полужирным шрифтом.

```
Игра "Крестики-нолики"
Вы выбираете X или O?
o
Человек ходит первым.
 | |
-+-+-
 | |
-+-+-
 | |
Ваш следующий ход? (1-9)
7
O| |X
-+-+-
 | |
-+-+-
 | |
Ваш следующий ход? (1-9)
4
O| |X
-+-+-
O| |
-+-+-
X| |
Ваш следующий ход? (1-9)
5
O| |X
-+-+-
O|O|
```

```
-+-+-
X| |X
Ваш следующий ход? (1-9)
6
O| |X
-+-+-
O|O|O
-+-+-
X| |X
Ура! Вы выиграли!
Сыграем еще раз? (да или нет)
нет
```

Исходный код игры «Крестики-нолики»

В редакторе файлов создайте новый файл, выбрав команду меню **File** ⇒ **New File** (Файл ⇒ Новый файл). В открывшемся окне введите приведенный ниже исходный код и сохраните файл под именем *tictactoe.py*. Затем запустите программу, нажав клавишу **F5**. Если при выполнении программы возникают ошибки, сравните код, который вы набрали, с оригинальным кодом с помощью онлайн-инструмента на сайте **inventwithpython.com/diff/**.

tictactoe.py

```
1. # Крестики-нолики
2.
3. import random
4.
5. def drawBoard(board):
6.     # Эта функция выводит на экран игровое поле, клетки которого будут заполняться.
7.
8.     # "board" — это список из 10 строк, для прорисовки игрового поля (индекс 0 игнорируется).
9.    print(board[7] + '|' + board[8] + '|' + board[9])
10.    print('-+-+-')
11.    print(board[4] + '|' + board[5] + '|' + board[6])
12.    print('-+-+-')
13.    print(board[1] + '|' + board[2] + '|' + board[3])
14.
```

```
15. def inputPlayerLetter():
16.     # Разрешение игроку ввести букву, которую он выбирает.
17.     # Возвращает список, в котором буква игрока — первый элемент, а буква компьютера — второй.
18.     letter = ''
19.     while not (letter == 'X' or letter == 'O'):
20.         print('Вы выбираете X или O?')
21.         letter = input().upper()
22.
23.     # Первым элементом списка является буква игрока, вторым — буква компьютера.
24.     if letter == 'X':
25.         return ['X', 'O']
26.     else:
27.         return ['O', 'X']
28.
29. def whoGoesFirst():
30.     # Случайный выбор игрока, который ходит первым.
31.     if random.randint(0, 1) == 0:
32.         return 'Компьютер'
33.     else:
34.         return 'Человек'
35.
36. def makeMove(board, letter, move):
37.     board[move] = letter
38.
39. def isWinner(bo, le):
40.  # Учитывая заполнение игрового поля и буквы игрока, эта функция возвращает True, если игрок выиграл.
41.   # Мы используем "bo" вместо "board" и "le" вместо "letter", поэтому нам не нужно много печатать.
42.     return ((bo[7] == le and bo[8] == le and bo[9] == le) or # across the top
43.     (bo[4] == le and bo[5] == le and bo[6] == le) or # через центр
44.     (bo[1] == le and bo[2] == le and bo[3] == le) or # через низ
45.     (bo[7] == le and bo[4] == le and bo[1] == le) or # вниз по левой стороне
46.     (bo[8] == le and bo[5] == le and bo[2] == le) or # вниз по центру
47.     (bo[9] == le and bo[6] == le and bo[3] == le) or # вниз по правой стороне
48.     (bo[7] == le and bo[5] == le and bo[3] == le) or # по диагонали
49.     (bo[9] == le and bo[5] == le and bo[1] == le)) # по диагонали
50.
51. def getBoardCopy(board):
52.     # Создает копию игрового поля и возвращает его.
53.     boardCopy = []
54.     for i in board:
55.         boardCopy.append(i)
```

```
56.     return boardCopy
57.
58. def isSpaceFree(board, move):
59.     # Возвращает True, если сделан ход в свободную клетку.
60.     return board[move] == ' '
61.
62. def getPlayerMove(board):
63.     # Разрешение игроку сделать ход.
64.     move = ' '
65.     while move not in '1 2 3 4 5 6 7 8 9'.split() or not isSpaceFree(board, int(move)):
66.         print('Ваш следующий ход? (1-9)')
67.         move = input()
68.     return int(move)
69.
70. def chooseRandomMoveFromList(board, movesList):
71.     # Возвращает допустимый ход, учитывая список сделанных ходов и список заполненных клеток.
72.     # Возвращает значение None, если больше нет допустимых ходов.
73.     possibleMoves = []
74.     for i in movesList:
75.         if isSpaceFree(board, i):
76.             possibleMoves.append(i)
77.
78.     if len(possibleMoves) != 0:
79.         return random.choice(possibleMoves)
80.     else:
81.         return None
82.
83. def getComputerMove(board, computerLetter):
84.  # Учитывая заполнение игрового поля и букву компьютера, определяет допустимый ход и возвращает его.
85.     if computerLetter == 'X':
86.         playerLetter = 'O'
87.     else:
88.         playerLetter = 'X'
89.
90.     # Это алгоритм для ИИ "Крестиков-Ноликов":
91.     # Сначала проверяем — победим ли мы, сделав следующий ход.
92.     for i in range(1, 10):
93.         boardCopy = getBoardCopy(board)
94.         if isSpaceFree(boardCopy, i):
95.             makeMove(boardCopy, computerLetter, i)
96.             if isWinner(boardCopy, computerLetter):
```

```
97.                 return i
98.
99.     # Проверяем — победит ли игрок, сделав следующий ход, и блокируем его.
100.     for i in range(1, 10):
101.         boardCopy = getBoardCopy(board)
102.         if isSpaceFree(boardCopy, i):
103.             makeMove(boardCopy, playerLetter, i)
104.             if isWinner(boardCopy, playerLetter):
105.                 return i
106.
107.     # Пробуем занять один из углов, если есть свободные.
108.     move = chooseRandomMoveFromList(board, [1, 3, 7, 9])
109.     if move != None:
110.         return move
111.
112.     # Пробуем занять центр, если он свободен.
113.     if isSpaceFree(board, 5):
114.         return 5
115.
116.     # Делаем ход по одной стороне.
117.     return chooseRandomMoveFromList(board, [2, 4, 6, 8])
118.
119. def isBoardFull(board):
120.     # Возвращает True, если клетка на игровом поле занята. В противном случае, возвращает False.
121.     for i in range(1, 10):
122.         if isSpaceFree(board, i):
123.             return False
124.     return True
125.
126.
127. print('Игра "Крестики-нолики"')
128.
129. while True:
130.     # Перезагрузка игрового поля
131.     theBoard = [' '] * 10
132.     playerLetter, computerLetter = inputPlayerLetter()
133.     turn = whoGoesFirst()
134.     print('' + turn + ' ходит первым.')
135.     gameIsPlaying = True
136.
```

```
137.     while gameIsPlaying:
138.         if turn == 'Человек':
139.             # Ход игрока.
140.             drawBoard(theBoard)
141.             move = getPlayerMove(theBoard)
142.             makeMove(theBoard, playerLetter, move)
143.
144.             if isWinner(theBoard, playerLetter):
145.                 drawBoard(theBoard)
146.                 print('Ура! Вы выиграли!')
147.                 gameIsPlaying = False
148.             else:
149.                 if isBoardFull(theBoard):
150.                     drawBoard(theBoard)
151.                     print('Ничья!')
152.                     break
153.                 else:
154.                     turn = 'Компьютер'
155.
156.         else:
157.             # Ход компьютера.
158.              move = getComputerMove(theBoard, computerLetter)
159.             makeMove(theBoard, computerLetter, move)
160.
161.             if isWinner(theBoard, computerLetter):
162.                 drawBoard(theBoard)
163.                  print('Компьютер победил! Вы проиграли.')
164.                 gameIsPlaying = False
165.             else:
166.                 if isBoardFull(theBoard):
167.                     drawBoard(theBoard)
168.                     print('Ничья!')
169.                     break
170.                 else:
171.                     turn = 'Человек'
172.
173.     print('Сыграем еще раз? (да или нет)')
174.     if not input().lower().startswith('д'):
175.         break
```

Проектирование программы

На рис. 10.2 показана блок-схема игры «Крестики-нолики». Программа начинается с предложения игроку выбрать букву «X» или «O». Кому принадлежит первый ход, выбирается случайно. Затем игрок и компьютер по очереди совершают свои ходы.

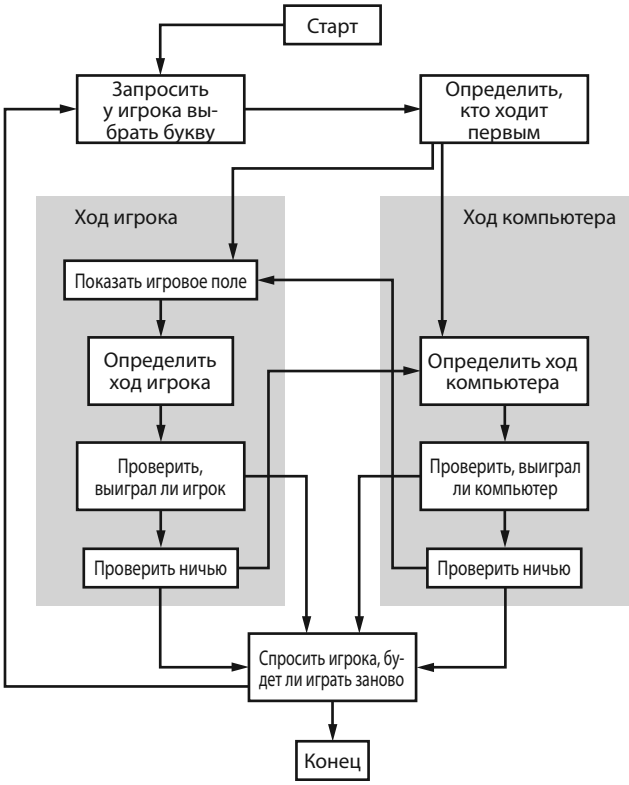


Рис. 10.2. Блок-схема программы «Крестики-нолики»

Клетки в левой части блок-схемы показывают, что происходит, когда игрок совершает ход. Клетки в правой части показывают, что происходит, когда ход совершает компьютер. После того как игрок или компьютер делают ход, программа проверяет, не выиграл ли кто-нибудь или не получилась ли ничья, далее возможны варианты. По завершении игры программа предлагает игроку сыграть заново.

Данные для прорисовки игрового поля

Сначала надо определить, как представить игровое поле данными, хранящимися в переменных. На бумаге поле для игры в «Крестики-нолики» вычерчивается парой горизонтальных и парой вертикальных линий, а де-

вять полученных клеток заполняются буквами «X» и «O». В программе «Крестики-нолики» игровое поле представлено списком строк, содержащих ASCII-символы, так же, как в игре «Виселица». В каждой строке представлена одна клетка игрового поля. Есть строки для буквы игрока «X», для буквы игрока «O», есть строка с единичным пробелом `' '` для пустых клеток.

Напомню, что размещение клеток игрового поля аналогично размещению клавиш на цифровой клавиатуре компьютера. Таким образом, если список из 10 строк хранится в переменной с именем `board`, то `board[7]` — это клетка в левом верхнем углу игрового поля, `board[8]` — верхняя средняя клетка, `board[9]` — верхняя правая клетка и так далее. Программа игнорирует индекс `0` списка. Чтобы сообщить программе, в какую клетку игрок хочет сделать ход, он должен ввести цифру от 1 до 9.

Стратегия игры ИИ

ИИ должен быть способен оценить состояние игрового поля и принять решение, в какую клетку сделать ход. Для удобства мы отметим три типа клеток: угловые, боковые и центр. Карта на рис. 10.3 показывает, где расположена каждая клетка.

Рис. 10.3. Расположение сторон, углов и центра клеток на поле

Стратегия ИИ игры в «Крестики-нолики» подчинена простому алгоритму — ограниченному числу инструкций, рассчитывающих результат. Одна программа может использовать несколько алгоритмов. Алгоритм можно представить блок-схемой. Алгоритм ИИ для игры в «Крестики-нолики» будет рассчитывать наилучший ход, как показано на рис. 10.4.

Алгоритм ИИ состоит из следующих шагов:

1. Проверить существует ли ход, сделав который, компьютер мог бы выиграть. Если существует, сделать его. В противном случае, перейти к шагу 2.
2. Проверить, существует ли ход, сделав который, игрок мог бы выиграть. Если существует, сделать ход для блокировки. В противном случае перейти к шагу 3.

3. Проверить, свободны ли какие-либо угловые клетки (под номером 1, 3, 7 или 9). Если да, то сделать ход в угол. Если нет свободных углов, перейти к шагу 4.
4. Проверить, свободен ли центр. Если да, то сделать ход в центр. Если нет, перейти к шагу 5.
5. Сделать ход в свободную боковую клетку (под номером 2, 4, 6 или 8). Если процесс выполнения алгоритма завершил 5-й шаг, возможных ходов больше не существует, так как все боковые клетки заняты.

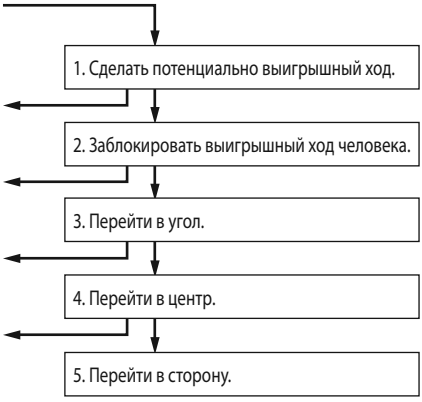


Рис. 10.4. Клетки представляют пять шагов алгоритма «Ход компьютера». Стрелки, указывающие налево, соответствуют указанию перейти к клетке «Проверить, выиграл ли компьютер» блок-схемы

Все это происходит в элементе «Определить ход компьютера» блок-схемы на рис. 10.2. Вы можете добавить клетки с этой информацией, изображенные на рис. 10.4, в блок-схему.

Этот алгоритм реализован в функции `getComputerMove()` и других, вызываемых ей, функциях.

Импорт модуля `random`

Первая пара строк содержит комментарий и инструкцию импорта модуля `random`. Таким образом, становится возможным вызов функции `randint()`.

```
1. # Крестики-нолики
2.
3. import random
```

С этими двумя понятиями вы уже встречались ранее, поэтому перейдем к следующей части программы.

Вывод игрового поля на экран

В следующей части кода определим функцию для прорисовки игрового поля.

```
5. def drawBoard(board):
6.     # Эта функция выводит на экран игровое поле, клетки которого будут заполняться.
7.
8.     # "board" — это список из 10 строк, для прорисовки игрового поля (индекс 0 игнорируется).
9.    print(board[7] + '|' + board[8] + '|' + board[9])
10.    print('-+-+-')
11.    print(board[4] + '|' + board[5] + '|' + board[6])
12.    print('-+-+-')
13.    print(board[1] + '|' + board[2] + '|' + board[3])
```

Функция `drawBoard()` выводит игровое поле, представленное переменной `board`. Напомню, что в переменной `board` хранится список из 10 строк, в котором строка с индексом `1` соответствует клетке 1 игрового поля «Крестики-нолики» и так далее.

Строка с индексом `0` игнорируется. Многие функции игры работают, передавая этот список как переменную `board`.

Не забудьте указать правильные расстояния в строках, иначе игровое поле будет выглядеть очень забавно при выводе на экран. Ниже продемонстрировано несколько примеров вызова функции `drawBoard()` с переменной `board` в качестве аргумента и несколько примеров того, что выводится на экран.

```
>>> drawBoard([' ', ' ', ' ', ' ', 'X', 'O', ' ', 'X', ' ', 'O'])
X| |
-+-+-
X|O|
-+-+-
 | |
>>> drawBoard([' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '])
 | |
-+-+-
 | |
-+-+-
 | |
```

Каждую строку программа размещает на игровом поле в порядке, соответствующем порядку цифровой клавиатуры компьютера, как показано на

рис. 10.1. Таким образом, первые три строки — это нижний ряд, следующие три строки — середина, и последние три строки — верхний ряд.

Предоставление игроку выбора между «X» или «O»

Далее определим функцию назначения игроку буквы «X» или «O».

```
15. def inputPlayerLetter():
16.     # Разрешение игроку ввести букву, которую он выбирает.
17.     # Возвращает список, в котором буква игрока — первый элемент, а буква компьютера — второй.
18.     letter = ''
19.     while not (letter == 'X' or letter == 'O'):
20.         print('Вы выбираете X или O?')
21.         letter = input().upper()
```

По завершении игры программа предлагает игроку сыграть заново. Условие цикла `while` содержит круглые скобки, это значит, что выражение в скобках вычисляется первым. Если переменной `letter` присваивается значение `'X'`, выражение вычисляется следующим образом:

```
not (letter == 'X' or letter == 'O')
     ↓                ↓
not ('X' == 'X'    or 'X' == 'O')
     ↓                ↓
not (True          or False)
     ↓
not (True)
↓
not True
↓
False
```

Если переменной `letter` присваивается значение `'X'` или `'O'`, то условие цикла становится ложным, что позволяет продолжить выполнение программы после блока `while`. Если условие истинно, программа продолжит предлагать игроку выбрать букву до тех пор, пока игрок не нажмет клавишу **X** или **O**. Код в строке 21 автоматически преобразует возвращаемое строковое значение переменной в прописные буквы. Делается это через вызов функции `input()` с помощью метода `upper()`.

Следующая функция возвращает список из двух элементов.

```
23.     # Первым элементом списка является буква игрока, вторым — буква компьютера.
24.     if letter == 'X':
25.         return ['X', 'O']
26.     else:
27.         return ['O', 'X']
```

Первый элемент (строковая переменная с индексом `0`) — это буква игрока, второй элемент (строка с индексом `1`) — буква компьютера. Инструкции `if` и `else` выбирают соответствующий список.

Выбор — кто будет ходить первым

Далее создаем функцию, которая использует `randnt()`, для выбора того, кто будет ходить первым.

```
29. def whoGoesFirst():
30.     # Случайный выбор игрока, который ходит первым.
31.     if random.randint(0, 1) == 0:
32.         return 'Компьютер'
33.     else:
34.         return 'Человек'
```

Функция `whoGoesFirst()` — виртуальный аналог подброшенной монетки, для определения того, кто будет ходить первым. Вместо подбрасываний монетки вызываем функцию `random.randint(0, 1)`. Шансы 50 на 50, что функция вернет `0` или `1`. Если эта функция возвращает `0`, функция `whoGoesFirst()` возвращает строку `'Компьютер'`. В противном случае функция вернет строку `'Человек'`. Код, вызвавший эту функцию, использует возвращенное значение для определения того, кто сделает первый ход.

Размещение меток на игровом поле

Функция `makeMove()` очень проста.

```
36. def makeMove(board, letter, move):
37.     board[move] = letter
```

Параметрами являются переменные board, letter и move. Переменная board — это список из 10 строк, представляющих состояние игрового поля. Переменная letter — это буква игрока («X» или «O»). Переменная move — это клетка, на которую игрок пожелал сделать ход (целочисленная переменная в диапазоне от 1 до 9).

Стоп! Но ведь, кажется, в строке 37 этот код заменяет значение одного из элементов списка board значением переменной letter. Так как этот код принадлежит функции, то, может быть, изменяемый параметр будет утерян при выходе из функции. Стоит ли вносить изменения, которые будут отменены?

В действительности это не так, потому что списки, которые передаются в функцию как аргументы, — особенные. Передаются не сами списки, а *ссылки* на них. Рассмотрим разницу между списками и ссылками на списки.

Ссылки на список

В интерактивной оболочке введите следующие команды:

```
>>> spam = 42
>>> cheese = spam
>>> spam = 100
>>> spam
100
>>> cheese
42
```

Переменной spam присваивается значение 42, затем значение spam передается переменной cheese. Потом, когда переменной spam присваивается значение 100, это не оказывает никакого влияния на переменную cheese. Так происходит, потому что spam и cheese — разные переменные и хранят разные значения. Но списки работают не так. Когда список сохраняется в переменной, в действительности в переменную передается не сам список, а ссылка на него. Ссылка — это некоторая величина, которая указывает, где хранится информация. Рассмотрим фрагмент кода, который упростит понимание этого факта. Наберите следующее:

```
u >>> spam = [0, 1, 2, 3, 4, 5]
v >>> cheese = spam
w >>> cheese[1] = 'Привет!'
>>> spam
[0, 'Привет!', 2, 3, 4, 5]
```

```
>>> cheese
[0, 'Привет!', 2, 3, 4, 5]
```

В коде изменяется только список cheese, но, кажется, изменились оба списка — cheese и spam. Это произошло потому, что переменная spam не содержит значений списка как таковых, а только ссылки на них, как показано на рис. 10.5. Списки не содержатся в переменных, а существуют совершенно самостоятельно.

```
❶ spam = [0, 1, 2, 3, 4, 5]
```

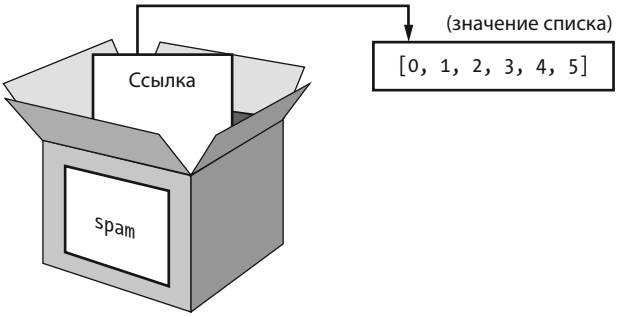


Рис. 10.5. Список spam, созданный на шаге ❶. Переменная хранит не список, а ссылку на него

Замечу, что выражение cheese = spam копирует из spam в cheese *ссылку на список*, а не сам список. Теперь обе переменные, spam и cheese, хранят ссылки, которые указывают на одни и те же значения списка. Но существует только один список, который никуда не копируется. На рис. 10.6 показан механизм такого копирования.

```
❷ cheese = spam
```

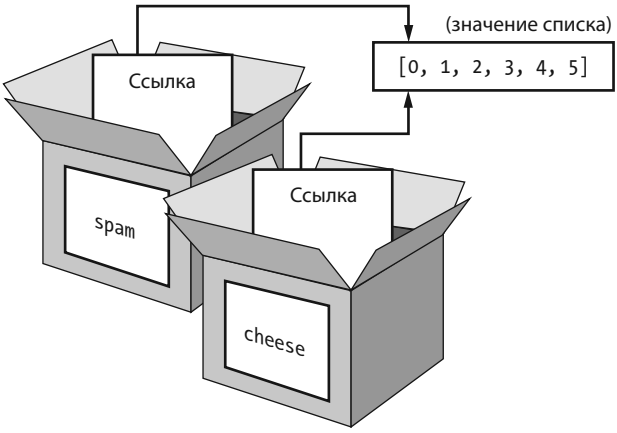


Рис. 10.6. Переменные spam и cheese хранят ссылки на один список

Итак, выражение `cheese[1] = 'Привет!'` в ❸ изменяет тот же список, на который ссылается переменная `spam`. Вот почему переменная `spam` возвращает то же самое значение, что и переменная `cheese`. Они обе содержат ссылки, указывающие на один список, как показано на рис. 10.7.

```
❸ cheese[1] = 'Привет!'
```

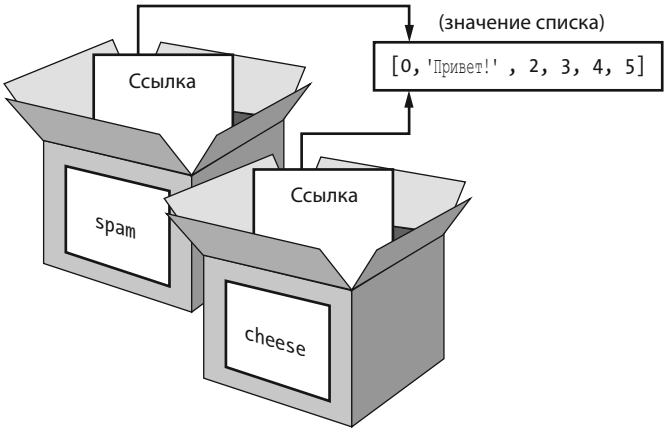


Рис. 10.7. Изменения списка затрагивают все переменные, содержащие ссылку на этот список

Если необходимо сохранить в переменных `spam` и `cheese` разные списки, то надо создать два списка, а не копировать ссылки на один и тот же.

```
>>> spam = [0, 1, 2, 3, 4, 5]
>>> cheese = [0, 1, 2, 3, 4, 5]
```

В приведенном примере переменные `spam` и `cheese` хранят два разных списка (путь даже списки идентичны по содержанию). Теперь, если модифицировать один список, это не произведет никакого эффекта на другой.

```
>>> spam = [0, 1, 2, 3, 4, 5]
>>> cheese = [0, 1, 2, 3, 4, 5]
>>> cheese[1] = 'Привет!'
>>> spam
[0, 1, 2, 3, 4, 5]
>>> cheese
[0, 'Привет!', 2, 3, 4, 5]
```

Рисунок 10.8 демонстрирует, как в этом примере переменные взаимодействуют со значениями списков.

Словари работают аналогичным образом. Переменные хранят не словари, а лишь *ссылки* на них.

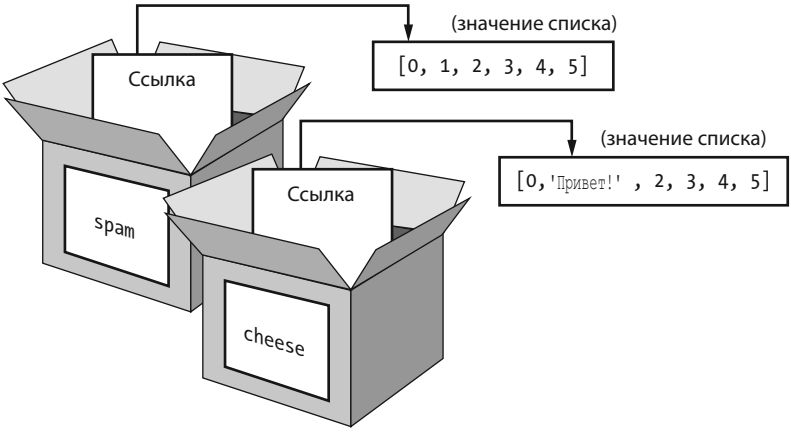


Рис. 10.8. Теперь переменные `spam` и `cheese` хранят ссылки на разные списки

Использование ссылок на списки в функции `makeMove()`

Вернемся к функции `makeMove()`.

```
36. def makeMove(board, letter, move):
37.     board[move] = letter
```

Когда список передается в переменную `board`, локальная переменная функции в действительности копирует ссылку на список, не сам список. Поэтому любые изменения в `board` в коде функции, будут применены к самому списку. Несмотря на то, что `board` — локальная переменная, функция `makeMove()` модифицирует сам список.

А вот переменные `letter` и `move` – это копии передаваемых строкового и целочисленного параметров. Поэтому, если внутри функции `makeMove()` модифицировать `letter` или `move`, это не приведет к изменению исходных значений переменных.

Проверка — не победил ли игрок

Строки 42–49 в функции `isWinner()` — это, по сути, длинная инструкция `return`.

```
39. def isWinner(bo, le):
40.   # Учитывая заполнение игрового поля и буквы игрока, эта функция возвращает True, если игрок выиграл.
```

```
41.   # Мы используем "bo" вместо "board" и "le" вместо "letter", поэтому нам не нужно много печатать.
42.     return ((bo[7] == le and bo[8] == le and bo[9] == le) or # across the top
43.     (bo[4] == le and bo[5] == le and bo[6] == le) or # через центр
44.     (bo[1] == le and bo[2] == le and bo[3] == le) or # через низ
45.     (bo[7] == le and bo[4] == le and bo[1] == le) or # вниз по левой стороне
46.     (bo[8] == le and bo[5] == le and bo[2] == le) or # вниз по центру
47.     (bo[9] == le and bo[6] == le and bo[3] == le) or # вниз по правой стороне
48.     (bo[7] == le and bo[5] == le and bo[3] == le) or # по диагонали
49.     (bo[9] == le and bo[5] == le and bo[1] == le)) # по диагонали
```

Слова `bo` и `le` — короткая форма имен `board` и `letter`. Короткая форма записи позволяет меньше печатать. Напомню, что интерпретатору Python безразлично, как вы назовете переменные.

Существуют восемь вариантов победы в игре «Крестики-нолики»: можно провести линию через верхний, средний или нижний ряды; можно провести линию вдоль левого, среднего или правого столбцов; или можно провести линию по одной из диагоналей.

Каждая строка условия проверяет, равны ли все три значения клеток, образующих линию, значению `letter` (в комбинации с оператором `and`). Между собой строки объединяются оператором `or`, чтобы проверить все варианты победы. Это значит, что лишь одна строка должна быть истинна, чтобы объявить победу игрока, чья буква содержится в переменной `letter`.

Пусть значение переменной `le` равно `'O'`, а значение переменной `bo` равно `[' ', 'O', 'O', 'O', ' ', 'X', ' ', 'X', ' ', ' ']`. Содержимое переменной `board` будет выглядеть так:

```
X| |
-+-+-
 |X|
-+-+-
O|O|O
```

Здесь приведена иллюстрация того, как будет преобразовано выражение после ключевого слова `return` в строке 42. Сначала Python замещает переменные `bo` и `le` их значениями.

```
return (('X' == 'O' and ' ' == 'O' and ' ' == 'O') or
(' ' == 'O' and 'X' == 'O' and ' ' == 'O') or
('O' == 'O' and 'O' == 'O' and 'O' == 'O') or
('X' == 'O' and ' ' == 'O' and 'O' == 'O') or
```

```
(' ' == 'O' and 'X' == 'O' and 'O' == 'O') or
(' ' == 'O' and ' ' == 'O' and 'O' == 'O') or
('X' == 'O' and 'X' == 'O' and 'O' == 'O') or
(' ' == 'O' and 'X' == 'O' and 'O' == 'O'))
```

Затем Python вычисляет все инструкции сравнения == в скобках и приводит к логическим значениям.

```
return ((False and False and False) or
(False and False and False) or
(True and True and True) or
(False and False and True) or
(False and False and True) or
(False and False and True) or
(False and False and True) or
(False and False and True))
```

После чего интерпретатор вычисляет значения выражений в круглых скобках.

```
return ((False) or
(False) or
(True) or
(False) or
(False) or
(False) or
(False) or
(False))
```

И наконец, после того как в круглых скобках останется по одному значению, от них можно избавиться:

```
return (False or
False or
True or
False or
False or
False or
False or
False)
```

Теперь Python вычисляет общее значение выражения с операторами `or`.

```
return (True)
```

И еще раз избавляется от скобок, оставляя одно значение.

```
return True
```

Таким образом, для выбранных значений переменных `bo` и `le` будет возвращено значение `True`. Так программа может определить, выиграл ли один из игроков.

Дублирование данных игрового поля

Функция `getBoardCopy()` позволяет создать копию 10-строкового списка, представляющего игровое поле в игре «Крестики-нолики».

```
51. def getBoardCopy(board):
52.     # Создает копию игрового поля и возвращает его.
53.     boardCopy = []
54.     for i in board:
55.         boardCopy.append(i)
56.     return boardCopy
```

Когда алгоритм ИИ планирует ход от лица компьютера, бывает нужно модифицировать временную копию игрового поля, не изменяя исходного. В таких случаях создается копия списка `board` посредством вызова этой функции. Новый список создается в строке 53. Во-первых, сохраняется пустой список в переменной `boardCopy`. Затем в цикле `for` производится перебор всех элементов `board` и копии всех строк добавляются в дубликат. После того как функция `getBoardCopy()` завершит построение копии текущих значений списка `board`, она передает в переменную `boardCopy` ссылку на новый список, а не на исходный.

Проверка — свободна ли клетка игрового поля

Простая функция `isSpaceFree()` определяет возможность хода в соответствии с ситуацией на игровом поле.

```
58. def isSpaceFree(board, move):
59.     # Возвращает True, если сделан ход в свободную клетку.
60.     return board[move] == ' '
```

Напомню, что свободные клетки в списке board — это строковые переменные, содержащие единичный пробел. Если какой-либо элемент с индексом пустой клетки не равен ' ', то он замещается пробелом.

Разрешение игроку сделать ход

Функция getPlayerMove() предлагает игроку ввести номер клетки, в которую он хочет сделать ход.

```
62. def getPlayerMove(board):
63.     # Разрешение игроку сделать ход.
64.     move = ' '
65.     while move not in '1 2 3 4 5 6 7 8 9'.split() or not isSpaceFree(board, int(move)):
66.         print('Ваш следующий ход? (1-9)')
67.         move = input()
68.     return int(move)
```

Условие в строке 65 принимает значение True, если выражение с правой или левой стороны оператора or истинно. Цикл гарантирует, что выполнение программы не будет продолжено, пока пользователь не введет целое число в диапазоне от 1 до 9. Заодно проверяется, что затребованная клетка свободна с учетом передаваемых в функцию параметров, определяющих состояние игрового поля. Две строки кода цикла while просто предлагают игроку ввести число от 1 до 9.

Выражение в левой части проверяет, ввел ли игрок значение, равное '1', '2', '3' и так далее до '9', создавая список этих строк (с помощью метода split()) и проверяя, содержится ли значение move в этом списке. Выражение '1 2 3 4 5 6 7 8 9'.split() эквивалентно выражению ['1', '2', '3', '4', '5', '6', '7', '8', '9'], просто форма более удобна для набора кода.

Выражение в правой части оператора проверяет, свободна ли клетка, затребованная пользователем, вызывая функцию isSpaceFree(). Напомню, что функция isSpaceFree() возвращает значение True, если предложенный игроком ход допустим. Необходимо отметить, что isSpaceFree() ожидает, что переменная move содержит целочисленное значение, поэтому используется функция int(), которая преобразует содержимое переменной move к целому типу.

Оператор not присутствует с обеих сторон, поэтому условие возвращает значение True, даже если не выполняется одно из требований. По этой причине цикл повторяет предложение игроку снова и снова, до тех пор, пока не будет сделан допустимый ход.

В финале код в строке 68 возвращает целочисленное значение любой введенной строки. Функция input() возвращает строку, затем функция int() преобразует строку в целое.

Вычисление по короткой схеме

Вы могли заметить, что при вызове функции `getPlayerMove()` возможны проблемы. Что произойдет, если игрок введет, например `'Z'`, или еще что-нибудь, вместо цифры? Выражение `move not in '1 2 3 4 5 6 7 8 9'.split()` в левой части оператора `or` вернет `False`, а затем Python перейдет к обработке выражения в правой части оператора `or`.

Но вызов функции `int('Z')` приведет к возникновению ошибки, так как эта функция может обрабатывать только строки типа `'9'` или `'0'`, но не такие строки как `'Z'`.

Чтобы увидеть пример такой ошибки, наберите в интерактивной оболочке следующий код:

```
>>> int('42')
42
>>> int('Z')
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    int('Z')
ValueError: invalid literal for int() with base 10: 'Z'
```

Однако, если играя в «Крестики-нолики», вы введете `'Z'` в качестве своего хода, этой ошибки не возникнет. Все потому, что условие цикла `while` вычисляется по *короткой схеме*.

Принцип заключается в том, что вычисляется только одна часть выражения, а оставшаяся часть не влияет на полученный результат. Ниже приведен краткий, но хороший пример вычисления по короткой схеме. Наберите следующий код:

```
>>> def ReturnsTrue():
print('Была вызвана функция ReturnsTrue().')
return True
>>> def ReturnsFalse():
print('Была вызвана функция ReturnsFalse().')
return False
>>> ReturnsTrue()
Была вызвана функция ReturnsTrue().
True
>>> ReturnsFalse()
Была вызвана функция ReturnsFalse().
False
```

Когда вызывается функция `ReturnsTrue()`, на экран выводится текст `'Была вызвана функция ReturnsTrue().'`, а затем выводится значение, возвращаемое функцией `ReturnsTrue()`. То же самое происходит при вызове функции `ReturnsFalse()`. Теперь В интерактивной оболочке введите следующие команды:

```
>>> ReturnsFalse() or ReturnsTrue()
Была вызвана функция ReturnsFalse().
Была вызвана функция ReturnsTrue().
True
>>> ReturnsTrue() or ReturnsFalse()
Была вызвана функция ReturnsTrue().
True
```

В первой части происходит следующее: выражение `ReturnsFalse() or ReturnsTrue()` вызывает обе функции, поэтому выводятся оба сообщения.

А вот вторая часть выводит только сообщение `'Была вызвана функция ReturnsTrue().'` и не выводит `'Была вызвана функция ReturnsFalse().'`. Так происходит потому, что Python вообще не вызывает функцию `ReturnsFalse()`. Так как левая часть оператора `or` истинна, уже не имеет значения, что вернет функция `ReturnsFalse()` в правой части, поэтому Python даже не заботится о ее вызове. Это преобразование и есть вычисление по короткой схеме.

То же самое применимо и к оператору `and`. Наберите следующий код:

```
>>> ReturnsTrue() and ReturnsTrue()
Была вызвана функция ReturnsTrue().
Была вызвана функция ReturnsTrue().
True
>>> ReturnsFalse() and ReturnsFalse()
Была вызвана функция ReturnsFalse().
False
```

Если левая часть оператора `and` ложна, то и все выражение ложно. И уже не имеет значения, какое значение вернет правая часть, — `True` или `False`, так как Python просто не будет его определять. Выражения `False and True` и `False and False` ложны и представляют собой вычисления по короткой схеме в языке Python.

Вернемся к строкам 65–68 игры «Крестики-нолики».

```
65.     while move not in '1 2 3 4 5 6 7 8 9'.split() or not isSpaceFree(board, int(move)):
66.         print('Ваш следующий ход? (1-9)')
67.         move = input()
68.     return int(move)
```

Так как левая часть оператора or (move not in '1 2 3 4 5 6 7 8 9'.split()) истинна, то все выражение истинно. Не важно, какое именно значение вернет правая часть, — True или False, так как достаточно, чтобы одна сторона оператора or была истинной для истинности всего выражения.

Таким образом, Python прекращает дальнейшую проверку и даже не вычисляет значение выражения not isSpaceFree(board, int(move)). Это значит, что функции int() и isSpaceFree() не будут вызваны до тех пор, пока выражение move not in '1 2 3 4 5 6 7 8 9'.split() будет оставаться истинным.

Это хорошо применимо к нашей программе, потому что если левая часть оператора истинна, значит значение move — это не строка с одной цифрой. И значит, она стала бы причиной ошибки при вызове функции int(). Но если выражение move not in '1 2 3 4 5 6 7 8 9'.split() истинно, интерпретатор, проводя вычисление по короткой схеме, не определяет значение выражения not isSpaceFree(board, int(move)) и не вызывает int(move).

Выбор хода из списка

Рассмотрим функцию chooseRandomMoveFromList(), которая пригодится нам далее, в программе ИИ.

```
70. def chooseRandomMoveFromList(board, movesList):
71.     # Возвращает допустимый ход, учитывая список сделанных ходов и список заполненных клеток.
72.     # Возвращает значение None, если больше нет допустимых ходов.
73.     possibleMoves = []
74.     for i in movesList:
75.         if isSpaceFree(board, i):
76.             possibleMoves.append(i)
```

Напомню, что переменная board — это список строк, в которых представлено игровое поле игры «Крестики-нолики». Вторая переменная, movesList, — это список целых чисел, из которого выбираются свободные клетки для совершения хода. Например, если переменная movesList имеет значение [1, 3, 7, 9], значит функция chooseRandomMoveFromList() должна вернуть одно из значений угловых клеток.

Но сначала функция chooseRandomMoveFromList() проверит, что клетки доступны для совершения хода. Список possibleMoves создается пустым. Затем в цикле for производится перебор значений списка movesList. Те значения, для которых функция isSpaceFree() возвращает True, добавляются в список possibleMoves с помощью метода append().

На этом этапе список possibleMoves содержит все клетки, представленные в movesList, как пустые. Затем программа проверяет, пуст ли список.

```
78.     if len(possibleMoves) != 0:
79.         return random.choice(possibleMoves)
80.     else:
81.         return None
```

Если список не пуст, то есть хотя бы один возможный ход, который можно сделать на поле.

Но этот список может быть и пустым. Например, если переменная movesList имеет значение [1, 3, 7, 9], но все угловые клетки игрового поля, представленные в переменной board, уже заняты, список possibleMoves будет равен []. В этом случае len(possibleMoves) принимает значение 0, и функция возвращает значение None.

Значение None

Значение None представляет собой отсутствие значения. None — единственное значение типа данных NoneType. Это значение можно использовать, когда необходимо получить ответ типа «не существует» или «ни один из числа, перечисленных выше».

Например, в переменной с именем quizAnswer хранится ответ пользователя на какой-нибудь вопрос популярной викторины в формате истина/ложь. То есть переменная возвращает значение True или False, оценивая ответ пользователя. Но если пользователь не отвечает на вопрос, значение True/False использовать нельзя, так как это означало бы ответ пользователя. Вместо этого переменной quizAnswer можно присвоить значение None, если пользователь пропускает ответ.

Необходимо отметить, что значение None не выводится в интерактивной оболочке, подобно другим значениям.

```
>>> 2 + 2
4
>>> 'Это строковое значение.'
'Это строковое значение.'
>>> None
>>>
```

Значения первых двух выражений выводятся на экран в последующих строках, но None не имеет значения, поэтому не выводится ничего.

Функция, которая, как кажется, не имеет возвращаемого значения, возвращает значение None. Например, print() возвращает значение None.

```
>>> spam = print('Привет, мир!')
Привет, мир!
>>> spam == None
True
```

Здесь мы присваиваем переменной spam значение print('Привет, мир!'). Подобно другим функциям, функция print() возвращает значение. Несмотря на то что print() — функция вывода на экран, при вызове она возвращает значение None. IDLE не показывает значение None в интерактивной оболочке, но можно сказать, что spam присвоено значение None, потому что выражение spam == None истинно.

Создание искусственного интеллекта

Код ИИ содержится в теле функции getComputerMove().

```
83. def getComputerMove(board, computerLetter):
84.  # Учитывая заполнение игрового поля и букву компьютера, определяет допустимый ход и возвращает его.
85.     if computerLetter == 'X':
86.         playerLetter = 'O'
87.     else:
88.         playerLetter = 'X'
```

Первый аргумент, переменная board, — это параметры игрового поля игры «Крестики-нолики». Второй аргумент, переменная letter, — это буква компьютера — 'X' или 'O', которая хранится в переменной computerLetter. Первые несколько строк просто присваивают другую букву переменной playerLetter. Код один и тот же, присвоены компьютеру буквы «X» или «O».

Напомню, как работает алгоритм искусственного интеллекта игры «Крестики-нолики»:

1. Проверить, существует ли ход, сделав который, компьютер победит. Если существует, сделать его. В противном случае перейти к шагу 2.
2. Проверить, существует ли ход, сделав который, победит игрок. Если существует, блокировать его. В противном случае перейти к шагу 3.
3. Проверить, свободна ли одна из угловых клеток (под номером 1, 3, 7 или 9). Если да, то сделать ход в эту клетку. Если нет свободных угловых клеток, перейти к шагу 4.

4. Проверить, свободен ли центр. Если свободен, занять его. Если нет, перейти к шагу 5.
5. Сделать ход на одну из боковых клеток (под номером 2, 4, 6 или 8). Если этот шаг выполнен, возможных ходов больше нет.

Функция возвращает целочисленные значения в диапазоне от 1 до 9, представляющие собой ходы компьютера. Давайте посмотрим, как каждый из этих шагов реализуется в коде программы.

Проверка — сможет ли компьютер победить, сделав ход

Прежде всего, если компьютер может победить, сделав следующий ход, его надо сделать немедленно.

```
90.     # Это алгоритм для ИИ "Игры «Крестики-Нолики»":
91.     # Сначала проверяем — победим ли мы, сделав следующий ход.
92.     for i in range(1, 10):
93.         boardCopy = getBoardCopy(board)
94.         if isSpaceFree(boardCopy, i):
95.             makeMove(boardCopy, computerLetter, i)
96.             if isWinner(boardCopy, computerLetter):
97.                 return i
```

Цикл `for`, который начинается в строке 92, перебирает все возможные значения ходов от 1 до 9. Код внутри цикла имитирует ситуацию, которая возникнет, если компьютер сделает этот ход.

Код в первой строке в цикле (строка 93) создает копию списка `board`. Таким образом, внутри цикла имитируется ход, не изменяя реальных значений игрового поля «Крестики-нолики», которые хранятся в переменной `board`. Функция `getBoardCopy()` возвращает другой список, хотя и идентичный списку `board`.

Код в строке 93 проверяет, свободна ли требуемая клетка, и делает ход в копии `board`. Если в результате компьютер побеждает, функция возвращает этот ход в виде целого значения.

Если нет клеток, приводящих к победе, цикл завершается, и выполнение программы продолжается со строки 100.

Проверка — сможет ли игрок победить, сделав ход

Затем для каждой клетки код имитирует ход человека.

```
99.     # Проверяем — победит ли игрок, сделав следующий ход, и блокируем его.
100.     for i in range(1, 10):
```

```
101.         boardCopy = getBoardCopy(board)
102.         if isSpaceFree(boardCopy, i):
103.             makeMove(boardCopy, playerLetter, i)
104.             if isWinner(boardCopy, playerLetter):
105.                 return i
```

Код такой же, как в цикле со строки 92, исключая букву игрока, помещенную в копию board. Если функция isWinner() показывает, что игрок может победить в следующем ходу, компьютер блокирует этот ход, чтобы этого не допустить. Если в следующем ходу победа игрока невозможна, цикл for завершается, и выполнение продолжается со строки 108.

Проверка угловых, центральной и боковых клеток (в порядке очереди)

Если компьютер не может сделать победный ход и нет необходимости блокировать ход игрока, ход совершается в угловую, центральную или боковую клетку, в зависимости от того, какие из них свободны.

Сначала компьютер пытается сделать ход в одну из угловых клеток.

```
107.     # Пробуем занять один из углов, если есть свободные.
108.     move = chooseRandomMoveFromList(board, [1, 3, 7, 9])
109.     if move != None:
110.         return move
```

Вызов функции chooseRandomMoveFromList() со списком [1, 3, 7, 9] гарантирует, что функция вернет целочисленное значение угловой клетки: 1, 3, 7 или 9. Если все угловые клетки заняты, функция chooseRandomMoveFromList() возвращает значение None, и программа переходит к строке 113.

```
112.     # Пробуем занять центр, если он свободен.
113.     if isSpaceFree(board, 5):
114.         return 5
```

Если нет свободных угловых клеток, код в строке 114 совершает ход в центр, если он свободен. Если центр занят, выполняется код в строке 117.

```
116.     # Делаем ход по одной стороне.
117.     return chooseRandomMoveFromList(board, [2, 4, 6, 8])
```

Этот код тоже вызывает функцию chooseRandomMoveFromList(), только ей передается список [2, 4, 6, 8]. Эта функция не возвращает значения None, потому что боковые клетки — последние из возможных. На этом заканчивается выполнение функции getComputerMove() и алгоритма ИИ.

Проверка — заполнено ли поле

И последняя функция — isBoardFull().

```
119. def isBoardFull(board):
120.     # Возвращает True, если клетка на игровом поле занята. В противном случае, возвращает
False.
121.     for i in range(1, 10):
122.         if isSpaceFree(board, i):
123.             return False
124.     return True
```

Эта функция возвращает True, если в переменную board для элементов всех индексов (исключая индекс 0, который просто игнорируется) в качестве аргументов переданы буквы 'X' или 'O'. Цикл for позволяет проверить все индексы списка board, от 1 до 9. Как только в списке board будет найдена свободная клетка (когда isSpaceFree(board, i) вернет True), функция isBoardFull() вернет False.

Если произведены все итерации цикла, значит, свободных клеток не осталось. Строка 124 вернет значение True.

Игровой цикл

Строка 127 — первая, которая не принадлежит коду функций, то есть это первая исполняемая строка программы.

```
127. print('Игра "Крестики-нолики"')
```

Эта строка приветствия перед началом игры. Затем программа переходит к выполнению цикла while в строке 129.

```
129. while True:
130.     # Перезагрузка игрового поля
131.     theBoard = [' '] * 10
```

Цикл while выполняется до тех пор, пока управление не будет передано инструкции break. Код в строке 131 сохраняет основное игровое поле игры «Крестики-нолики» в переменной theBoard.

Сначала выводится пустое игровое поле, представленное списком из 10 пустых строк. Вместо того чтобы вводить этот список целиком, строка 131 использует репликацию списка. Проще ввести [' '] * 10, чем [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '].

Выбор буквы игрока и того, кто будет ходить первым

Далее функция inputPlayerLetter() предлагает игроку ввести букву, которую он выбирает, «X» или «O» .

```
132.    playerLetter, computerLetter = inputPlayerLetter()
```

Функция возвращает список из двух элементов, ['X', 'O'] или ['O', 'X']. Операцией множественного присваивания переменной playerLetter присваивается значение первого элемента, а переменной computerLetter — второго.

Затем функция whoGoesFirst() случайным образом выбирает, кому принадлежит первый ход, возвращая строку 'Человек' или 'Компьютер', и в строке 137 сообщает игроку о своем выборе.

```
133.    turn = whoGoesFirst()
134.    print('' + turn + ' ходит первым.')
135.    gameIsPlaying = True
```

Переменная gameIsPlaying хранит информацию о состоянии сеанса игры — продолжается ли игра, или победил один из участников, или вышла ничья.

Переменная turn в значении 'Человек'

Цикл в строке 137 будет метаться между кодом хода игрока и кодом хода компьютера до тех пор, пока переменной gameIsPlaying присвоено значение True.

```
137.    while gameIsPlaying:
138.        if turn == 'Человек':
139.            # Ход игрока.
140.            drawBoard(theBoard)
141.            move = getPlayerMove(theBoard)
142.            makeMove(theBoard, playerLetter, move)
```

Переменной turn уже было присвоено значение 'Человек' или 'Компьютер' при вызове функции whoGoesFirst() в строке 133. Если turn присвоено значение 'Компьютер', то условие в строке 138 ложно и управление переходит к коду в строке 156.

Но если условие этой строки истинно, код в строке 140 вызывает функцию drawBoard() и передает значение переменной theBoard для вывода на экран в игровом поле Игры «Крестики-нолики». Затем функция getPlayerMove() предлагает игроку сделать ход и проверяет его допустимость. Функция makeMove() добавляет в переменную theBoard букву игрока «X» или «O».

После совершения игроком хода программа должна проверить, привел ли этот ход к победе.

```
144.            if isWinner(theBoard, playerLetter):
145.                drawBoard(theBoard)
146.                print('Ура! Вы выиграли!')
147.                gameIsPlaying = False
```

Если функция isWinner() возвращает значение True, код из блока if выводит на экран сообщение о победе игрока. Значение переменной gameIsPlaying становится равным False, и перехода к совершению хода компьютером не происходит. Если игрок не побеждает при последнем своем ходе, то, может быть, поле заполнено с результатом — ничья. Эта вероятность проверяется в инструкции else.

```
148.            else:
149.                if isBoardFull(theBoard):
150.                    drawBoard(theBoard)
151.                    print('Ничья!')
152.                    break
```

В блоке else функция isBoardFull() возвращает значение True, если больше нет возможных ходов. В этом случае код из блока if в строке 149 выводит панель с сообщением игроку о ничьей. Программа выходит из цикла while и переходит к строке 173.

Если не произошла ни победа игрока, ни ничья, программа переходит к следующей инструкции else.

```
153.                else:
154.                    turn = 'Компьютер'
```

В строке 154 переменной turn присваивается значение 'Компьютер' и в следующей итерации программа выполняет код, который совершает ход компьютера.

Переменная turn в значении 'Компьютер'

Если в условии строки 138 значение переменной turn не равно 'Человек', значит, оно должно быть равно 'Компьютер'. Код в этом блоке else такой же, как и для переменной turn в значении 'Человек'.

```
156.        else:
157.            # Ход компьютера.
158.             move = getComputerMove(theBoard, computerLetter)
159.            makeMove(theBoard, computerLetter, move)
160.
161.            if isWinner(theBoard, computerLetter):
162.                drawBoard(theBoard)
163.                 print('Компьютер победил! Вы проиграли.')
164.                gameIsPlaying = False
165.            else:
166.                if isBoardFull(theBoard):
167.                    drawBoard(theBoard)
168.                    print('Ничья!')
169.                    break
170.                else:
171.                    turn = 'Человек'
```

Строки 157–171 почти идентичны строкам 139–154. Единственное отличие в том, что используется буква компьютера и вызывается функция getComputerMove().

Если нет победителя или ничьей, код в строке 171 присваивает переменной turn значение 'Человек'. В этом цикле while больше нет строк, поэтому управление передается инструкции while в строке 137.

Предложение игроку сыграть заново

В финале игры программа спрашивает игрока, хочет ли он сыграть еще раз.

```
173.    print('Сыграем еще раз? (да или нет)')
174.    if not input().lower().startswith('д'):
175.        break
```

Код в строках 173–175 выполняется сразу после старта цикла в строке 137. Переменной `gameIsPlaying` присваивается значение `False`, когда завершается игра. В этот момент программа спрашивает игрока, не хочет ли он сыграть еще раз.

Выражение `not input().lower().startswith('д')` принимает значение `True`, если игрок введет какую-либо строку, не начинающуюся с буквы `'д'`. В этом случае будет выполнена инструкция `break`. Эта инструкция прерывает выполнение цикла `while`, который был запущен кодом в строке 129. Но так как в программе больше нет строк для выполнения, происходит выход из программы и завершение игры.

Заключение

Создание программы с искусственным интеллектом сводится к тщательному рассмотрению всех возможных ситуаций, с которыми он может столкнуться, и вариантов его реакции на каждую из этих ситуаций. Искусственный интеллект игры «Крестики-нолики» прост, потому что в игре немного возможных ходов, по сравнению, скажем, с шахматами или шашками.

Сначала наш компьютерный ИИ проверяет, есть ли у него победный ход. Затем проверяет необходимость блокировки ходов игрока. Далее ИИ просто выбирает свободную угловую клетку, потом центральную, а затем боковую. Это простой алгоритм для совершения хода.

Ключ к реализации ИИ заключается в создании копии данных игрового поля и моделирования ходов в этой копии. Таким образом, ИИ может видеть, приведет ли ход к победе или поражению. Затем ИИ совершает ход в реальном поле. Этот тип моделирования эффективен при прогнозировании результатов предложенного хода.

11

ДЕДУКТИВНАЯ ИГРА «ХОЛОДНО-ГОРЯЧО»

«Холодно-горячо» — это дедуктивная игра, в которой игрок пытается угадать случайное трехзначное число (без повторяющихся цифр), сгенерированное компьютером. После каждой попытки компьютер предоставляет игроку подсказки трех типов:

- Холодно. Ни одна цифра не отгадана.
- Тепло. Одна цифра отгадана, но не отгадана ее позиция.
- Горячо. Одна цифра и ее позиция отгаданы.

Компьютер может дать несколько подсказок, сортируемых в алфавитном порядке. Если секретное число 456, а предположение игрока — 546, подсказки будут иметь вид «Горячо Тепло Тепло». Подсказка «Горячо» относится к 6, а «Тепло Тепло» — к 4 и 5.

В этой главе вы изучите несколько новых методов и функций, доступных в Python. Вы также узнаете о расширенных операторах присваивания и интерполяции строк. Хотя эти возможности и не позволяют делать ничего принципиально нового, это хороший способ сэкономить время, упрощая код.

В ЭТОЙ ГЛАВЕ РАССМАТРИВАЮТСЯ СЛЕДУЮЩИЕ ТЕМЫ:

- Функция `random.shuffle()`
- Расширенные операторы присваивания, `+=`, `-=`, `*=` и `/=`
- Метод списка `sort()`
- Строковый метод `join()`
- Интерполяция строк
- Спецификатор преобразования `%s`
- Вложенные циклы

Пример запуска игры «Холодно-горячо»

Вот что видит пользователь, когда запускается программа «Холодно-горячо». Текст, который вводит игрок, выделен полужирным шрифтом.

```
Я загадаю 3-х значное число, которое вы должны отгадать.
Я дам несколько подсказок...
Когда я говорю:  Это означает:
    Холодно     Ни одна цифра не отгадана.
    Тепло       Одна цифра отгадана, но не отгадана ее позиция.
    Горячо      Одна цифра и ее позиция отгаданы.
Итак, я загадал число. У вас есть 10 попыток, чтобы отгадать его.
Попытка №1:
123
Тепло
Попытка №2:
245
Тепло
Попытка №3:
672
Горячо Горячо
Попытка №4:
682
Горячо Горячо
Попытка №5:
692
Вы угадали!
Хотите сыграть еще раз? (да или нет)
нет
```

Исходный код игры «Холодно-горячо»

В редакторе файлов создайте новый файл, выбрав команду меню **File** ⇒ **New File** (Файл ⇒ Новый файл). В открывшемся окне введите приведенный ниже исходный код и сохраните файл под именем *bagels.py*. Затем нажмите клавишу **F5** и запустите программу. Если при выполнении программы возникают ошибки, сравните код, который вы набрали, с оригинальным кодом с помощью онлайн-инструмента на сайте **inventwithpython.com/diff/**.

```
1. import random
2.
3. NUM_DIGITS = 3
4. MAX_GUESS = 10
5.
6. def getSecretNum():
7.     # Возвращает строку уникальных случайных цифр, длина которой составляет NUM_DIGITS.
8.     numbers = list(range(10))
9.     random.shuffle(numbers)
10.     secretNum = ''
11.     for i in range(NUM_DIGITS):
12.         secretNum += str(numbers[i])
13.     return secretNum
14.
15. def getClues(guess, secretNum):
16.     # Возвращает строку с подсказками пользователю "Тепло", "Горячо" и "Холодно".
17.     if guess == secretNum:
18.         return 'Вы угадали!'
19.
20.     clues = []
21.     for i in range(len(guess)):
22.         if guess[i] == secretNum[i]:
23.             clues.append('Горячо')
24.         elif guess[i] in secretNum:
25.             clues.append('Тепло')
26.     if len(clues) == 0:
27.         return 'Холодно'
28.
29.     clues.sort()
30.     return ' '.join(clues)
31.
32. def isOnlyDigits(num):
33.     # Возвращает значение True, если num - строка, состоящая только из цифр. В противном
случае возвращает False.
34.     if num == '':
35.         return False
36.
37.     for i in num:
38.         if i not in '0 1 2 3 4 5 6 7 8 9'.split():
```

```
39.             return False
40.
41.     return True
42.
43.
44. print('Я загадаю %s-х значное число, которое вы должны отгадать.' % (NUM_DIGITS))
45. print('Я дам несколько подсказок...')
46. print('Когда я говорю:    Это означает:')
47. print('  Холодно          Ни одна цифра не отгадана.')
48. print('  Тепло            Одна цифра отгадана, но не отгадана ее позиция.')
49. print('  Горячо           Одна цифра и ее позиция отгаданы.')
50.
51. while True:
52.     secretNum = getSecretNum()
53.     print('Итак, я загадал число. У вас есть %s попыток, чтобы отгадать его.' % (MAX_GUESS))
54.
55.     guessesTaken = 1
56.     while guessesTaken <= MAX_GUESS:
57.         guess = ''
58.         while len(guess) != NUM_DIGITS or not isOnlyDigits(guess):
59.             print('Попытка №%s: ' % (guessesTaken))
60.             guess = input()
61.
62.         print(getClues(guess, secretNum))
63.         guessesTaken += 1
64.
65.         if guess == secretNum:
66.             break
67.         if guessesTaken > MAX_GUESS:
68.             print('Попыток больше не осталось. Я загадал число %s.' % (secretNum))
69.
70.     print('Хотите сыграть еще раз? (да или нет)')
71.     if not input().lower().startswith('д'):
72.         break
```

Блок-схема игры «Холодно-горячо»

Блок-схема, показанная на рис. 11.1, описывает происходящее в этой игре и порядок, в котором может быть сделан каждый шаг.

Блок-схема игры «Холодно-горячо» довольно проста. Компьютер генерирует секретное число, игрок пытается угадать это число, компьютер выдает

игроку подсказки, основанные на его предположениях. Это происходит снова и снова, пока игрок не победит или не проиграет. После окончания игры компьютер предложит сыграть еще раз.

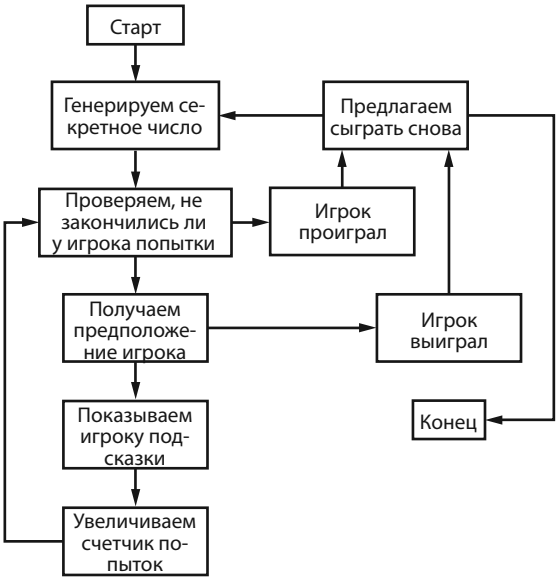


Рис. 11.1. Блок-схема игры «Холодно-горячо»

Импорт модуля `random` и определение функции `getSecretNum()`

В начале программы мы импортируем модуль `random` и настроим некоторые глобальные переменные. Затем мы определим функцию с именем `getSecretNum()`.

```
1. import random
2.
3. NUM_DIGITS = 3
4. MAX_GUESS = 10
5.
6. def getSecretNum():
7.     # Возвращает строку уникальных случайных цифр, длина которой составляет NUM_DIGITS.
```

Вместо того чтобы использовать целое число `3` для количества цифр в ответе, мы используем константу `NUM_DIGITS`. То же самое касается количества попыток, которые получает игрок; мы используем константу `MAX_GUESS` вместо

целого числа 10. Теперь будет легко изменить количество попыток или цифр секретного числа. Просто измените значения в строке 3 и/или 4, остальная часть программы по-прежнему будет работать без каких-либо изменений.

Функция getSecretNum() генерирует секретное число, которое содержит неповторяющиеся цифры. Игра «Холодно-горячо» станет намного веселее, если цифры секретного числа не будут повторяться, как например '244' или '333'. Мы будем использовать некоторые новые функции Python, чтобы осуществить это в функции getSecretNum().

Перетасовка уникального набора цифр

Первые две строки функции getSecretNum() перетасовывают набор неповторяющихся чисел:

```
8.    numbers = list(range(10))
9.    random.shuffle(numbers)
```

Функция list(range(10)) в строке 8 принимает значение [0, 1, 2, 3, 4, 5, 6, 7, 8, 9], поэтому переменная numbers содержит список из всех 10 цифр.

Изменение порядка элементов списка с помощью функции random.shuffle()

Функция random.shuffle() случайным образом изменяет порядок элементов списка (в данном случае — списка цифр). Эта функция не возвращает значение, а, скорее, изменяет список, который вы ей передаете, прямо *на месте*. Похоже на то, как функция makeMove() в игре «Крестики-нолики» из главы 10 изменяла передаваемый ей список на месте, вместо того чтобы возвращать новый список с изменением. Вот почему вы *не используете* код типа numbers = random.shuffle(numbers).

Поэкспериментируйте с функцией shuffle(), введя следующий код в интерактивной оболочке:

```
>>> import random
>>> spam = list(range(10))
>>> print(spam)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> random.shuffle(spam)
>>> print(spam)
```

```
[3, 0, 5, 9, 6, 8, 2, 4, 1, 7]
>>> random.shuffle(spam)
>>> print(spam)
[9, 8, 3, 5, 4, 7, 1, 2, 0, 6]
```

Каждый раз, когда функция random_shuffle() вызывается в spam, элементы в списке spam перетасовываются. Далее вы увидите, как можно применить функцию shuffle() для генерации секретного числа.

Получение секретного числа из перетасованных цифр

Секретное число будет представлять собой строку из первых цифр NUM_DIGITS перетасованного списка с целыми числами.

```
10.     secretNum = ''
11.     for i in range(NUM_DIGITS):
12.         secretNum += str(numbers[i])
13.     return secretNum
```

Переменная secretNum вначале принимает вид пустой строки. Цикл for в строке 11 выполняется NUM_DIGITS количество раз. При каждой итерации цикла целое число с индексом i извлекается из перетасованного списка, преобразуется в строку и конкатенируется в конец secretNum.

Например, если numbers относится к списку [9, 8, 3, 5, 4, 7, 1, 2, 0, 6], то при первой итерации numbers[0] (то есть 9) будет передан str(); это возвращает значение '9', конкатенируемое в конец secretNum. При второй итерации то же самое происходит с numbers[1] (то есть 8), а при третьей итерации то же самое происходит с numbers[2] (то есть 3). Конечное возвращаемое значение secretNum равно '983'.

Обратите внимание, что secretNum в этой функции содержит строку, а не целое число. Это может показаться странным, но помните, что нельзя конкатенировать целые числа. Выражение 9 + 8 + 3 принимает значение 20, в то время как нам нужно '9' + '8' + '3', принимающее значение '983'.

Расширенные операторы присваивания

Оператор += в строке 12 является одним из *расширенных операторов присваивания*. Обычно если вы хотите добавить или конкатенировать значение с переменной, то используете код, который выглядит следующим образом:

```
>>> spam = 42
>>> spam = spam + 10
>>> spam
52
>>> eggs = 'Привет, '
>>> eggs = eggs + 'мир!'
>>> eggs
'Привет, мир!'
```

Расширенные операторы присваивания — это короткий путь, освобождающий вас от перепечатывания имени переменной. Следующий код делает то же, что и предыдущий.

```
>>> spam = 42
>>> spam += 10 # То же, что и spam = spam + 10
>>> spam
52
>>> eggs = 'Привет, '
>>> eggs += 'мир!' # То же, что и eggs = eggs + 'мир!'
>>> eggs
'Привет, мир!'
```

Есть и другие расширенные операторы присваивания. Введите следующий код в интерактивной оболочке:

```
>>> spam = 42
>>> spam -= 2
>>> spam
40
```

Инструкция `spam -= 2` аналогична инструкции `spam = spam - 2`, поэтому выражение принимает значение `spam = 42 - 2`, а затем `spam = 40`.

Существуют также расширенные операторы присваивания для умножения и деления.

```
>>> spam *= 3
>>> spam
120
>>> spam /= 10
```

```
>>> spam
12.0
```

Инструкция `spam *= 3` аналогична инструкции `spam = spam * 3`. Итак, поскольку переменной `spam` ранее было присвоено значение `40`, полностью выражение будет иметь вид `spam = 40 * 3`, что равно `120`. Выражение `spam /= 10` аналогично `spam = spam / 10`, а результат вычисления выражения `spam = 120/10` равен `12.0`. Обратите внимание, что после деления значение переменной `spam` становится числом с плавающей запятой.

Подсчет выдаваемых подсказок

Функция `getClues()` вернет строку с подсказками «горячо», «тепло» и «холодно» в зависимости от параметров `guess` и `secretNum`.

```
15. def getClues(guess, secretNum):
16.     # Возвращает строку с подсказками пользователю "Тепло", "Горячо" и "Холодно".
17.     if guess == secretNum:
18.         return 'Вы угадали!'
19.
20.     clues = []
21.     for i in range(len(guess)):
22.         if guess[i] == secretNum[i]:
23.             clues.append('Горячо')
24.         elif guess[i] in secretNum:
25.             clues.append('Тепло')
```

Наиболее очевидный шаг — проверить, совпадает ли предположение с секретным числом, что мы и делаем в строке 17. В данном случае код в строке 18 возвращает значение `'Вы угадали!'`.

Если предположение не совпадает с секретным числом, программа должна определить, какие игроку дать подсказки. Список в переменной `clues` сначала будет пустым, а по мере необходимости в него будут добавлены строки `'Горячо'` и `'Тепло'`.

Программа делает это, перебирая каждый возможный индекс в `guess` и `secretNum`. Строки в обеих переменных будут одинаковой длины, так что код в строке 21 работал бы одинаково, независимо от того, использовались бы там `len(guess)` или `len(secretNum)`. Поскольку значение переменной `i` изменяется в диапазоне от `0` до `1` до `2` и так далее, код в строке 22 проверяет,

совпадает ли первый, второй, третий и так далее символ guess с символом соответствующего индекса secretNum. Если совпадает, код в строке 23 добавляет строку 'Горячо' к значению переменной clues.

В противном случае код в строке 24 проверяет, содержится ли в secretNum цифра с i-й позиции в guess. Если содержится, это значит, что цифра есть в секретном числе, но не в той же позиции. В данном случае код в строке 25 добавляет 'Тепло' к значению переменной clues.

Если список clues пуст после выполнения цикла, тогда ясно, что в guess вообще нет правильных цифр.

```
26.     if len(clues) == 0:
27.         return 'Холодно'
```

В данном случае код в строке 27 возвращает строку 'Холодно' как единственную подсказку.

Метод списка sort()

Для списков доступен метод sort(), который размещает элементы списка в алфавитном или нумерованном порядке. Когда вызывается метод sort(), он не возвращает отсортированный список, а сортирует список на месте. Это похоже на работу метода shuffle().

Нельзя использовать return spam.sort(), потому что это вернуло бы значение None. Вместо этого вам нужна отдельная строка, spam.sort(), и тогда строка вернет spam.

В интерактивной оболочке введите следующий код:

```
>>> spam = ['кот', 'пес', 'мышь', 'аргонавт']
>>> spam.sort()
>>> spam
['аргонавт', 'кот', 'мышь', 'пес']
>>> spam = [9, 8, 3, 5.5, 5, 7, 1, 2.1, 0, 6]
>>> spam.sort()
>>> spam
[0, 1, 2.1, 3, 5, 5.5, 6, 7, 8, 9]
```

Когда мы сортируем список строк, строки возвращаются в алфавитном порядке, но при сортировке списка чисел числа возвращаются в порядке возрастания.

В строке 29 мы используем функцию `sort()` в списке `clues`.

```
29.     clues.sort()
```

Список `clue` нужно сортировать в алфавитном порядке, чтобы избавиться от дополнительной информации, которая помогла бы игроку отгадать секретное число с меньшими усилиями. Если бы содержимое `clues` выглядело так: `['Тепло', 'Горячо', 'Тепло']`, это указывало бы игроку, что средняя цифра в его предположении находится на правильном месте. Поскольку две других подсказки — `'Тепло'`, игрок поймет, что для отгадывания секретного числа ему нужно всего лишь переставить местами первую и третью цифры.

Если же подсказки сортируются в алфавитном порядке, игрок не может быть уверен, к какому числу относится подсказка `Горячо`. Это делает игру сложнее и веселее.

Строковый метод `join()`

Строковый метод `join()` возвращает список строк в виде единой строки, соединенной вместе.

```
30.     return ' '.join(clues)
```

Строка, в которой вызывается этот метод (в строке 30 это одинарный пробел, `' '`), появляется между каждой строкой в списке. Чтобы увидеть пример, введите в интерактивной оболочке следующее:

```
>>> ' '.join(['Меня', 'зовут', 'Вася', 'Пупкин'])
'Меня зовут Вася Пупкин'
>>> ', '.join(['Жизнь', 'Вселенная', 'Бесконечность'])
'Жизнь, Вселенная, Бесконечность'
```

Таким образом, строка, возвращаемая в строке кода 30, представляет собой каждую строку в `clue` плюс одинарные пробелы между этими строками. Строковый метод `join()` — это своего рода противоположность строковому методу `split()`. Если `split()` возвращает список из разделенной строки, то `join()` возвращает строку из объединенного списка.

Проверка на содержание в строке только чисел

Функция `isOnlyDigits()` помогает определить, ввел ли игрок предположение в допустимом формате.

```
32. def isOnlyDigits(num):
33.             # Возвращает значение True, если num - строка, состоящая только из цифр. В противном случае возвращает False.
34.     if num == '':
35.         return False
```

Сначала код в строке 34 проверяет, является ли `num` пустой строкой, и если это так, то возвращает значение `False`. Затем цикл `for` выполняет итерацию по каждому символу в строке `num`.

```
37.     for i in num:
38.         if i not in '0 1 2 3 4 5 6 7 8 9'.split():
39.             return False
40.
41.     return True
```

Значение переменной `i` будет содержать один символ при каждой итерации. Внутри блока `for` код проверяет, есть ли `i` в списке, возвращенном `'0 1 2 3 4 5 6 7 8 9'.split()`. (Возвращаемое значение из `split()` эквивалентно `['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']`.) Если `i` отсутствует в этом списке, понятно, что в значении переменной `num` есть символ, отличный от цифры. В таком случае код в строке 39 возвращает значение `False`.

Но если выполнение продолжается после цикла `for`, тогда каждый символ в `num` — это цифра. В этом случае код в строке 41 возвращает значение `True`.

Начало игры

После того как были определены все функции, код в строке 44 фактически запускает программу.

```
44. print('Я загадаю %s-х значное число, которое вы должны отгадать.' % (NUM_DIGITS))
45. print('Я дам несколько подсказок...')
46. print('Когда я говорю:    Это означает:')
```

```
47. print('  Холодно          Ни одна цифра не отгадана.')
48. print('  Тепло           Одна цифра отгадана, но не отгадана ее позиция.')
49. print('  Горячо          Одна цифра и ее позиция отгаданы.')
```

Вызовы функции `print()` сообщают игроку правила игры, а также значения подсказок «холодно», «тепло» и «горячо». В строке 44 в код функции `print()` добавлены переменные `%(NUM_DIGITS)` в конце строки и `%s` внутри нее. Этот прием известен как *интерполяция строк*.

Интерполяция строк

Интерполяция строк, также известная как *форматирование строк,* — это способ экономии времени при наборе кода. Обычно, если вы хотите использовать значения одной строки внутри переменных в другой строке, вам нужно использовать оператор конкатенации, `+`.

```
>>> name = 'Алиса'
>>> event = 'ночной клуб'
>>> location = 'центре города'
>>> day = 'субботу'
>>> time = '23:00'
>>>print('Привет,'+name+'.Ты пойдешь в'+event+'в'+location+'в эту'+day+'в'+time+'часа'+'?')
Привет, Алиса. Ты пойдешь в ночной клуб в центре города в эту субботу в 23:00 часа?
```

Как видите, может потребоваться много времени, чтобы набрать строку, которая объединяет несколько строк. Вместо этого вы можете использовать интерполяцию строк, которая позволяет помещать в строку заполнители, вроде `%s`. Эти заполнители называются *спецификаторами преобразования*. Как только вы ввели спецификаторы преобразования, можно поместить все имена переменных в конец строки. Каждый спецификатор `%s` заменяется переменной в конце строки в том порядке, в котором вы ввели эти переменные. Например, следующий код делает то же самое, что и предыдущий:

```
>>> name = 'Алиса'
>>> event = 'ночной клуб'
>>> location = 'центре города'
>>> day = 'субботу'
>>> time = '23:00'
```

```
>>> print('Привет, %s. Ты пойдешь в %s в %s в эту %s в %s часа?' % (name, event, location, day, time))
Привет, Алиса. Ты пойдешь в ночной клуб в центре города в эту субботу в 23:00 часа?
```

Обратите внимание, что имя первой переменной используется для первого спецификатора `%s`, имя второй — для второго и так далее. У вас должно быть одинаковое количество спецификаторов преобразования `%s` и переменных.

Еще одно преимущество использования интерполяции строк вместо конкатенации заключается в том, что интерполяция работает с любым типом данных, а не только со строками. Все значения автоматически преобразуются в строчный тип данных. Если вы соедините целое число со строкой, то получите следующую ошибку.

```
>>> spam = 42
>>> print('Спам == ' + spam)
    Traceback (most recent call last):
        File "<stdin>", line 1, in <module>
    TypeError: Can't convert 'int' object to str implicitly
```

Конкатенация строк предназначена только для объединения двух строк, но значение переменной `spam` — целое число. Вам придется помнить о необходимости использовать `str(spam)` вместо `spam`.

Теперь введите следующий код в интерактивной оболочке:

```
>>> spam = 42
>>> print('Спам — это %s' % (spam))
Спам — это 42
```

При использовании интерполяции строк преобразование в строки выполняется автоматически.

Игровой цикл

Код в строке 51 представляет собой бесконечный цикл `while` с условием `True`, который будет повторяться до тех пор, пока не будет выполнена инструкция `break`.

```
51. while True:
52.     secretNum = getSecretNum()
53.     print('Итак, я загадал число. У вас есть %s попыток, чтобы отгадать его.' % (MAX_GUESS))
```

```
54.
55.     guessesTaken = 1
56.     while guessesTaken <= MAX_GUESS:
```

Внутри бесконечного цикла находится секретное число из функции `getSecretNum()`. Это секретное число присваивается переменной `secretNum`. Помните, что значение `secretNum` является строкой, а не целым числом.

Используя интерполяцию строк вместо конкатенации, код в строке 53 сообщает игроку, сколько в секретном числе цифр. Код в строке 55 присваивает переменной `guessesTaken` значение `1`, отмечая это как первую попытку. Код в строке 56 содержит новый цикл `while`, выполняющийся, пока игроку доступны попытки. На языке кода это значит, что `guessesTaken` меньше или равно значению `MAX_GUESS`.

Обратите внимание, что цикл `while` в строке 56 находится внутри другого цикла `while`, который был запущен в строке 51. Такие циклы внутри циклов называются *вложенными*. Инструкции `break` или `continue` (как `break` в строке 66) будут прерывать или продолжать только вложенный цикл, но ни один из внешних циклов.

Получение предположения игрока

Переменная `guess` содержит предположение игрока, возвращенное функцией `input()`. Код продолжает цикл, спрашивая игрока, пока тот не введет допустимое значение.

```
57.         guess = ''
58.         while len(guess) != NUM_DIGITS or not isOnlyDigits(guess):
59.             print('Попытка №%s: ' % (guessesTaken))
60.             guess = input()
```

Допустимое предположение содержит только цифры, причем столько же цифр, сколько и секретное число. Цикл `while`, который начинается в строке 58, проверяет допустимость введенного значения.

Переменной `guess` присвоена пустая строка в строке 57, поэтому, чтобы убедится, что интерпретатор перейдет внутрь цикла к строке 59, условие цикла `while` в строке 58 при первой проверке ложно.

Получение подсказок в зависимости от предположения игрока

После того как интерпретатор пройдет цикл `while`, который был запущен в строке 58, в переменную `guess` помещается допустимая предположение. Те-

перь программа передает значения переменных guess и secretNum функции getClues().

```
62.         print(getClues(guess, secretNum))
63.         guessesTaken += 1
```

В строке 62 программа возвращает строку подсказок, которые сообщаются игроку. Код в строке 63 увеличивает значение переменной guessesTaken с помощью расширенного оператора присваивания, используемого для добавления.

Проверка победы или поражения игрока

Теперь выясним, победил игрок или проиграл.

```
65.         if guess == secretNum:
66.             break
67.         if guessesTaken > MAX_GUESS:
68.             print('Попыток больше не осталось. Я загадал число %s.' % (secretNum))
```

Если переменная guess имеет то же значение, что и secretNum, то игрок правильно угадал секретное число, и код в строке 66 прерывает цикл while, который был запущен в строке 56. Если значения отличаются, то выполнение продолжается до строки 67, где программа проверяет, закончились ли у игрока попытки.

Если у игрока остались попытки, интерпретатор перемещается к циклу while в строке 56, позволяя игроку осуществить дополнительную попытку. Если у игрока заканчиваются попытки (или программа прерывает цикл с помощью инструкции break в строке 66), интерпретатор пропускает цикл и переходит к строке 70.

Предложение сыграть снова

Код в строке 70 спрашивает игрока, хочет ли он сыграть снова.

```
70.     print('Хотите сыграть еще раз? (да или нет)')
71.     if not input().lower().startswith('д'):
72.         break
```

Ответ игрока, возвращаемый с помощью функции `input()`, содержит метод `lower()`, вызванный в нем, а затем вызывается метод `startswith()`, чтобы проверить, начинается ли ответ игрока с буквы `д`. Если он не начинается с `д`, программа прерывает цикл `while`, который был запущен в строке 51. Поскольку после этого цикла больше нет кода, программа завершает работу.

Если же ответ начинается с `д`, программа не запускает инструкцию `break`, и интерпретатор переходит к строке 51. Затем программа генерирует новое секретное число, чтобы игрок мог начать новую игру.

Заключение

Игру «Холодно-горячо» просто написать, но в ней бывает сложно победить. Впрочем, если вы продолжите играть, то, в конце концов, обнаружите лучшие способы угадывать, используя подсказки, которые дает вам игра. Очень напоминает то, как вы лучше овладеваете программированием, чем больше занимаетесь им.

В этой главе были представлены несколько новых функций и методов — `shuffle()`, `sort()` и `join()` — вместе с несколькими удобными сокращениями кода. Расширенные операторы присваивания требуют меньше усилий по набору кода для изменения относительного значения переменной; например, выражение `spam = spam + 1` можно сократить до `spam += 1`. С помощью интерполяции строк вы можете сделать код более удобным для чтения, поместив `%s` (*спецификатор преобразования*) внутрь строки вместо использования массы операций по конкатенации строк.

В главе 12 мы не будем заниматься программированием, а изучим следующие понятия — декартова система координат и отрицательные числа. Они понадобятся нам при разработке игр в последующих главах этой книги. Эти математические концепции используются не только в создаваемых нами играх «Охотник за сокровищами», «Реверси» и «Ловкач», но и во многих других. Даже если вы уже знакомы с этими понятиями, бегло просмотрите главу 12, чтобы немного освежить информацию.

12

ДЕКАРТОВА СИСТЕМА КООРДИНАТ

В этой главе мы рассмотрим простые математические понятия, которые будем использовать. В двумерных (или 2D, от английского 2-dimensional) играх графические элементы на экране могут перемещаться влево, вправо, вверх или вниз. Для таких игр нужен способ преобразования позиции на экране в целые числа, которыми программа может оперировать.

Вот где пригодится *декартова система координат*. *Координаты* — это числа, которые представляют собой определенную позицию на экране. Мы можем хранить их как целые числа в переменных нашей программы.

В ЭТОЙ ГЛАВЕ РАССМАТРИВАЮТСЯ СЛЕДУЮЩИЕ ТЕМЫ:

- Декартова система координат
- Оси *x* и *y*
- Отрицательные числа
- Пиксели
- Коммутативное свойство сложения
- Абсолютные значения и функция `abs()`

Сетки и декартовы координаты

Если мы говорим о шахматах, распространенный способ обращения к конкретным позициям на шахматной доске — обозначение каждой строки и столбца буквами и цифрами. На рис. 12.1 показана шахматная доска с маркированными строками и столбцами.

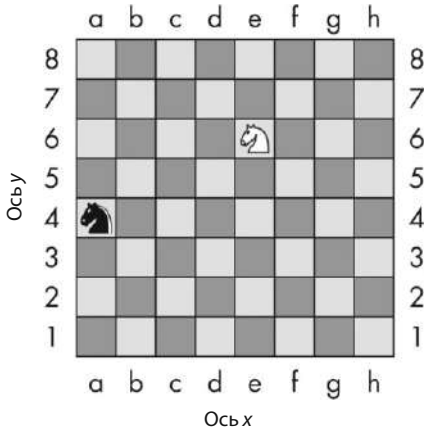


Рис. 12.1. Пример шахматной доски с черным конем на а4 и белым конем на е6

Координата для клетки на шахматной доске представляет собой комбинацию строки и столбца. Белый конь на рис. 12.1 расположен в позиции е6, так как он находится в столбце е и строке 6, а черный конь — в позиции а4, поскольку это столбец а и строка 4.

Шахматную доску можно представить как декартову систему координат. Используя метки строки и столбца, вы получите координату, которая предназначена для одной и только одной клетки на доске. Если вы знаете о декартовой системе координат из школьных уроков математики, то вам, наверное, известно, что для отсчета как строк, так и столбцов используются числа. Шахматная доска с использованием числовых координат будет выглядеть, как показано на рис. 12.2.

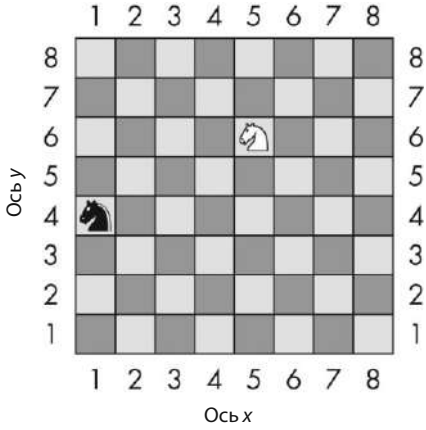


Рис. 12.2. Та же шахматная доска, но с числовыми координатами строк и столбцов

Числа слева направо (по столбцам) принадлежат оси *x*. Числа снизу вверх (по рядам) — оси *y*. Координаты всегда задаются координатой *x* на первом месте,

за которой следует координата *y*. На рис. 12.2 координата *x* белого коня равна 5, а координата *y* — 6, поэтому позиция белого коня обозначается координатами (5, 6), а не (6, 5). Аналогично черный конь расположен в позиции с координатами (1, 4), а не (4, 1), так как координата *x* черного коня равна 1, а координата *y* — 4.

Обратите внимание: чтобы занять позицию белого коня, черному коню нужно переместиться на две клетки вверх и на четыре вправо. Но необязательно смотреть на доску, чтобы понять это. Если вам известно, что белый конь находится в позиции с координатами (5, 6), а черный — в позиции (1, 4), для этого достаточно использовать вычитание.

Вычтите координату *x* черного коня из координаты *x* белого коня: 5 – 1 = 4. Черный конь должен переместиться вдоль оси *x* на четыре клетки. Теперь вычтите координату *y* черного коня из координаты *y* белого коня: 6 – 4 = 2. Черный конь должен переместиться вдоль оси *y* на две клетки.

Выполняя несложные арифметические подсчеты с координатами, можно выяснить расстояния между ними.

Отрицательные числа

В декартовых координатах также используются *отрицательные числа*, то есть числа меньше нуля. Знак «минус» перед числом показывает, что оно отрицательное: –1 меньше 0, а –2 меньше –1. Но сам 0 не является ни положительным, ни отрицательным. На рис. 12.3 изображена числовая ось, на которой положительные числа возрастают по направлению вправо, а отрицательные числа уменьшаются влево.

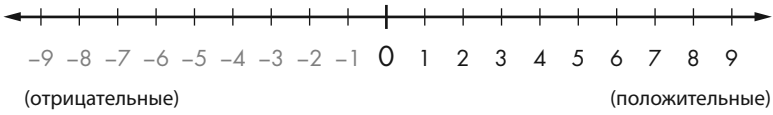


Рис. 12.3. Числовая ось с положительными и отрицательными числами

Числовая ось удобна для понимания принципа вычитания и сложения. Вместо выражения 5 + 3 вы можете представлять белого коня, перемещающегося с позиции 5 на 3 значения вправо. Как видно на рис. 12.4, белый конь займет позицию 8. И это правильно, ведь 5 + 3 равно 8.

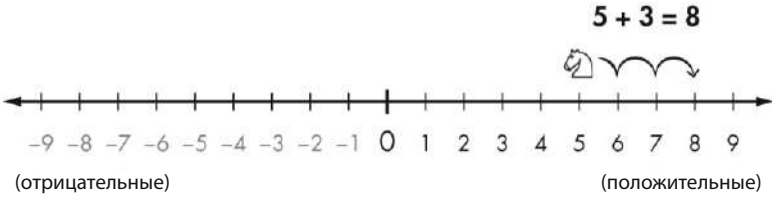


Рис. 12.4. Перемещение белого коня вправо увеличивает значение координаты

Вычитание происходит путем перемещения белого коня влево. Поэтому при выражении 5 – 6 белый конь из позиции 5 перемещается на 6 значений влево, как показано на рис. 12.5.

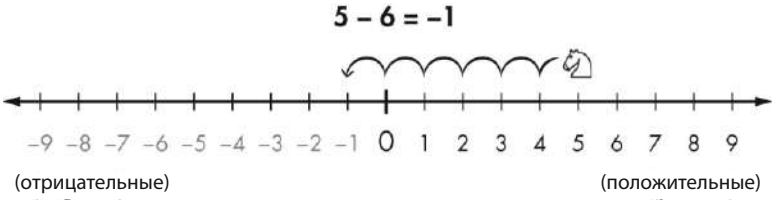


Рис. 12.5. Перемещение белого коня влево уменьшает значение координаты

Белый конь занимает позицию –1. Это значит, что 5 – 6 равно –1.

Если вы прибавите или вычтете отрицательное число, белый конь переместится в направлении, *противоположном* тому, в котором он перемещался в случае с положительными числами. Когда вы прибавляете отрицательное число, конь перемещается *влево*. Когда вы вычитаете отрицательное число, конь перемещается *вправо*. Выражение –1 – (–4) будет равно 3, как показано на рис. 12.6. Обратите внимание, что выражения –1 – (–4) и –1 + 4 приводят к одному и тому же результату.

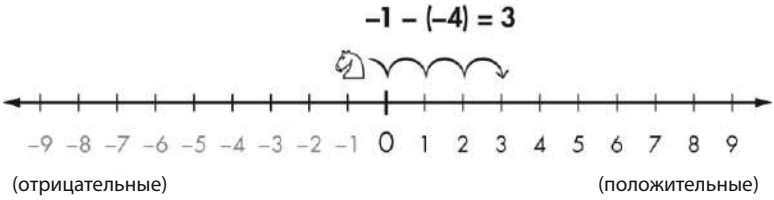


Рис. 12.6. Белый конь перемещается с позиции –6 на 4 значения вправо

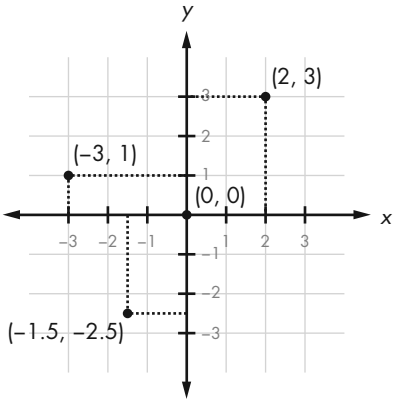


Рис. 12.7. Объединение двух числовых осей создает декартову систему координат

Вы можете рассматривать ось *x* как числовую. Добавьте еще одну числовую ось, проходящую по вертикали, и обозначьте буквой *y*. Объединив эти

две числовые оси, вы получите декартову систему координат, вроде той, что изображена на рис. 12.7. Прибавление положительного числа (или вычитание отрицательного) приведет к перемещению коня вверх по оси *y* или вправо по оси *x*, а вычитание положительного числа (или прибавление отрицательного) переместит коня вниз по оси *y* или влево по оси *x*.

Позиция (0, 0) по центру называется *началом координат*. Возможно, вы пользовались подобной системой координат на уроках математики. Давайте рассмотрим несколько приемов, которые пригодятся для усвоения принципа координат.

Система координат компьютерного экрана

Экран вашего компьютера состоит из *пикселей*, наименьших цветовых точек, которые он может отобразить. Для экранов компьютеров обычно используется система координат с началом (0, 0) в верхнем левом углу и возрастающая по направлению вниз и вправо. Это видно на рис. 12.8, где показан ноутбук с разрешением экрана 1920 пикселей в ширину и 1080 пикселей в высоту.

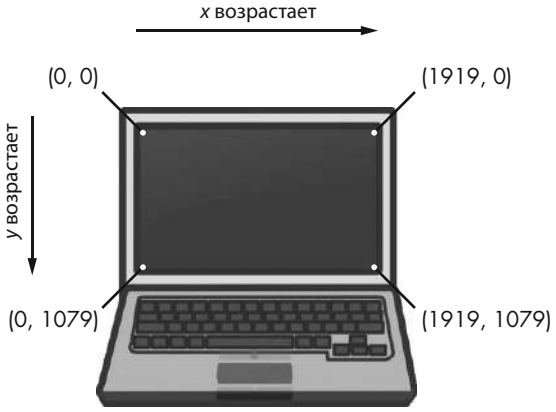


Рис. 12.8. Декартова система координат на экране компьютера

Здесь нет отрицательных координат. Обычно компьютерная графика использует такую систему координат для отображения пикселей на экране, и вы примените ее в играх из этой книги. В контексте программирования важно понимать, как работают системы координат — и те, которые используются в математике, и те, которые относятся к экранам компьютеров.

Математические хитрости

Вычитание и сложение отрицательных чисел производить легко, когда у вас перед глазами есть числовая ось. Это также может быть легко и при ее

отсутствии. Далее представлены три хитрости, которые помогут вам сложить и вычесть отрицательные числа самостоятельно.

Хитрость 1: минус «съедает» плюс слева от себя

Когда вы видите знак минуса со знаком «плюс» слева от него, вы можете заменить знак «плюс» знаком «минус». Представьте, что знак минус «съедает» знак плюс слева. Ответ все тот же, потому что прибавление отрицательного значения эквивалентно вычитанию положительного. Итак, оба выражения, 4 + –2 и 4 – 2, равны 2, как видно из примера ниже:

$$4 + -2 \rightarrow 4 - 2 \rightarrow 2$$

Хитрость 2: два минуса в сумме дают плюс

Когда вы видите два минусовых знака друг за другом без числа между ними, их можно превратить в знак плюс. Ответ тот же, потому что вычитание отрицательного значения равнозначно прибавлению положительного.

$$4 - -2 \rightarrow 4 + 2 \rightarrow 6$$

Хитрость 3: два слагаемых числа можно переставлять местами

Складывая числа, их всегда можно поменять местами. Это называют *коммутативным (переместительным) свойством сложения.* Имеется в виду, что перестановка, вроде замены 6 + 4 на 4 + 6, не повлияет на ответ, в чем вы можете убедиться, подсчитав клетки на рис. 12.9.

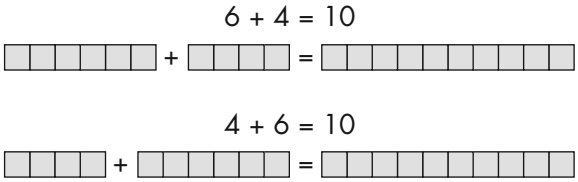


Рис. 12.9. Коммутативное свойство сложения позволяет переставлять числа местами

Скажем, вы складываете отрицательное и положительное число, например –6 + 8.

Поскольку вы выполняете операцию сложения, то можете изменить порядок чисел без изменения ответа. То есть –6 + 8 это то же самое, что и 8 + –6. Затем, взглянув на 8 + –6, легко понять, что знак минус может «съесть» знак плюс слева от себя, и задача сводится к 8 – 6 = 2, как видно ниже:

```
–6 + 8
  # Измените порядок сложения.
8 + –6
  # Знак минуса "съедает" плюс слева от себя.
8 – 6
2
```

Я перестроил задачу так, чтобы ее было проще решить без использования калькулятора или компьютера.

Абсолютные величины и функция `abs()`

Абсолютным значением (или модулем) числа является число без отрицательного знака.

Следовательно, при вычислении модуля положительные числа не изменяются, а отрицательные числа становятся положительными. Например, абсолютное значение –4 равно 4. Абсолютное значение –7 равно 7. Абсолютное значение числа 5 (которое уже положительно само по себе) составляет просто 5.

Вы можете определить расстояние между двумя объектами, вычитая их координаты и преобразовывая разность в абсолютное значение. Представьте себе, что белый конь находится в позиции 4, а черный — в позиции –2. Расстояние между ними будет составлять 6, так как 4 – –2 равно 6, а абсолютное значение 6 равно 6.

Это работает независимо от порядка чисел. Например, –2 – 4 (то есть отрицательное число два минус четыре) составляет –6, а абсолютное значение –6 также равно 6.

Функция `abs()` в Python возвращает абсолютное значение целого числа. В интерактивной оболочке введите следующие команды:

```
>>> abs(-5)
5
>>> abs(42)
42
>>> abs(-10.5)
10.5
```

Абсолютное значение `-5` равно `5`. Абсолютное значение положительного числа — просто это число, поэтому абсолютное значение `42` равно `42`. Даже у чисел с запятой есть абсолютное значение, так что абсолютное значение `-10.5` составляет `10.5`.

Заключение

Чаще всего программирование не требует глубокого понимания принципов математики. До этой главы мы обходились простым сложением и умножением.

Декартова система координат необходима для описания определенной позиции в двумерной области. Координаты содержат два числа: координату *x* и координату *y*. Ось *x* расположена горизонтально, влево и вправо, а ось *y* — вертикально, вверх и вниз. На экране компьютера начало координат находится в верхнем левом углу, а координаты возрастают по направлению вправо и вниз.

Три математические хитрости, о которых вы узнали в этой главе, позволяют легко сочетать положительные и отрицательные целые числа. Первая: знак минус «съедает» знак плюс слева. Вторая: два минуса рядом друг с другом объединяются в знак плюс. Третья: можно менять порядок чисел, которые вы сочетаете.

Остальные игры в этой книге используют эти принципы, потому что содержат в себе двумерные области. Все игры, использующие графику, требуют понимания того, как работает декартова система координат.

13

ИГРА «ОХОТНИК ЗА СОКРОВИЩАМИ»

В игре «Охотник за сокровищами» мы впервые применим декартовую систему координат, изученную в главе 12. В этой игре также используются структуры данных (на самом деле, это просто способ красиво выразиться о наличии списков, содержащих другие списки и аналогичные составные переменные). Поскольку ваши игры становятся более сложными, вам понадобится организовывать свои данные в структуры.

Действие игры происходит в океане, игрок погружает гидролокаторы, чтобы найти на затонувших судах сундуки с сокровищами. *Гидролокатор* (сонар) — это технология, которую используют корабли для поиска объектов под водой. Гидролокаторы в этой игре сообщают игроку, как далеко находится ближайший сундук с сокровищами, не сообщая направление. Но, используя несколько гидролокаторов, игрок может определить местоположение сундука.

В ЭТОЙ ГЛАВЕ РАССМАТРИВАЮТСЯ СЛЕДУЮЩИЕ ТЕМЫ:

- Структуры данных
- Теорема Пифагора
- Метод списка `remove()`
- Строковый метод `isdigit()`
- Функция `sys.exit()`

Нужно найти и поднять на палубу 3 сундука, игрок может использовать только 20 гидролокаторов для их обнаружения. Представьте, что не видите сундук с сокровищами на рис. 13.1. Поскольку гидролокаторы могут определять только расстояние до сундука, но не его направление, сокровище может быть в любой позиции на кольце вокруг гидролокатора.

Рис. 13.1. Кольцо гидролокатора касается (спрятанного) сундука с сокровищами

Но несколько таких устройств, работающих сообща, могут сузить область поиска сундука до точных координат там, где кольца пересекаются (рис. 13.2).

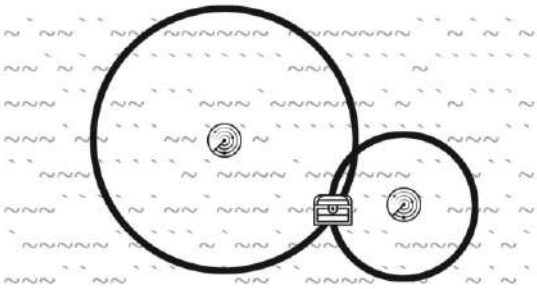


Рис. 13.2. Сочетание нескольких колец позволяет увидеть, где может быть спрятан сундук

Пример запуска игры «Охотник за сокровищами»

Ниже показано, что видит пользователь, когда запускается программа «Охотник за сокровищами».

Текст, который вводит игрок, выделен полужирным шрифтом.

```
Охотник за сокровищами!

Показать инструктаж? (да/нет)
нет

           1         2         3         4         5
 012345678901234567890123456789012345678901234567890123456789

 0 ~`~``~``~``~`~`~```~`~``````~```~`~~~`~~```~~`~~~~~`~~`~~~~` 0
 1 ~~~~~`~~~~~~```~``~~```~``~`~`~`~``~~```~~~`~`~```~~~~`~`~~` 1
 2 ```~```~``~`~`~~~``~~`~~~~~`~~``~``~``~~```~~``~~`~~~```~~`~ 2
 3 `~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ 3
 4 ~~~`~~~`~`~~~``~~~`~`~``~~~~``~~~~``~~~~`~`~``~~```~``~~`~`~ 4
 5 `~``~```~`~`~~``~~~~```~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ 5
 6 ~`~```~~`~~```~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ 6
 7 ~`~~~```~``~```~`~```~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ 7
 8 `~``~~`~`~~`~~`~~``~```~```~`~```~``~```~~~```~~~~~~~~~~~~~~ 8
 9 ~`~```~~```~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ 9
10 `~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ 10
11 ~``~~~```~`~~`~~~`~~~```~```~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ 11
12 ~~~~~```~`~```~```~`~`~`~~`~~`~```~~~~~~~~~~~~~~~~~~~~~~~~ 12
13 `~~```~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ 13
14 ```~```~`~`~```~```~`~```~`~```~~```~`~~~~```~~```~```~`~~`~`~~~~~~~ 14

 012345678901234567890123456789012345678901234567890123456789
           1         2         3         4         5
Осталось гидролокаторов: 20. Осталось сундуков с сокровищами: 3.
Где следует опустить гидролокатор? (координаты: 0-59 0-14) (или введите "выход")
25 5
           1         2         3         4         5
 012345678901234567890123456789012345678901234567890123456789

 0 ~`~```~```~``~`~`~```~`~~~~~~~```~`~~~`~~```~~`~~~~~`~~`~~~~` 0
 1 ~~~~~`~~~~~~```~``~~```~``~`~`~`~``~~```~~~`~`~```~~~~`~`~~` 1
 2 ```~```~``~`~`~~~``~~`~~~~~`~~``~``~``~~```~~``~~`~~~```~~`~ 2
 3 `~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ 3
 4 ~~~`~~~`~`~~~``~~~`~`~```~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ 4
 5 `~``~```~`~`~~``~~~~```5`~```~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ 5
 6 ~`~```~~`~~```~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ 6
 7 ~`~~~```~``~```~`~```~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ 7
 8 `~``~~`~`~~`~~`~~``~```~```~`~```~``~```~~~```~~~~~~~~~~~~~~ 8
 9 ~`~```~~```~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ 9
10 `~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ 10
11 ~``~~~```~`~~`~~~`~~~```~```~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ 11
12 ~~~~~```~`~```~```~`~`~`~~`~~`~```~~~~~~~~~~~~~~~~~~~~~~~~ 12
13 `~~```~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ 13
14 ```~```~`~`~```~```~`~```~`~```~~```~`~~~~```~~```~```~`~~`~`~~~~~~~ 14
```

```
 012345678901234567890123456789012345678901234567890123456789
           1         2         3         4         5
Сундук с сокровищами обнаружен на расстоянии 9 от гидролокатора.
Осталось гидролокаторов: 19. Осталось сундуков с сокровищами: 3.
Где следует опустить гидролокатор? (координаты: 0-59 0-14) (или введите "выход")
30 5
           1         2         3         4         5
 012345678901234567890123456789012345678901234567890123456789

 0 ~`~``~``~``~`~`~```~`~``````~```~`~~~`~~```~~`~~~~~`~~`~~~~` 0
 1 ~~~~~`~~~~~````~``~~```~``~`~`~`~``~~```~~~`~`~```~~~~`~`~~` 1
 2 ```~````~``~`~`~~~``~~`````~~``~``~``~~```~~``~~`~~~````~~`~ 2
 3 `````~~``````~`~~~~~```~~``~~~`~`~~~~~~`````~`~```~~~``~``~` 3
 4 ~~~`~~~`~`~~~``~~~`~`~``~~~~``~~~~``~~~~`~`~``~~```~``~~`~`~ 4
 5 `~``~````~`~`~~``~~~~````5`~``3~~~~````````~``~~~`~~``~~```` 5
 6 ~`~```~~`~~```~````````~~```~```~~~~``~~~`~`~~`~``~~~`~~`~`` 6
 7 ~`~~~```~``~```~`~```~~~~~~~`~~`~`~~~~``~```~~~`~```~``~``~` 7
 8 `~``~~`~`~~`~~`~~``~```~````~`~```~``~````~~~````~~``~~``~~` 8
 9 ~`~``~~````~~```~`~~```~~`~``~`~~``~`~`~~~~`~`~~`~`~```~~``` 9
10 `~~~~~~`~``~``~~~``~``~~~~`~``~```~`~~``~~~~~~``````~~`~``~~ 10
11 ~``~~~````~`~~`~~~`~~~``~``````~`~``~~~~`````~~~``````~`~`~~ 11
12 ~~~~~``~`~````~```~`~`~`~~`~~`~``~~~~~~~`~~```~~``~~`~~~~``` 12
13 `~~```~~````````~~~`~~~```~~~~~~~~`~~``~~`~```~`~~````~~~``~ 13
14 ```~``~`~`~``~```~`~``~`~``~~```~`~~~``~~``~```~`~~`~``````~ 14

 012345678901234567890123456789012345678901234567890123456789
           1         2         3         4         5
Гидролокатор ничего не обнаружил. Все сундуки с сокровищами вне пределов досягаемости.
Осталось гидролокаторов: 18. Осталось сундуков с сокровищами: 3.
Где следует опустить гидролокатор? (координаты: 0-59 0-14) (или введите "выход")
25 10
           1         2         3         4         5
 012345678901234567890123456789012345678901234567890123456789

 0 ~`~``~``~``~`~`~```~`~``````~```~`~~~`~~```~~`~~~~~`~~`~~~~` 0
 1 ~~~~~`~~~~~````~``~~```~``~`~`~`~``~~```~~~`~`~```~~~~`~`~~` 1
 2 ```~````~``~`~`~~~``~~`````~~``~``~``~~```~~``~~`~~~````~~`~ 2
 3 `````~~``````~`~~~~~```~~``~~~`~`~~~~~~`````~`~```~~~``~``~` 3
 4 ~~~`~~~`~`~~~``~~~`~`~``~~~~``~~~~``~~~~`~`~``~~```~``~~`~`~ 4
 5 `~``~````~`~`~~``~~~~````5`~``3~~~~````````~``~~~`~~``~~```` 5
 6 ~`~```~~`~~```~````````~~```~```~~~~``~~~`~`~~`~``~~~`~~`~`` 6
 7 ~`~~~```~``~```~`~```~~~~~~~`~~`~`~~~~``~```~~~`~```~``~``~` 7
 8 `~``~~`~`~~`~~`~~``~```~````~`~```~``~````~~~````~~``~~``~~` 8
 9 ~`~``~~````~~```~`~~```~~`~``~`~~``~`~`~~~~`~`~~`~`~```~~``` 9
10 `~~~~~~`~``~``~~~``~``~~~4`~``~```~`~~``~~~~~~``````~~`~``~~ 10
11 ~``~~~````~`~~`~~~`~~~``~``````~`~``~~~~`````~~~``````~`~`~~ 11
12 ~~~~~``~`~````~```~`~`~`~~`~~`~``~~~~~~~`~~```~~``~~`~~~~``` 12
13 `~~```~~````````~~~`~~~```~~~~~~~~`~~``~~`~```~`~~````~~~``~ 13
14 ```~``~`~`~``~```~`~``~`~``~~```~`~~~``~~``~```~`~~`~``````~ 14
```

```
 0123456789012345678901234567890123456789012345678901234567890123456789
           1         2         3         4         5
```

Сундук с сокровищами обнаружен на расстоянии 8 от гидролокатора.

Осталось гидролокаторов: 17. Осталось сундуков с сокровищами: 3.

Где следует опустить гидролокатор? (координаты: 0-59 0-14) (или введите "выход")

29 8

```
           1         2         3         4         5
 0123456789012345678901234567890123456789012345678901234567890123456789

 0 ~`~``~``~``~`~`~```~`~``````~```~`~~~`~~```~~`~~~~~`~~`~~~~` 0
 1 ~~~~~`~~~~~''''~``~~```~``~`~`~`~``~~```~~~`~`~```~~~~`~`~~` 1
 2 ```~''''~``~`~`~~~``~~`''''~~``~``~``~~```~~``~~`~~~''''~~`~ 2
 3 `''''~~``''''''~`~~~~~```~~``~~~`~`~~~~~~`''''~`~```~~~``~``~` 3
 4 ~~~`~~~`~`~~~``~~~`~`~``~~~~``~~~~``~~~~`~`~``~~```~``~~`~`~ 4
 5 `~``~''''~`~`~~``~~~~''''X`~``X~~~~`''''''''~``~~~`~~``~~'''' 5
 6 ~`~```~~`~~```~''''''''~~```~```~~~~``~~~`~`~~`~``~~~`~~`~`` 6
 7 ~`~~~```~``~```~`~```~~~~~~~`~~`~`~~~~``~```~~~`~```~``~`` 7
 8 `~``~~`~`~~`~~`~~``~```~''''~X~```~``~''''~~~''''~~``~~``~~` 8
 9 ~`~``~~''''~~```~`~~```~~`~``~`~~``~`~`~~~~`~`~~`~`~```~~''' 9
10 `~~~~~~`~``~``~~~``~``~~~X`~``~```~`~~``~~~~~~''''''~~`~``~~ 10
11 ~``~~~''''~`~~`~~~`~~~``~''''''~`~``~~~~'''''~~~''''''~`~`~~ 11
12 ~~~~~``~`~''''~```~`~`~`~~`~~`~``~~~~~~~`~~```~~``~~`~~~~''' 12
13 `~~```~~''''''''~~~`~~~```~~~~~~~~`~~``~~`~```~`~~''''~~~``~ 13
14 ```~``~`~`~``~```~`~``~`~``~~```~`~~~``~~``~```~`~~`~''''''~ 14
```

```
 0123456789012345678901234567890123456789012345678901234567890123456789
           1         2         3         4         5
```

Вы нашли сундук с сокровищами на затонувшем судне!

Осталось гидролокаторов: 16. Осталось сундуков с сокровищами: 2.

Где следует опустить гидролокатор? (координаты: 0-59 0-14) (или введите "выход")

--*пропуск*--

Исходный код игры «Охотник за сокровищами»

В редакторе файлов создайте новый файл, выбрав команду меню **File** ⇒ **New File** (Файл ⇒ Новый файл). В открывшемся окне введите приведенный ниже исходный код и сохраните файл под именем *sonar.py*. Затем нажмите клавишу **F5** и запустите программу. Если при выполнении программы возникают ошибки, сравните код, который вы набрали, с оригинальным кодом с помощью онлайн-инструмента на сайте **inventwithpython.com/diff/**.

sonar.py

```
1. # Охотник за сокровищами
2.
3. import random
4. import sys
5. import math
6.
7. def getNewBoard():
8. # Создать структуру данных нового игрового поля  размером  60x15.
9. board = []
10. for x in range(60): # Главный список из 60 списков.
11.     board.append([])
12. for y in range(15): # Каждый список в главном списке содержит 15 односимвольных строк.
13. # Для создания океана используем разные символы, чтобы сделать его реалистичнее.
14.             if random.randint(0, 1) == 0:
15.                 board[x].append('~')
16.             else:
17.                 board[x].append('`')
18.     return board
19.
20. def drawBoard(board):
21.     # Изобразить структуру данных игрового поля.
22. tensDigitsLine = ' ' # Создать место для чисел вниз по левой стороне поля.
23.     for i in range(1, 6):
24.         tensDigitsLine += (' ' * 9) + str(i)
25.
26.     # Вывести числа в верхней части поля.
27.     print(tensDigitsLine)
28.     print(' ' + ('0123456789' * 6))
```

```
29.     print()
30.
31.     # Вывести каждый из 15 рядов.
32.     for row in range(15):
33. # К однозначным числам нужно добавить дополнительный пробел.
34.         if row < 10:
35.             extraSpace = ' '
36.         else:
37.             extraSpace = ''
38.
39.         # Создать строку для этого ряда на игровом поле.
40.         boardRow = ''
41.         for column in range(60):
42.             boardRow += board[column][row]
43.
44. print('%s%s %s %s' % (extraSpace, row, boardRow, row))
45.
46.     # Вывести числа в нижней части поля.
47.     print()
48.     print(' ' + ('0123456789' * 6))
49.     print(tensDigitsLine)
50.
51. def getRandomChests(numChests):
52. # Создать список структур данных сундука (двухэлементные списки целочисленных координат x и y)).
53.     chests = []
54.     while len(chests) < numChests:
55.          newChest = [random.randint(0, 59), random.randint(0, 14)]
56. if newChest not in chests: # Убедиться, что сундука здесь еще нет.
57.             chests.append(newChest)
58.     return chests
59.
60. def isOnBoard(x, y):
61. # Возвращать True, если координаты есть на поле; в противном случае возвращать False.
62.     return x >= 0 and x <= 59 and y >= 0 and y <= 14
63.
64. def makeMove(board, chests, x, y):
65. # Изменить структуру данных поля, используя символ гидролокатора. Удалить сундуки
66. # с сокровищами из списка с сундуками, как только их нашли. Вернуть False, если это
67. # недопустимый ход. В противном случае, вернуть строку с результатом этого хода.
68. smallestDistance = 100 # Все сундуки будут расположены ближе, чем на расстоянии в 100 единиц.
69. for cx, cy in chests:
70. distance = math.sqrt((cx - x) * (cx - x) + (cy - y) * (cy - y))
71.
```

```
72. if distance < smallestDistance: # Нам нужен ближайший сундук с сокровищами.
73.             smallestDistance = distance
74.
75.     smallestDistance = round(smallestDistance)
76.
77.     if smallestDistance == 0:
78. # Координаты xy попали прямо в сундук с сокровищами!
79.         chests.remove([x, y])
80. return 'Вы нашли сундук с сокровищами на затонувшем судне!'
81.     else:
82.         if smallestDistance < 10:
83.             board[x][y] = str(smallestDistance)
84. return 'Сундук с сокровищами обнаружен на расстоянии %s от гидролокатора.' % (smallestDistance)
85.         else:
86.             board[x][y] = 'X'
87. return 'Гидролокатор ничего не обнаружил. Все сундуки с сокровищами вне пределов досягаемости.'
88.
89. def enterPlayerMove(previousMoves):
90. # Позволить игроку сделать ход. Вернуть двухэлементный список с целыми координатами x и y.
91. print('Где следует опустить гидролокатор? (координаты: 0-59 0-14) (или введите "выход")')
92.     while True:
93.         move = input()
94.         if move.lower() == 'выход':
95.             print('Спасибо за игру!')
96.             sys.exit()
97.
98.         move = move.split()
99. if len(move) == 2 and move[0].isdigit() and move[1].isdigit() and isOnBoard(int(move[0]), int(move[1])):
100. if [int(move[0]), int(move[1])] in previousMoves:
101. print('Здесь вы уже опускали гидролокатор.')
102.                 continue
103.             return [int(move[0]), int(move[1])]
104.
105.         print('Введите число от 0 до 59, потом пробел, а затем число от 0 до 14.')
106.
107. def showInstructions():
108.     print('''Инструктаж:
109. Вы - капитан корабля, плывущего за сокровищами. Ваша задача - с помощью
110. гидролокаторов найти три сундука с сокровищами в затонувших судах на дне океана.
111. Но гидролокаторы очень просты и определяют только расстояние, но не направление.
112. Введите координаты, чтобы опустить гидролокатор в воду. На карте будет показано
113. число, обозначающее, на каком расстоянии находится ближайший сундук. Или будет
114. показана буква X, обозначающая, что сундук в области действия гидролокатора не
```

```
115. обнаружен. На карте ниже метки C - это сундуки.
116. Цифра 3 обозначает, что ближайший сундук находится на отдалении в 3 единицы.
117.
118.                            1         2         3
119.                  012345678901234567890123456789012
120.
121.                0 ~~~~`~```~`~``~~~``~`~~``~~~``~`~ 0
122.                1 ~`~`~``~~`~```~~~```~~`~`~~~`~~~~ 1
123.                2 `~`C``3`~~~~`C`~~~~`~~~~~~~~``~~~`` 2
124.                3 ```~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~` 3
125.                4 ~`~~~~`~~`~~`C`~``~~`~~~`~```~``~ 4
126.
127.                  012345678901234567890123456789012
128.                            1         2         3
129. (Во время игры сундуки на карте не обозначаются!)
130.
131. Нажмите клавишу Enter, чтобы продолжить...''')
132.     input()
133.
134.     print('''Если гидролокатор опущен прямо на сундук, вы сможете поднять
135. сундук. Другие гидролокаторы обновят данные о расположении ближайшего сундука.
136. Сундуки ниже находятся вне диапазона локатора, поэтому отображается буква X.
137.
138.                            1         2         3
139.                  012345678901234567890123456789012
140.
141.                0 ~~~~`~```~`~``~~~``~`~~``~~~``~`~ 0
142.                1 ~`~`~``~~`~```~~~```~~`~`~~~`~~~~ 1
143.                2 `~`X``7`~~~~`C`~~~~`~~~~~~~~``~~~`` 2
144.                3 ```~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~` 3
145.                4 ~`~~~~`~~`~~`C`~``~~`~~~`~```~``~ 4
146.
147.                  012345678901234567890123456789012
148.                            1         2         3
149.
150. Сундуки с сокровищами не перемещаются. Гидролокаторы определяют сундуки
151. на расстоянии до 9 единиц. Попробуйте поднять все 3 сундука до того, как все
152. гидролокаторы будут опущены на дно. Удачи!
153.
154. Нажмите клавишу Enter, чтобы продолжить...''')
155.     input()
156.
157.
```

```
158.
159. print('Охотник за сокровищами!')
160. print()
161. print('Показать инструктаж? (да/нет)')
162. if input().lower().startswith('д'):
163.     showInstructions()
164.
165. while True:
166.     # Настройка игры
167.     sonarDevices = 20
168.     theBoard = getNewBoard()
169.     theChests = getRandomChests(3)
170.     drawBoard(theBoard)
171.     previousMoves = []
172.
173.     while sonarDevices > 0:
174. # Показать гидролокаторные устройства и сундуки с сокровищами.
175. print('Осталось гидролокаторов: %s. Осталось сундуков с сокровищами: %s.' % (sonarDevices, len(theChests)))
176.
177.         x, y = enterPlayerMove(previousMoves)
178. previousMoves.append([x, y]) # Мы должны отслеживать все ходы, чтобы гидролокаторы могли
обновляться.
179.
180. moveResult = makeMove(theBoard, theChests, x, y)
181.         if moveResult == False:
182.             continue
183.         else:
184.             if moveResult == 'Вы нашли сундук с сокровищами на затонувшем судне!':
185. # Обновить все гидролокаторные устройства, в настоящее время находящиеся на карте.
186.                 for x, y in previousMoves:
187.                     makeMove(theBoard, theChests, x, y)
188.             drawBoard(theBoard)
189.             print(moveResult)
190.
191.         if len(theChests) == 0:
192. print('Вы нашли все сундуки с сокровищами на затонувших судах! Поздравляем и приятной игры!')
193.             break
194.
195.         sonarDevices -= 1
196.
197.     if sonarDevices == 0:
198. print('Все гидролокаторы опущены на дно! Придется разворачивать корабль и')
199. print('отправляться домой, в порт! Игра окончена.')
```

```
200. print('Вы не нашли сундуки в следующих местах:')
201.         for x, y in theChests:
202.             print(' %s, %s' % (x, y))
203.
204.     print('Хотите сыграть еще раз? (да или нет)')
205.     if not input().lower().startswith('д'):
206.         sys.exit()
```

Проектирование программы

Прежде чем пытаться понять исходный код, сыграйте в игру несколько раз. Игра «Охотник за сокровищами» использует списки списков и другие сложные переменные, называемые *структурами данных*. Структуры данных хранят значения в определенном порядке и используются для представления чего-либо. Например, в игре «Крестики-нолики» из главы 10 структура данных игрового поля представляла собой список строк. Строка представляла собой *X*, *O* или пустое значение, а индекс строки в списке — позицию на поле. Код игры «Охотник за сокровищами» будет включать аналогичные структуры данных для местоположений сундуков с сокровищами и гидролокаторов.

Импорт модулей `random`, `sys` и `math`

В начале программы импортируем модули `random`, `sys` и `math`.

```
1. # Охотник за сокровищами
2.
3. import random
4. import sys
5. import math
```

Модуль `sys` содержит функцию `exit()`, которая немедленно завершает работу программы. Ни одна из строк кода не будет выполняться после вызова функции `sys.exit()`; программа просто прекращает работу, как будто интерпретатор достиг конца кода. Эта функция используется в программе позже.

Модуль `math` содержит функцию `sqrt()`, которая используется для вычисления квадратного корня из числа. Соответствующие вычисления объясняются в разделе «Поиск ближайшего сундука с сокровищами» далее в этой главе.

Создание поля для новой игры

Для начала каждой новой игры требуется новая структура данных board, которая создается функцией getNewBoard(). Игровое поле программы «Охотник за сокровищами» генерируется с помощью ASCII-символов и координат *x* и *y* вокруг него.

Когда мы используем структуру данных board, то хотим получить доступ к ее системе координат таким же образом, как обращаемся к декартовым координатам. Для этого мы будем использовать список списков, вызывая каждую координату на поле следующим образом: board[x][y]. Координата *x* указывается перед координатой *y* — для получения строки с координатами (26, 12) вы должны использовать код board[26][12], а не board[12][26].

```
7. def getNewBoard():
8. # Создать структуру данных нового игрового поля  размером  60x15.
9. board = []
10. for x in range(60): # Главный список из 60 списков.
11.     board.append([])
12. for y in range(15): # Каждый список в главном списке содержит 15 односимвольных строк.
13. # Для создания океана используем разные символы, чтобы сделать его реалистичнее.
14.             if random.randint(0, 1) == 0:
15.                 board[x].append('~')
16.             else:
17.                 board[x].append('`')
```

Структура данных board представляет собой список списков строк. Первый список представляет координату *x*. Поскольку ширина игрового поля составляет 60 символов, в этом первом списке должно быть 60 списков. В строке кода 10 мы создаем цикл for, который добавит к нему 60 пустых списков.

Но board — это не просто список из 60 пустых списков. Каждый из этих 60 списков представляет координату *x* игрового поля. У поля 15 столбцов, поэтому каждый из этих 60 списков должен содержать 15 строк. Код в строке 12 — еще один цикл for, который добавляет 15 строк из символов, представляющих океан.

Океан будет массой случайно выбранных строк '~' и '`'. Символы тильды (~) и кавычки (`), расположенные рядом с клавишей **1** на клавиатуре, будут использоваться для создания океанских волн. Для определения символа, который необходимо использовать, строки кода с 14 по 17 применяют такую логику: если возвращаемое random.randint() значение равно 0, добавляется строка '~'; в противном случае — строка '`'. Это придаст океану случайный, изменчивый вид.

В качестве небольшого примера, если списку `board` присвоено значение `[['~', '~', '`'], [['~', '~', '`'], [['~', '~', '`'], ['~', '`', '`'], ['`', '~', '`']]`, тогда поле на экране будет выглядеть так:

```
~~~~`
~~~`~
`````
```

Наконец, функция возвращает значение в переменной `board` в строке кода 18.

```
18.     return board
```

Генерация игрового поля

Затем определим метод `drawBoard()`, который мы будем вызывать, чтобы создать новое поле:

```
20. def drawBoard(board):
```

Полностью поле с координатами по его краям выглядит вот так:

```
             1         2         3         4         5
   012345678901234567890123456789012345678901234567890123456789
 0 ~~~`~``~~~``~~~~``~`~`~`~`~~`~~~`~~`~``````~~`~``~`~~```~`~` 0
 1 `~`~````~~``~`~```~```~```~`~~~``~~`~~~``````~`~``~~``~~`~~` 1
 2 ```~~~~`~`~~```~~~``~````~~`~`~~`~`~`~``~~`~``~~`~`~~~~~~`~ 2
 3 ~~~~`~~~``~```~``~~`~`~~`~`~~``~````~`~````~```~`~`~`~````~ 3
 4 ~```~~~~~`~~````~~~~```~~~`~`~`~````~`~~`~`~~``~~`~``~`~``~~ 4
 5 `~```~`~`~~`~~~```~~``~``````~~``~`~`~~~~`~~``~~~~~~`~```~~` 5
 6 ``~~`~~`~``~`````~````~~``~`~~~~`~~```~~~``~`~`~~``~~~``~~~ 6
 7 ``~```~~~~~~```~`~```~~~``~`~``~`~~~~~~``````~~~`~~`~~`~~`~~ 7
 8 ~~`~`~~```~``~~``~~~``~~`~`~~`~`~```~```~~~```~~~~~~`~`~~~~` 8
 9 ```~``~`~~~`~~```~``~``~~~```~````~```~`~~`~~~~~`~``~~~~~``` 9
10 `~~~~```~`~````~`~`~~``~`~~~~`~``~``~```~~```````~`~``~````` 10
11 ~~`~`~~`~``~`~~~````````````````~~`````~`~~``~`~~~~`~~~`~~`~ 11
12 ~~`~~~~``~~~`````~~``~`~`~~``````~`~~``~```````~~``~~~`~~`~ 12
13 `~```````~~``~`~~~```~~~~```~~`~`~~~`~```````~~`~```~``~`~~~~ 13
14 ~~~``~```~`````~~`~`~``~~`~``~`~~`~`~``~`~``~~``~`~``~```~~~ 14
   012345678901234567890123456789012345678901234567890123456789
             1         2         3         4         5
```
```

Генерация поля с помощью функции `drawBoard()` подразумевает четыре этапа:

1. Создание строковой переменной строки поля с числами 1, 2, 3, 4 и 5, разделенными широкими промежутками. Эти числа обозначают координаты 10, 20, 30, 40 и 50 по оси *x*.
2. Применение этой строки, чтобы отобразить координаты оси *x* в верхней части экрана.
3. Вывод каждой строки океана вместе с координатами оси *y* по обе стороны экрана.
4. Повторный вывод оси *x* внизу. Видя координаты со всех сторон, проще понять, где нужно опустить гидролокатор.

Изображение координат *x* вдоль верхней части поля

Первая часть функции `drawBoard()` выводит ось *x* в верхней части поля. Поскольку клетки поля должны быть одного размера, каждая метка координат может соответствовать лишь одному символу. Когда нумерация координат достигает 10, число станет двузначным. Для корректного отображения мы помещаем второй знак (показатель десятков) на отдельной строке поля в позиции, как показано на рис. 13.3. Ось *x* организована таким образом, что первая строка поля показывает разряды десятков, а вторая — единицы.

```
+++++++++++++1+++++++++2+++++++++3              # Первая строка
+++0123456789012345678901234567890123456789     # Вторая строка

+0 ~`~``~``~`~`~`~```~`~``~~~~~~~```~`~~~`~~ 0 # Третья строка
```

Рис. 13.3. Расположение символов при выводе верхней части игрового поля

Строки кода с 22 по 24 создают строку для первой строки поля, которая представляет собой часть оси *x* с десятками.

```
21.     # Изобразить структуру данных игрового поля.
22. tensDigitsLine = ' ' # Создать место для чисел вниз по левой стороне поля.
23.     for i in range(1, 6):
24.         tensDigitsLine += (' ' * 9) + str(i)
```

Цифры, обозначающие позиции десятков в первой строке поля, разделены 9 пробелами, а перед цифрой `1` содержится 13 пробелов.

Строки кода 22–24 создают эту строку поля и сохраняют ее в переменной `tensDigitsLine`:

```
26.     # Вывести числа в верхней части поля.
27.     print(tensDigitsLine)
28.     print(' ' + ('0123456789' * 6))
29.     print()
```

Чтобы вывести числа в верхней части игрового поля, сначала выведите содержимое переменной `tensDigitsLine`.

Затем в следующей строке кода выведите три пробела (так, чтобы этот ряд выстроился правильно), а затем выведите строку `'0123456789'` шесть раз: (`'0123456789' * 6`).

Рисование океана

Код в строках 32–44 выводит каждый ряд океанских волн, включая числа для обозначения оси *y*, идущие сверху вниз по бокам поля.

```
31.     # Вывести каждый из 15 рядов.
32.     for row in range(15):
33. # К однозначным числам нужно добавить дополнительный пробел.
34.         if row < 10:
35.             extraSpace = ' '
36.         else:
37.             extraSpace = ''
```

Цикл `for` выводит ряды с 0 по 14, а также их номера по обе стороны поля.

Но у нас та же проблема, что и с осью *x*. Числа с одной цифрой (например, `0`, `1`, `2` и т. д.) при выводе занимают только одну позицию, но числа с двумя цифрами (например, `10`, `11` и `12`) занимают две позиции. Ряды не выстроятся правильно, если координаты будут разного размера. Поле будет выглядеть следующим образом:

```
8 ~~`~`~~```~``~~``~~~``~~`~`~~`~`~```~```~~~```~~~~~~`~`~~~~` 8
9 ```~``~`~~~`~~```~``~``~~~```~```~```~`~~`~~~~~`~``~~~~~``` 9
10 `~~~~```~`~```~`~`~~``~`~~~~`~``~``~```~~```````~`~``~`````` 10
11 ~~`~`~~`~``~`~~~`''''''''''''''''''~~`''''''~`~~``~`~~~~`~~~`~~`~ 11
```

Выход прост: добавьте пробел перед каждым однозначным числом. Строки кода 34–37 задают переменную extraSpace равной либо пробелу, либо пустой строке. Переменная extraSpace выводится всегда, но символ пробела она содержит только при однозначных номерах рядов, в противном случае принимая значение пустой строки. Таким образом мы обеспечим правильный вывод рядов на экран.

Вывод ряда в океане

Параметр board — это структура данных, отвечающая за все океанские волны. Строки кода 39–44 считывают переменную board и выводят один ряд.

```
39.         # Создать строку для этого ряда на игровом поле.
40.         boardRow = ''
41.         for column in range(60):
42.             boardRow += board[column][row]
43.
44. print('%s%s %s %s' % (extraSpace, row, boardRow, row))
```

В строке кода 40 переменная boardRow сначала принимает значение пустой строки. Цикл for в строке кода 32 заставляет переменную row выводить текущий ряд океанских волн. Внутри этого цикла в строке кода 41 есть еще один цикл for, который перебирает каждый столбец текущего ряда. В этом цикле мы присваиваем переменной boardRow результат конкатенации board[column][row], что подразумевает объединение board[0][row], board[1][row], board[2][row] и так далее до board[59][row]. Это связано с тем, что ряд содержит 60 символов, начиная с индекса 0 и заканчивая индексом 59.

Цикл for в строке кода 41 перебирает целые числа от 0 до 59. При каждой такой итерации, следующий символ в структуре данных игрового поля копируется в конец boardRow. К моменту выхода из цикла переменная boardRow содержит полностью нарисованные с помощью ASCII волны ряда. Затем в строке кода 44 выводится строка в переменной boardRow вместе с номерами рядов.

Изображение координат *x* вдоль нижней части игрового поля

Код в строках 46–49 аналогичен строкам кода 26–29.

```
46.     # Вывести числа в нижней части поля.
47.     print()
48.     print(' ' + ('0123456789' * 6))
49.     print(tensDigitsLine)
```

Эти строки выводят ось координат *x* в нижней части поля.

Создание случайных сундуков с сокровищами

Код в игре случайным образом размещает скрытые сундуки с сокровищами. Сундуки с сокровищами представлены в виде списка списков из двух целых чисел. Эти два числа представляют собой координаты *x* и *y* каждого сундука. Например, если структура данных сундука была `[[2, 2], [2, 4], [10, 0]]`, то это означало бы наличие трех сундуков с сокровищами, одного в позиции с координатами (2, 2), другого — в позиции (2, 4) и третьего — в позиции (10, 0).

Функция `getRandomChests()` создает определенное количество структур данных сундука в случайных координатах.

```
51. def getRandomChests(numChests):
52. # Создать список структур данных сундука (двухэлементные списки целочисленных координат x и y)).
53.     chests = []
54.     while len(chests) < numChests:
55.         newChest = [random.randint(0, 59), random.randint(0, 14)]
56. if newChest not in chests: # Убедиться, что сундука здесь еще нет.
57.             chests.append(newChest)
58.     return chests
```

Параметр `numChests` сообщает функции о том, сколько сундуков с сокровищами нужно сгенерировать. Цикл `while` в строке кода 54 будет повторяться до тех пор, пока всем сундукам не будут назначены координаты. Для координат в строке кода 55 выбраны два случайных целых числа. Координата *x* может быть любой в диапазоне значений от 0 до 59, а координата *y* — в диапазоне от 0 до 14. Выражение `[random.randint(0, 59), random.randint(0, 14)]` станет списком наподобие `[2, 2]`,`[2, 4]` или `[10, 0]`. Если эти координаты еще не содержатся в списке `chests`, они присоединяются к нему в строке кода 57.

Определение допустимости хода

Когда игрок вводит координаты *x* и *y* позиции, где он желает опустить гидролокатор в воду, нужно убедиться, что введенные числа допустимы.

Как упоминалось ранее, существуют два условия для того, чтобы ход был допустимым: значение координаты *x* должно находиться в диапазоне от 0 до 59, а координаты *y* — в диапазоне от 0 до 14.

В коде функции `isOnBoard()` используются операторы `and`, чтобы объединить упомянутые условия в одно выражение и удостовериться, что каждая часть выражения истинна.

```
60. def isOnBoard(x, y):
61. # Возвращать True, если координаты есть на поле; в противном случае возвращать False.
62.     return x >= 0 and x <= 59 and y >= 0 and y <= 14
```

Поскольку мы используем логический оператор `and`, все выражение становится ложным, если хотя бы одна из координат недопустима.

Отражение хода на игровом поле

В игре «Охотник за сокровищами» игровое поле обновляется, чтобы отобразить число, указывающее расстояние от каждого гидролокатора до ближайшего сундука с сокровищами. Поэтому, когда игрок делает ход, задавая программе координаты *x* и *y*, поле изменяется в зависимости от местонахождения сундуков с сокровищами.

```
64. def makeMove(board, chests, x, y):
65. # Изменить структуру данных поля, используя символ гидролокатора. Удалить сундуки
66. # с сокровищами из списка с сундуками, как только их нашли. Вернуть False, если это
67. # недопустимый ход. В противном случае, вернуть строку с результатом этого хода.
```

Функция `makeMove()` принимает четыре параметра: структуру данных игрового поля, структуру данных сундука с сокровищами, координату *x* и координату *y*. Функция `makeMove()` возвращает строковое значение, описывающее, что произошло в ответ на ход.

- Если координаты попадают прямо на сундук с сокровищами, функция `makeMove()` возвращает `'Вы нашли сундук с сокровищами на затонувшем судне!'`.

- Если координаты находятся от сундука на расстоянии 9 или менее единиц, функция `makeMove()` возвращает `'Сундук с сокровищами обнаружен на расстоянии %s от гидролокатора.'` (где `%s` заменяется на целое значение расстояния).
- В противном случае функция `makeMove()` вернет `'Гидролокатор ничего не обнаружил. Все сундуки с сокровищами вне пределов досягаемости.'`.

Имея координаты позиции, в которой игрок хочет опустить в воду гидролокатор, и список координат *x* и *y* для сундуков с сокровищами, вам понадобится алгоритм, чтобы узнать, какой сундук с сокровищами находится ближе всего.

Поиск ближайшего сундука с сокровищами

Строки кода 68–75 содержат алгоритм определения, какой сундук с сокровищами находится ближе всего к гидролокатору.

```
68. smallestDistance = 100 # Все сундуки будут расположены ближе, чем на расстоянии в 100 единиц.
69. for cx, cy in chests:
70. distance = math.sqrt((cx - x) * (cx - x) + (cy - y) * (cy - y))
71.
72. if distance < smallestDistance: # Нам нужен ближайший сундук с сокровищами.
73.             smallestDistance = distance
```

Параметры *x* и *y* являются целыми числами (например, `3` и `5`), и в паре они обозначают позицию на игровом поле, которую игрок указал в своем предположении. Переменная `chests` получит значение типа `[[5, 0], [0, 2], [4, 2]]`, представляющее собой расположение трех сундуков с сокровищами. На рис. 13.4 это значение представлено в графическом виде.

Чтобы определить расстояние между гидролокатором и сундуком с сокровищами, нам нужно будет произвести некоторые арифметические расчеты и найти расстояние между двумя координатами *x* и двумя *y*. Предположим, мы поместили гидролокатор в позицию (3, 5) и хотим определить расстояние до сундука с сокровищами, находящегося в позиции (4, 2).

Чтобы определить расстояние между двумя наборами координат *x* и *y*, мы воспользуемся *теоремой Пифагора*. Эта теорема применима к *прямоугольным треугольникам* — таким, у которых один угол равен 90 градусам (такие же углы у прямоугольника). В теореме Пифагора сообщается, что диагональную сторону треугольника можно рассчитать исходя из длин горизонтальной и вертикальной сторон.

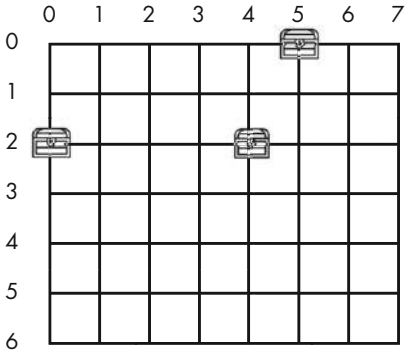


Рис. 13.4. Сундуки с сокровищами, представленные значением [[5, 0], [0, 2], [4, 2]]

На рис. 13.5 показан прямоугольный треугольник, образованный путем соединения позиций гидролокатора (3, 5) и сундука с сокровищами (4, 2).

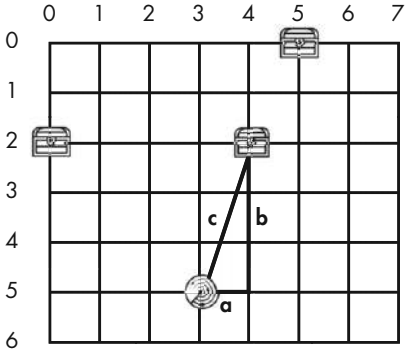


Рис. 13.5. Поле с прямоугольным треугольником, соединяющим гидролокатор и сундук с сокровищами

Теорема Пифагора выглядит как $a^2 + b^2 = c^2$, где a — длина горизонтальной стороны, b — длина вертикальной стороны, c — длина диагональной стороны или *гипотенузы*. Эти длины *возведены в квадрат*, то есть числа были умножены сами на себя. Обратная операция называется нахождением *квадратного корня* числа — это то, что мы делаем, чтобы получить c из c^2.

Давайте воспользуемся теоремой Пифагора, чтобы определить расстояние между гидролокатором (3, 5) и сундуком (4, 2).

1. Чтобы вычислить a, вычтите вторую координату x (4) из первой координаты x (3): $3 - 4 = -1$.
2. Чтобы определить a^2, умножьте a на a: $-1 \times -1 = 1$. (Отрицательное число, умноженное на отрицательное, всегда дает положительное число.)
3. Чтобы вычислить b, вычтите вторую координату y (2) из первой координаты y (5): $5 - 2 = 3$.

4. Чтобы вычислить b^2, умножьте b на b: $3 \times 3 = 9$.
5. Чтобы определить c^2, сложите a^2 и b^2: $1 + 9 = 10$.
6. Чтобы получить c из c^2, вам нужно найти квадратный корень c^2.

Модуль `math`, который мы импортировали в строке кода 5, содержит функцию с именем `sqrt()`, вычисляющую квадратный корень. В интерактивной оболочке введите следующие команды:

```
>>> import math
>>> math.sqrt(10)
3.1622776601683795
>>> 3.1622776601683795 * 3.1622776601683795
10.000000000000002
```

Обратите внимание, что результатом умножения квадратного корня числа на самого себя будет это число. (Дополнительная `2` в конце `10` — результат погрешности нахождения корня числа 10.)

Передав значение c^2 функции `sqrt()`, можно утверждать, что сундук с сокровищами находится на расстоянии 3,16 единицы от гидролокатора. Игра округлит это значение до `3`.

Давайте снова взглянем на строки кода с 68 по 70.

```
68. smallestDistance = 100 # Все сундуки будут расположены ближе, чем на расстоянии в 100 единиц.
69. for cx, cy in chests:
70. distance = math.sqrt((cx - x) * (cx - x) + (cy - y) * (cy - y))
```

Код внутри цикла `for` в строке кода 69 вычисляет расстояние до каждого сундука.

В начале цикла код в строке 68 присваивает переменной `smallestDistance` невероятно большое расстояние в `100` единиц, так что по крайней мере один из сундуков с сокровищами, что вы найдете, будет помещен в `smallestDistance` в строке кода 73.

Так как `cx - x` представляет собой горизонтальное расстояние a между сундуком и гидролокатором, то `(cx - x) * (cx - x)` — это, согласно нашей теореме Пифагора, a^2.

Значение a^2 добавляется к `(cy - y) * (cy - y)`, то есть к b^2. Эта сумма равна c^2 и передается функции `sqrt()`, чтобы определить расстояние между сундуком и гидролокатором.

Нам нужно определить расстояние между гидролокатором и ближайшим сундуком, поэтому, если это расстояние меньше, чем минимальное, оно сохраняется как новое минимальное расстояние в строке кода 73.

```
72. if distance < smallestDistance: # Нам нужен ближайший сундук с сокровищами.
73.             smallestDistance = distance
```

К моменту завершения цикла `for` вам известно, что переменная `smallestDistance` содержит наиболее короткое расстояние между гидролокатором и всеми существующими в игре сундуками с сокровищами.

Удаление значений с помощью метода списка `remove()`

Метод списка `remove()` удаляет первое вхождение значения, соответствующее переданному аргументу. Например, введите следующий код в интерактивной оболочке:

```
>>> x = [42, 5, 10, 42, 15, 42]
>>> x.remove(10)
>>> x
[42, 5, 42, 15, 42]
```

Значение `10` было удалено из списка `x`.

Теперь введите в интерактивной оболочке следующее:

```
>>> x = [42, 5, 10, 42, 15, 42]
>>> x.remove(42)
>>> x
[5, 10, 42, 15, 42]
```

Обратите внимание, что было удалено только первое значение `42`, а второе и третье остались на месте. Метод `remove()` удаляет только первое вхождение значения, которое вы ему передаете.

Если вы попытаетесь удалить значение, которого нет в списке, то получите сообщение об ошибке.

```
>>> x = [5, 42]
>>> x.remove(10)
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
ValueError: list.remove(x): x not in list
```

Как и append(), метод remove() вызывается в списке и не возвращает список. Вам необходимо использовать код типа x.remove(42), а не x = x.remove(42).

Давайте вернемся к определению расстояний между гидролокаторами и сундуками с сокровищами. Единственный случай, когда переменная smallestDistance принимает значение 0, — когда координаты *x* и *y* гидролокатора совпадают с соответствующими координатами сундука. Это значит, что игрок угадал местонахождение сундука с сокровищами.

```
77.     if smallestDistance == 0:
78. # Координаты xy попали прямо в сундук с сокровищами!
79.         chests.remove([x, y])
80. return 'Вы нашли сундук с сокровищами на затонувшем судне!'
```

В этом случае программа с помощью метода списка remove() удаляет из структуры данных chests список из двух чисел, относящийся к этому сундуку. Затем функция возвращает строку 'Вы нашли сундук с сокровищами на затонувшем судне!'.

Но если переменная smallestDistance не равна 0, это значит, что игрок не угадал точное местоположение сундука с сокровищами, и запускается блок else со строки 81.

```
81.     else:
82.         if smallestDistance < 10:
83.             board[x][y] = str(smallestDistance)
84. return 'Сундук с сокровищами обнаружен на расстоянии %s от гидролокатора.' % (smallestDistance)
85.         else:
86.             board[x][y] = 'X'
87. return 'Гидролокатор ничего не обнаружил. Все сундуки с сокровищами вне пределов досягаемости.'
```

Если расстояние от гидролокатора до сундука с сокровищами меньше 10, код в строке 83 отмечает поле строковой версией переменной smallestDistance. В противном случае на поле выводится символ 'X'.

Таким образом, игрок знает, насколько близко каждый гидролокатор расположен к сундуку с сокровищами. Если игрок видит X, то понимает, что он очень далеко.

Получение хода игрока

Функция enterPlayerMove() получает координаты *x* и *y* следующего хода игрока.

```
89. def enterPlayerMove(previousMoves):
90. # Позволить игроку сделать ход. Вернуть двухэлементный список с целыми координатами x и y.
91. print('Где следует опустить гидролокатор? (координаты: 0-59 0-14) (или введите "выход")')
92.     while True:
93.         move = input()
94.         if move.lower() == 'выход':
95.             print('Спасибо за игру!')
96.             sys.exit()
```

Параметр `previousMoves` представляет собой список списков с двумя целыми числами, обозначающими предыдущие позиции, в которых игрок опускал гидролокатор. Это нужно для того, чтобы игрок не сделал повторный ход в одной и той же позиции. Цикл `while` будет продолжать предлагать игроку сделать следующий ход, пока он не введет координаты позиции, где еще нет гидролокатора. Игрок может также ввести `'выход'`, чтобы выйти из игры. В этом случае код в строке 96 вызывает функцию `sys.exit()`, немедленно завершающую работу программы.

Предполагая, что игрок не ввел `'выход'`, программа проверяет, что ввод представляет собой два целых числа, разделенных пробелом. Код в строке 98 вызывает метод `split()` в списке `move` как новое значение `move`.

```
98.         move = move.split()
99. if len(move) == 2 and move[0].isdigit() and move[1].isdigit() and isOnBoard(int(move[0]),
int(move[1])):
100. if [int(move[0]), int(move[1])] in previousMoves:
101. print('Здесь вы уже опускали гидролокатор.')
102.                 continue
103.             return [int(move[0]), int(move[1])]
104.
105.         print('Введите число от 0 до 59, потом пробел, а затем число от 0 до 14.')
```

Если игрок ввел значение, подобное `'1 2 3'`, тогда список, возвращаемый функцией `split()`, будет иметь вид `['1', '2', '3']`. В этом случае выражение `len(move) == 2` станет ложным (список позиции должен содержать лишь два числа, поскольку он представляет собой координаты), и все выражение немедленно станет ложным. Интерпретатор Python не проверяет остальную часть выражения из-за короткого замыкания (описанного в разделе «Вычисление по короткой схеме» главы 10).

Если длина списка равна 2, то два значения будут иметь индексы move[0] и move[1]. Чтобы проверить, числовые ли это значения (как '2' или '17'), вы можете использовать функцию типа isOnlyDigits(), описанную в разделе «Проверка на содержание в строке только чисел» главы 11. Но в Python уже есть метод, который делает это.

Строковый метод isdigit() возвращает True, если строка состоит исключительно из чисел. В противном случае он возвращает значение False. В интерактивной оболочке введите следующие команды:

```
>>> '42'.isdigit()
True
>>> 'сорок'.isdigit()
False
>>> ''.isdigit()
False
>>> 'привет'.isdigit()
False
>>> x = '10'
>>> x.isdigit()
True
```

И move[0].isdigit(), и move[1].isdigit() должны принимать значение True, чтобы все условие было истинным. Заключительная часть условия в строке кода 99 вызывает функцию isOnBoard(), чтобы проверить, существуют ли координаты *x* и *y* на игровом поле.

Если все условие истинно, код в строке 100 проверяет, есть ли данный ход в списке previousMoves. Если есть, тогда инструкция continue в строке 102 возвращает интерпретатор к началу цикла while в строке кода 92, а затем снова предлагает игроку сделать ход. Если хода нет в списке, код в строке 103 возвращает список с двумя целыми числами — координатами *x* и *y*.

Вывод игроку инструкций по игре

Функция showInstructions() содержит вызов print() с многострочным выводом.

```
107. def showInstructions():
108.     print('''Инструктаж:
109. Вы - капитан корабля, плывущего за сокровищами. Ваша задача - с помощью
--пропуск--
154. Нажмите клавишу Enter, чтобы продолжить...''')
155.     input()
```

Функция `input()` позволяет игроку нажать клавишу **Enter** перед выводом следующей строки. Это связано с тем, что окно IDLE может отображать определенное количество текста за раз, а мы не хотим вынуждать игрока прокручивать экран вверх, чтобы прочесть начало текста. После нажатия клавиши **Enter**, функция возвращается к вызвавшей ее строке кода.

Игровой цикл

Теперь, когда мы ввели все функции, нужные нашей игре, давайте напишем основной код. Первое, что видит игрок после запуска программы, — это название игры, выводимое строкой кода 159. Это — основная часть программы, которая сначала выдает игроку инструкции, а затем присваивает значения переменным, которые будут использоваться в игре.

```
159. print('Охотник за сокровищами!')
160. print()
161. print('Показать инструктаж? (да/нет)')
162. if input().lower().startswith('д'):
163.     showInstructions()
164.
165. while True:
166.     # Настройка игры
167.     sonarDevices = 20
168.     theBoard = getNewBoard()
169.     theChests = getRandomChests(3)
170.     drawBoard(theBoard)
171.     previousMoves = []
```

Выражение `input().lower().startswith('д')` позволяет игроку запросить инструкции и принимает значение `True`, если игрок вводит строку, начинающуюся с буквы `'д'` или `'Д'`. Например:

```
input().lower().startswith('д')
'Д'.lower().startswith('д')
'д'.startswith('д')
True
```

Если это условие истинно, в строке кода 163 вызывается функция `showInstructions()`. Иначе начинается игра.

В строках кода с 167 по 171 присваиваются значения нескольким переменным; они описаны в таблице 13.1.

Таблица 13.1. Переменные, используемые в основном цикле игры

Переменная	Описание
`sonarDevices`	Количество гидролокаторов, оставшихся у игрока
`theBoard`	Структура данных игрового поля, используемая в этой игре
`theChests`	Список структур данных сундуков. Функция `getRandomChests()` возвращает список трех сундуков с сокровищами в случайных позициях на игровом поле
`previousMoves`	Список всех ходов с координатами *x* и *y*, сделанных игроком

Ниже мы будем использовать эти переменные, поэтому обязательно прочитайте их описание, прежде чем двигаться дальше!

Демонстрация игроку статуса игры

До тех пор пока у игрока остаются гидролокаторы, в строке 173 выполняется цикл `while`, сообщающий игроку, сколько гидролокаторов у него еще есть и сколько сундуков с сокровищами осталось на поле.

```
173.     while sonarDevices > 0:
174. # Показать гидролокаторные устройства и сундуки с сокровищами.
175. print('Осталось гидролокаторов: %s. Осталось сундуков с сокровищами: %s.' % (sonarDevices, len(theChests)))
```

После вывода информации о том, сколько осталось устройств, цикл `while` продолжает выполняться.

Обработка хода игрока

Код в строке 177 также является частью цикла `while` и использует множественное присваивание для присвоения переменных `x` и `y` двухэлементному списку, который представляет координаты хода игрока, возвращаемые `enterPlayerMove()`. Мы добавим переменную `previousMoves`, чтобы код `enterPlayerMove()` смог гарантировать, что игрок не повторит предыдущий ход.

```
177.         x, y = enterPlayerMove(previousMoves)
178. previousMoves.append([x, y]) # Мы должны отслеживать все ходы, чтобы гидролокаторы могли обновляться.
179.
```

```
180. moveResult = makeMove(theBoard, theChests, x, y)
181.         if moveResult == False:
182.             continue
```

Затем переменные x и y добавляются в конец списка previousMoves. Переменная previousMoves представляет собой список координат x и y каждого хода, совершаемого игроком. Этот список позже используется в программе в строках 177 и 186.

Переменные x, y, theBoard и theChests передаются функции makeMove() в строке кода 180. Эта функция делает необходимые изменения, чтобы разместить гидролокатор на игровом поле.

Если функция makeMove() возвращает False, значит, возникла проблема с переданными ей значениями x и y. Оператор continue отбрасывает интерпретатор обратно в начало цикла while в строке кода 173, чтобы снова запросить у игрока координаты.

Нахождение затонувшего сундука с сокровищами

Если функция makeMove() не возвращает значение False, то возвращает строку с результатами этого хода. Если это строка 'Вы нашли сундук с сокровищами на затонувшем судне!', то все гидролокаторы на поле должны обновить значения для определения *следующего* ближайшего сундука с сокровищами.

```
183.         else:
184.             if moveResult == 'Вы нашли сундук с сокровищами на затонувшем судне!':
185. # Обновить все гидролокаторные устройства, в настоящее время находящиеся на карте.
186.                 for x, y in previousMoves:
187.                     makeMove(theBoard, theChests, x, y)
188.             drawBoard(theBoard)
189.             print(moveResult)
```

Координаты *x* и *y* всех гидролокаторов содержатся в переменной previousMoves.

Перебирая значения previousMoves в строке кода 186, вы можете снова передать все эти координаты *x* и *y* функции makeMove(), чтобы обновить значения на поле. Поскольку программа здесь не выводит никакого нового текста, игрок не подозревает, что интерпретатор заново делает все предыдущие ходы. Кажется, что поле обновляется автоматически.

Проверка победы игрока

Помните, что функция makeMove() изменяет список theChests, который вы ей отправили. Поскольку theChests — список, любые изменения, внесенные в него внутри функции, сохраняются после того, как интерпретатор из нее возвращается. Функция makeMove() удаляет элементы из списка theChests при обнаружении сундуков с сокровищами, поэтому в конечном итоге (если игрок продолжит правильно угадывать) все сундуки с сокровищами будут удалены. (Помните, что под «сундуком с сокровищами» я имею в виду двухэлементные списки из координат *x* и *y* внутри списка theChests).

```
191.        if len(theChests) == 0:
192. print('Вы нашли все сундуки с сокровищами на затонувших судах! Поздравляем и приятной игры!')
193.            break
```

Когда все сундуки с сокровищами будут обнаружены на поле и удалены из списка theChests, он будет иметь длину 0. Когда это произойдет, программа выведет игроку поздравление, а затем выполнит инструкцию break, чтобы прервать цикл while. Затем интерпретатор перейдет к строке кода 197, первой после блока while.

Проверка проигрыша игрока

Строка кода 195 — последняя строка цикла while, начавшегося в строке 173.

```
195.        sonarDevices -= 1
```

Программа уменьшает значение переменной sonarDevices, потому что игрок израсходовал один гидролокатор. Если игрок продолжит упускать сундуки с сокровищами, значение переменной sonarDevices, в конце концов, будет сведено к 0. После этой строки кода интерпретатор переходит к строке 173, чтобы заново оценить условие while (т.е. sonarDevices > 0).

Когда значение переменной sonarDevices становится равным 0, это условие становится ложным, и интерпретатор продолжает выполнение за пределами блока while в строке кода 197. Но до тех пор условие останется истинным, а игрок может продолжать делать предположения.

```
197.    if sonarDevices == 0:
198. print('Все гидролокаторы опущены на дно! Придется разворачивать корабль и')
199. print('отправляться домой, в порт! Игра окончена.')
```

```
200. print('Вы не нашли сундуки в следующих местах:')
201.         for x, y in theChests:
202.             print(' %s, %s' % (x, y))
```

Код в строке 197 — первая за пределами цикла `while`. Когда интерпретатор достигает этой точки, игра заканчивается. Если переменная `sonarDevices` равна `0`, то игрок потерпел поражение, так как израсходовал гидролокаторы, не отыскав все сундуки.

Код в строках 198–200 сообщит игроку, что тот проиграл. Цикл `for` в строке 201 переберет сундуки с сокровищами, оставшиеся в списке `theChests`, и отобразит их местоположение, чтобы игрок мог увидеть, где они скрывались.

Завершение работы программы с помощью функции `sys.exit()`

Вне зависимости от исхода игры программа предлагает сыграть еще раз. Если игрок не вводит `'да'` или `'Д'`, либо вводит какую-то другую строку, которая не начинается с буквы *д*, тогда условие `not input().lower().startswith('д')` становится истинным, и выполняется функция `sys.exit()`. Это приводит к завершению работы программы.

```
204.     print('Хотите сыграть еще раз? (да или нет)')
205.     if not input().lower().startswith('д'):
206.         sys.exit()
```

В противном случае интерпретатор переходит к началу цикла `while` в строке 165, и начинается новая игра.

Заключение

Помните, как наша игра «Крестики-нолики» пронумеровала участки на игровом поле с 1 по 9? Такое подобие системы координат подходит для поля менее чем с 10 клетками, но ведь у игрового поля «Охотник за сокровищами» их 900! Декартова система координат, о которой мы узнали в главе 12, позволяет управлять этими клетками и определять расстояние между двумя позициями на поле.

Позиции в играх, использующих декартову систему координат, можно сохранять в списке списков, в которых под первым индексом содержится значе-

ние координаты *x*, а под вторым — значение *y*. Это упрощает доступ к координатам с помощью кода `board[x][y]`.

Такие структуры данных (как те, что использовались для океана и местоположений сундуков) позволяют представлять сложные вещи в виде данных, и работа с вашими играми в основном сводится к работе с этими структурами.

В следующей главе мы будем представлять буквы в виде чисел. Представляя текст в виде чисел, можно выполнять математические операции, позволяющие шифровать секретные сообщения.

14

ШИФР ЦЕЗАРЯ

Программа в этой главе на самом деле не является игрой, но, тем не менее, повеселит вас. Она превращает обычный русский язык в секретный код, а также может преобразовывать этот секретный код обратно в русский язык. Только тот, кому известен ключ к секретным кодам, сможет понять зашифрованные сообщения.

Поскольку эта программа обрабатывает текст, чтобы преобразовать его в секретные сообщения, вам придется выучить несколько новых функций и методов для управления строками. Кроме того, вы узнаете, каким образом программы могут выполнять математические операции с текстовыми строками так же, как с числами.

В ЭТОЙ ГЛАВЕ РАССМАТРИВАЮТСЯ СЛЕДУЮЩИЕ ТЕМЫ:

- Криптография и шифры
- Шифротекст, открытый текст, ключи и символы
- Шифрование и расшифровывание
- Шифр Цезаря
- Строковый метод `find()`
- Криптоанализ
- Полный перебор

Криптография и шифрование

Наука, изучающая создание секретных кодов, называется *криптографией*. Веками криптография позволяла отправлять секретные сообщения, кото-

рые могли прочесть только отправитель и получатель, даже если кто-то перехватывал закодированное сообщение. Система секретного кода называется *шифром*. Шифр, используемый программой в этой главе, называется *шифром Цезаря*.

В криптографии сообщение, которое хотят сохранить в секрете, называют *открытым текстом*. Допустим, у нас есть сообщение с открытым текстом, которое выглядит так:

```
Здесь спрятан ключ от книжного шкафа.
```

Преобразование открытого текста в закодированное сообщение называется *шифрованием* открытого текста. Открытый текст зашифрован в *шифротексте*. Шифротекст выглядит как случайные буквы, поэтому мы не можем понять, каким был исходный открытый текст, просто взглянув на шифротекст. Ниже показан предыдущий пример, зашифрованный в шифротекст:

```
Оклшг шщчещжф стею хщ сфпнфхйх ясжыж.
```

Если вам известен шифр, используемый для шифрования сообщения, вы можете *расшифровать* зашифрованный текст обратно в открытый текст (расшифровывание (дешифровка) — это противоположность шифрования).

Многие шифры используют *ключи*, то есть секретные значения, позволяющие расшифровать шифротекст, который был зашифрован с помощью конкретного шифра. Думайте о шифре как о дверном замке — его можно открыть только с помощью определенного ключа.

Как работает шифр Цезаря

Шифр Цезаря — один из самых ранних шифров. В этом шифре сообщение шифруется путем замены каждой буквы в нем «сдвинутой» буквой. В криптографии зашифрованные буквы называются *символами*, потому что они могут быть буквами, числами или любыми другими знаками. Если вы сдвинете букву «А» на одну позицию, то получите букву «Б». Если сдвинете букву «А» на две позиции, получите букву «В». На рис. 14.1 показано несколько букв, сдвинутых на три позиции.

Чтобы получить все сдвинутые буквы, нарисуйте ряд клеток с каждой буквой алфавита. Затем нарисуйте второй такой же ряд под ним, но начни-

те его с буквы через определенное количество позиций. Когда вы дойдете до конца алфавита открытого текста, начните его с начала, с «А». На рис. 14.2 показан пример с буквами, сдвинутыми на три позиции.

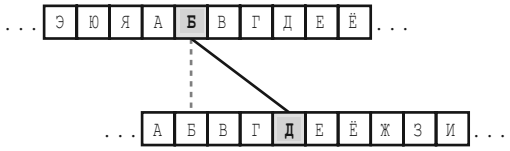


Рис. 14.1. Шифр Цезаря сдвигает буквы на три позиции. Так, буква «Б» превращается в «Д»

Рис. 14.2. Целый алфавит сдвинут на три позиции

Количество позиций, на которое вы сдвигаете буквы (от 1 до 33) — это ключ в шифре Цезаря. Если вы не знаете ключ (число, используемое для шифрования сообщения), то не сможете расшифровать секретный код. В примере на рис. 14.2 показаны преобразования букв для ключа 3.

Примечание. Несмотря на то что существуют 33 возможных ключа, шифрование вашего сообщения числом 33 образует шифротекст, который будет точно таким же, как и открытый текст!

Если вы зашифруете слово открытого текста ПРИВЕТ ключом 3, тогда:

- Буква «П» станет «Т».
- Буква «Р» станет «У».
- Буква «И» станет «Л».
- Буква «В» станет «Е».
- Буква «Е» станет «З».
- Буква «Т» станет «Х».

Итак, шифротекст слова ПРИВЕТ с ключом 3 имеет вид ТУЛЕЗХ. Чтобы расшифровать код ТУЛЕЗХ с помощью ключа 3, мы переходим от нижних клеток к верхним.

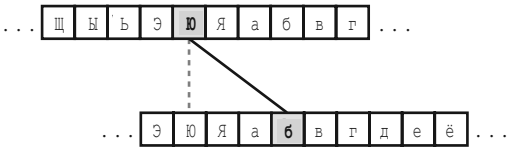


Рис. 14.3. Целый алфавит, включающий строчные буквы, сдвинут на три позиции

Если вы хотите учесть строчные буквы как отличающиеся от прописных, добавьте еще 33 клетки к тем, которые у вас уже есть, и заполните их 33 строчными буквами. Теперь с ключом, равным 3, буква «Ю» становится «б», как показано на рис. 14.3.

В этом случае шифр работает так же. Вы можете использовать буквы другого языка, заполнив клетки его алфавитом. Таким образом вы получите свой шифр.

Пример запуска программы «Шифр Цезаря»

Ниже показан пример работы программы «Шифр Цезаря», шифрующей сообщение:

```
Вы хотите зашифровать или расшифровать текст?
зашифровать
Введите текст:
Если больной очень хочет жить, врачи бессильны.
Введите ключ шифрования (1-66)
15
Преобразованный текст:
УАъч пэъКьэш эЕуьК ДэЕуБ хчБК, ряоЕч пуААчъКьЙ.
```

Теперь запустите программу и расшифруйте текст, который вы только что зашифровали.

```
Вы хотите зашифровать или расшифровать текст?
расшифровать
Введите текст:
УАъч пэъКьэш эЕуьК ДэЕуБ хчБК, ряоЕч пуААчъКьЙ.
Введите ключ шифрования (1-66)
15
Преобразованный текст:
Если больной очень хочет жить, врачи бессильны.
```

Если вы выполните расшифровку, используя неправильный ключ, текст не расшифруется должным образом.

```
Вы хотите зашифровать или расшифровать текст?
расшифровать
Введите текст:
```

```
УАъч пэъКьэш эЕуьК ДэЕуБ хчБК, ряоЕч пуААчъКьЙ.
Введите ключ шифрования (1-66)
13
Преобразованный текст:
Жунк грнюпрл рщжпю чрщжф икфю, дтвщк гжуукнюпэ.
```

Исходный код программы «Шифр Цезаря»

В редакторе файлов создайте новый файл, выбрав команду меню **File** ⇒ **New File** (Файл ⇒ Новый файл). В открывшемся окне введите приведенный ниже исходный код и сохраните файл под именем *cipher.py*. Нажмите клавишу **F5** и запустите программу. Если при выполнении программы возникают ошибки, сравните код, который вы набрали, с оригинальным кодом с помощью онлайн-инструмента на сайте **inventwithpython.com/diff/**.

```
1. # Шифр Цезаря
2.      SYMBOLS      =      'АБВГДЕЕЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯабвгдеежзийклмнопрстуфхцчшщъыьэюя'
3. MAX_KEY_SIZE = len(SYMBOLS)
4.
5. def getMode():
6.     while True:
7.         print('Вы хотите зашифровать или расшифровать текст?')
8.         mode = input().lower()
9.         if mode in ['зашифровать', 'з', 'расшифровать', 'р']:
10.             return mode
11.         else:
12.     print('Введите "зашифровать" или "з" для зашифровки или "расшифровать" или "р" для расшифровки.')
13.
14. def getMessage():
15.     print('Введите текст:')
16.     return input()
17.
18. def getKey():
19.     key = 0
20.     while True:
21.         print('Введите ключ шифрования (1-%s)' % (MAX_KEY_SIZE))
22.         key = int(input())
```

```
23.         if (key >= 1 and key <= MAX_KEY_SIZE):
24.             return key
25.
26. def getTranslatedMessage(mode, message, key):
27.     if mode[0] == 'p':
28.         key = -key
29.     translated = ''
30.
31.     for symbol in message:
32.         symbolIndex = SYMBOLS.find(symbol)
33.         if symbolIndex == -1: # Символ не найден в SYMBOLS.
34.             # Просто добавить этот символ без изменений.
35.             translated += symbol
36.         else:
37.             # Зашифровать или расшифровать
38.             symbolIndex += key
39.
40.             if symbolIndex >= len(SYMBOLS):
41.                 symbolIndex -= len(SYMBOLS)
42.             elif symbolIndex < 0:
43.                 symbolIndex += len(SYMBOLS)
44.
45.             translated += SYMBOLS[symbolIndex]
46.     return translated
47.
48. mode = getMode()
49. message = getMessage()
50. key = getKey()
51. print('Преобразованный текст:')
52. print(getTranslatedMessage(mode, message, key))
```

Установление максимальной длины ключа

Процессы шифрования и расшифровывания — это операции, обратные по отношению друг к другу. В то же время они содержат много общего кода. Давайте посмотрим, как работает каждая строка кода программы.

```
1. # Шифр Цезаря
2. SYMBOLS = 'АБВГДЕЕЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯабвгдеежзийклмнопрстуфхцчшщъыьэюя'
3. MAX_KEY_SIZE = len(SYMBOLS)
```

MAX_KEY_SIZE — константа, которая хранит длину строки SYMBOLS(66). Эта константа напоминает нам, что в нашей программе значение ключа, используемого в шифре, всегда должно быть между 1 и 66.

Выбор между шифрованием и расшифровыванием сообщения

Функция getMode() позволяет пользователю решить, желает он использовать режим шифрования или расшифровывания.

```
5. def getMode():
6.     while True:
7.         print('Вы хотите зашифровать или расшифровать текст?')
8.         mode = input().lower()
9.         if mode in ['зашифровать', 'з', 'расшифровать', 'р']:
10.             return mode
11.         else:
12.     print('Введите "зашифровать" или "з" для зашифровки или "расшифровать" или "р" для расшифровки.')
```

Код в строке 8 вызывает функцию input(), чтобы пользователь мог выбрать желаемый режим. Затем метод lower() вызывается в этой строке для возврата строчной версии (нижнего регистра) строки. Значение, возвращаемое из input().lower(), сохраняется в переменной mode. Условие конструкции if проверяет, существует ли строка, хранящаяся в mode, в списке ['зашифровать', 'з', 'расшифровать', 'р'].

Эта функция вернет строку в mode, если mode равен 'зашифровать', 'з', 'расшифровать', 'р'. Следовательно, функция getMode() вернет строку mode. Если пользователь вводит что-то отличное от 'зашифровать', 'з', 'расшифровать' или 'р', тогда цикл while вновь запросит пользователя ввести правильное значение.

Получение сообщения от игрока

Функция getMessage() просто получает от пользователя сообщение (для шифрования или расшифровывания) и возвращает его.

```
14. def getMessage():
15.     print('Введите текст:')
16.     return input()
```

Вызов input() совмещен с return, так что мы используем только одну строку кода вместо двух.

Получение ключа от игрока

Функция getKey() позволяет игроку ввести ключ, который будет использоваться для шифрования или расшифровки сообщения.

```
18. def getKey():
19.     key = 0
20.     while True:
21.         print('Введите ключ шифрования (1-%s)' % (MAX_KEY_SIZE))
22.         key = int(input())
23.         if (key >= 1 and key <= MAX_KEY_SIZE):
24.             return key
```

Цикл while гарантирует, что функция продолжит перебор до тех пор, пока пользователь не введет допустимый ключ. Значение допустимого ключа здесь находится между целочисленными значениями 1 и 66 (помните, что значение переменной MAX_KEY_SIZE равно 66, потому что в переменной SYMBOLS 66 символов). Затем функция getKey() возвращает этот ключ. Код в строке 22 присваивает переменной key значение, равное целой части того числа, которое ввел пользователь, поэтому метод getKey() возвращает целое число.

Шифрование/расшифровывание сообщения

Функция getTranslatedMessage(), собственно, и выполняет шифрование и расшифровывание.

```
26. def getTranslatedMessage(mode, message, key):
27.     if mode[0] == 'p':
28.         key = -key
29.     translated = ''
```

Эта функция имеет три параметра:

- **mode**. Этот параметр устанавливает функцию в режим шифрования или расшифровывания;
- **message**. Это открытый текст (или шифротекст), который должен быть зашифрован (или расшифрован);
- **key**. Это ключ, который используется в этом шифре.

Код в строке 27 проверяет, является ли первая буква в переменной mode строкой 'p'. Если это так, то программа переходит в режим расшифровывания. Единственное различие между режимами расшифровывания и шифрования заключается в том, что в первом ключ является отрицательной версией самого себя. Например, если key — целое число 22, режим расшифровывания превращает его в -22. Причина объясняется в разделе «Шифрование/расшифровка каждой буквы» далее в этой главе.

Переменная translated будет содержать строку результата: либо шифротекст (если вы выполняете шифрование), либо открытый текст (если расшифровываете). В начале она содержит пустую строку, а затем к ее значению присоединяются зашифрованные или расшифрованные символы. Но прежде чем мы сможем начать присоединять символы к значению переменной translated, нам нужно зашифровать или расшифровать текст, чем мы и займемся в оставшейся части функции getTranslatedMessage().

Нахождение переданных строк с помощью строчного метода find()

Чтобы сдвигать буквы, выполняя шифрование или расшифровывание, нам сначала нужно преобразовать их в числа. Числом для каждой буквы в строке SYMBOLS будет индекс, который занимает буква. Поскольку буква «А» занимает индекс SYMBOLS[0], число 0 будет представлять прописную букву «А». Если бы мы захотели зашифровать ее с помощью ключа 3, то просто использовали бы операцию 0 + 3 для получения индекса зашифрованной буквы: SYMBOLS[3] или 'Г'.

Мы будем использовать строковый метод find(), который обнаруживает первое вхождение переданной строки в строке, в которой вызывается метод. В интерактивной оболочке введите следующие команды:

```
>>> 'Привет, мир!'.find('П')
0
>>> 'Привет, мир!'.find('и')
2
>>> 'Привет, мир!'.find('вет')
3
```

'Привет, мир!.find ('П') возвращает 0, потому что 'П' находится под первым индексом в строке 'Привет, мир!. Помните, индексы начинаются с 0, не с 1. Код 'Привет, мир!.find('и') возвращает 2, потому что первое вхождение строчной буквы 'и' находится в середине слова 'Привет'. Метод find() прекращает поиск после первого вхождения, поэтому вторая буква 'и' в слове

'мир' уже не имеет значения. Вы также можете искать строки с более чем одним символом. Начало строки 'вет' находится под индексом 3.

Если переданная строка не может быть найдена, метод find() возвращает -1.

```
>>> 'Привет, мир!'.find('пупс')
-1
```

Вернемся к программе «Шифр Цезаря». Код в строке 31 представляет собой цикл for, перебирающий каждый символ в строке message.

```
31.     for symbol in message:
32.         symbolIndex = SYMBOLS.find(symbol)
33.         if symbolIndex == -1: # Символ не найден в SYMBOLS.
34.             # Просто добавить этот символ без изменений.
35.             translated += symbol
```

Метод find() в строке 32 используется для получения индекса строки в symbol. Если метод find() возвращает -1, символ в переменной symbol будет просто присоединен к значению переменной translated без каких-либо изменений. Это означает, что любые символы, не являющиеся частью алфавита, такие как запятые и точки, не будут изменены.

Шифрование/расшифровка каждой буквы

Как только значение индекса буквы определено, прибавление ключа к этому значению осуществит смещение и выдаст индекс зашифрованной буквы.

Код в строке 38 производит это сложение для получения зашифрованной (или расшифрованной) буквы.

```
36.         else:
37.             # Зашифровать или расшифровать
38.             symbolIndex += key
```

Помните, что в строке 28 мы сделали целое число в переменной key отрицательным — для расшифровывания. Код, прибавляющий значение ключа, теперь будет вычитать его, так как прибавление отрицательного числа аналогично вычитанию.

Однако, если это сложение (или вычитание, если значение key отрицательно) заставляет переменную symbolIndex проходить последний индекс строки SYMBOLS, нам нужно вернуть его к 0 в начале списка. Это делается с помощью конструкции if, начинающейся в строке 40.

```
40.            if symbolIndex >= len(SYMBOLS):
41.                symbolIndex -= len(SYMBOLS)
42.            elif symbolIndex < 0:
43.                symbolIndex += len(SYMBOLS)
44.
45.            translated += SYMBOLS[symbolIndex]
```

Код в строке 40 проверяет, прошла ли переменная symbolIndex последний индекс, сравнивая его с длиной строки SYMBOLS. Если прошла, то код в строке 41 вычитает длину SYMBOLS из значения переменной symbolIndex. Если значение переменной symbolIndex теперь отрицательное, то расчет индексов должен начаться с другого конца строки SYMBOLS. Код в строке 42 проверяет, является ли значение symbolIndex отрицательным после прибавления к нему значения ключа расшифровывания. Если оно отрицательно, код в строке 43 прибавляет длину SYMBOLS к значению переменной symbolIndex.

Переменная symbolIndex теперь содержит индекс правильно зашифрованного или расшифрованного символа. Код SYMBOLS[symbolIndex] укажет на конкретный символ для этого индекса, и этот символ будет добавлен в конец значения переменной translated в строке 45.

Интерпретатор возвращается к строке 31, чтобы повторить то же для следующего символа в переменной message. Как только цикл завершается, функция возвращает зашифрованную (или расшифрованную) строку в translated в строке кода 46.

```
46.    return translated
```

Последняя строка кода функции getTranslatedMessage() возвращает строку translated.

Запуск программы

Запуск программы вызывает каждую из трех определенных ранее функций для получения от пользователя всех необходимых данных — режима, сообщения и ключа (mode, message и key).

```
48. mode = getMode()
49. message = getMessage()
50. key = getKey()
51. print('Преобразованный текст:')
52. print(getTranslatedMessage(mode, message, key))
```

Эти три значения передаются функции `getTranslatedMessage()`, возвращаемое значение которой (строка `translated`) выводится пользователю.

ДОБАВЛЕНИЕ НОВЫХ СИМВОЛОВ

Если вы хотите зашифровывать числа, пробелы и знаки препинания, просто добавьте их к значению переменной `SYMBOLS` в строке 2. Например, с помощью программы можно шифровать числа, пробелы и знаки препинания, изменив код в строке 2 на следующий:

```
2. SYMBOLS = 'АБВГДЕЕЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯабвгдеежзийклмнопрстуфхцчшщъыьэюя
1234567890!@#$%^&*()'
```

Обратите внимание, что значение переменной `SYMBOLS` содержит пробел после строчной буквы «я».

Также можно добавить еще больше символов в этот список. И не нужно менять остальную часть вашей программы, так как все строки кода, где необходим список символов, используют константу `SYMBOLS`.

Просто убедитесь, что каждый символ содержится в строке только один раз. Кроме того, вам нужно будет расшифровывать сообщения с помощью той же строки `SYMBOLS`, с помощью которой они были зашифрованы.

Полный перебор

Вот и весь шифр Цезаря. Впрочем, несмотря на то, что этот шифр может обмануть кого-то, кто не разбирается в криптографии, он не удержит сообщение в секрете от знатоков *криптоанализа*. В то время как криптография — это наука о создании кодов, криптоанализ изучает их взлом.

Весь смысл криптографии в том, что если даже зашифрованное сообщение попадет в чужие руки, то никто не сможет понять исходный текст. Давайте представим себя на месте взломщика кодов, и будто все, чем мы располагаем, — этот зашифрованный текст:

```
Цямхд ъжомуц игъд кмхрмф щшмлр тцпжйцт.
```

Полный перебор (метод «грубой силы», от англ. *brute force*) — это метод последовательного перебора всех возможных ключей вплоть до нахождения правильного. Поскольку существует только 66 возможных ключей, криптоаналитику будет легко написать программу для взлома, расшифровывающую сообщения с использованием всех возможных ключей. Затем он найдет ключ, который расшифровывает в чистый английский. Давайте добавим в программу режим полного перебора

Добавление режима полного перебора

Сначала измените код в строках 7, 9 и 12 в функции `getMode()` так, чтобы они выглядели следующим образом (изменения выделены жирным шрифтом):

```
5. def getMode():
6.     while True:
7.         print('Вы хотите зашифровать, расшифровать или взломать текст?')
8.         mode = input().lower()
9.         if mode in ['зашифровать', 'з', 'расшифровать', 'р', 'взломать', 'в']:
10.             return mode
11.         else:
12.  print('Введите "зашифровать" или "з" для зашифровки или "расшифровать" или "р" для расшифровки или "взломать" или "в" для взлома.')
```

Этот код позволит пользователю выбрать в качестве режима полный перебор (взлом шифра).

Затем внесите следующие изменения в основной код программы:

```
48. mode = getMode()
49. message = getMessage()
50. if mode[0] != 'в':
51.     key = getKey()
52. print('Преобразованный текст:')
53. if mode[0] != 'в':
54.     print(getTranslatedMessage(mode, message, key))
55. else:
56.    for key in range(1, MAX_KEY_SIZE + 1):
57.        print(key, getTranslatedMessage('расшифровать', message, key))
```

Если пользователь не выбрал режим полного перебора, программа спрашивает у него ключ, вызывает функцию `getTranslatedMessage()` и выводит «переведенную» строку. Если же пользователь выбрал режим полного перебора, цикл `getTranslatedMessage()` выполняет перебор от `1` до `MAX_KEY_SIZE` (то есть до `66`). Помните, что функция `range()` возвращает список целых чисел вплоть до второго параметра, но не включая его, вот почему мы добавляем `+ 1`. Затем программа выводит каждый возможный «перевод» сообщения (включая значение ключа, используемого при преобразовании). Ниже показан пример работы модифицированной программы.

```
Вы хотите зашифровать, расшифровать или взломать текст?
взломать
Введите текст:
Цямхд ъжомуц игъд кмхрмф щшмлр тцпжйцт.
Преобразованный текст:
 1 Хюлфг щенлтх звщг йлфплу шчлкп схоеихс.
 2 Фэкув шемксф жбшв икуокт чцкйо рфнезфр.
 3 Уьйтб чдлйру еачб зйтнйс цхйин пумджуп.
 4 Тыиса цгкипт еЯца жисмир хфизм отлгето.
 5 Съзря хвйзос дЮхЯ езрлзп фузжл нсквесн.
 6 РщжпЮ фбижнр гЭфЮ ежпкжо утжек мрйбдрм.
 7 ПшеоЭ уаземп вЬуЭ деойен тсеей лпиагпл.
 8 ОченЬ тЯжело бЫтЬ гением среди козЯвок.
 9 НцдмЫ сЮедкн аЪсЫ вдмздл рпдгз йнжЮбнй.
10 МхглЪ рЭегйм ЯЩрЪ бглжгк погвж имеЭами.
11 ЛфвкЩ пЬдвил ЮШпЩ авкевй онвбе злеЬЯлз.
12 КубйШ оЫгбзк ЭЧоШ Ябйеби нмбае жкдЫЮкж.
--пропуск--
```

Просмотрев каждую строку, вы увидите, что восьмое сообщение не бессмысленность, а понятный русский текст! Криптоаналитик придет к выводу, что исходный ключ для этого зашифрованного текста был `8`. Подобный метод было бы проблематично осуществить в дни Юлия Цезаря и Римской империи, но сегодня у нас есть компьютеры, которые могут за короткое время перебрать миллионы или даже миллиарды ключей.

Заключение

Компьютеры хороши в математических расчетах. Когда мы создаем систему для преобразования некоторой части информации в числа (как с тек-

стом и порядковыми числительными или с геопозициями и системами координат), компьютерные программы могут обрабатывать эти числа быстро и эффективно. Большая часть разработки программы заключается в том, как представить информацию, которой вы хотите управлять, в виде понятных Python значений.

При том, что наша программа «Шифр Цезаря» может шифровать сообщения, смысл которых останется тайной для людей, вооруженных карандашом и бумагой, программа не сможет сохранить их в секрете от тех, кто знает, как обрабатывать информацию с помощью компьютеров (это подтверждает режим полного перебора).

В главе 15 мы создадим игру «Реверси» (также известную как «Отелло»). Искусственный интеллект в этой игре намного более продвинут, чем в программе «Крестики-нолики» в главе 10. Он будет настолько хорош, что в большинстве случаев вы не сможете его обыграть!

15

ИГРА «РЕВЕРСИ»

В этой главе мы создадим игру «Реверси», также известную под названием «Отелло». В этой настольной игре для двух игроков используется сетка, поэтому нам пригодится декартова система с координатами *x* и *y*. У нашей версии игры будет компьютерный искусственный интеллект, более продвинутый, чем ИИ игры «Крестики-нолики» из главы 10. Вообще-то этот ИИ настолько хорош, что он, скорее всего, будет побеждать вас почти в каждой игре (во всяком случае он непременно обыгрывает автора).

В ЭТОЙ ГЛАВЕ РАССМАТРИВАЮТСЯ СЛЕДУЮЩИЕ ТЕМЫ:

- Как играть в «Реверси»
- Функция `bool()`
- Моделирование ходов на игровом поле
- Программирование искусственного интеллекта в игре «Реверси»

Как играть в «Реверси»

В игре «Реверси» используется поле размером 8×8 клеток и фишки — черные с одной стороны и белые с другой (вместо этого мы будем использовать буквы *O* и *X*). В начале игры поле выглядит так, как показано на рис. 15.1.

Два игрока по очереди выставляют на поле фишки выбранного ими цвета — черные или белые. Когда игрок помещает фишку на поле, все фишки противника, которые находятся между новой фишкой и остальными фишками игрока, переворачиваются.

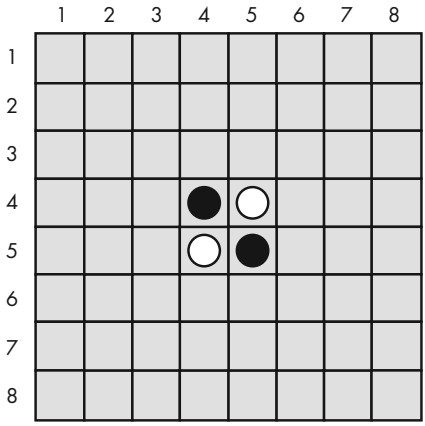


Рис. 15.1. В начале игры на поле выставляются две белые фишки и две черные

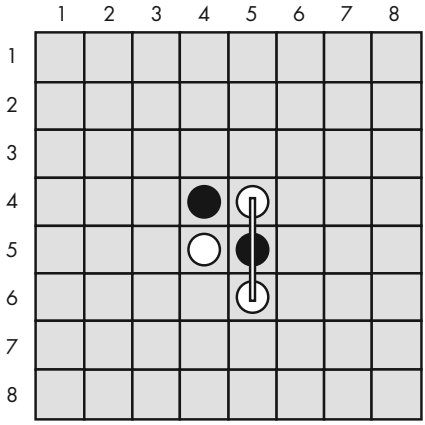


Рис. 15.2. Игрок белыми ставит новую фишку

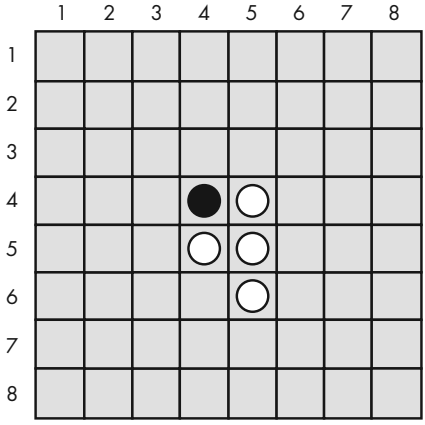


Рис. 15.3. Ход игрока белыми привел к тому, что одна из фишек игрока черными перевернулась

Например, когда игрок белыми помещает новую белую фишку в позицию (5, 6), как показано на рис. 15.2, черная фишка в позиции (5, 5) оказывается между двумя белыми, поэтому она переворачивается и тоже становится белой, как показано на рис. 15.3. Цель игры состоит в том, чтобы фишек вашего цвета было больше, чем фишек противника.

Следом игрок черными мог сделать аналогичный ход, поместив свою фишку в позицию (4, 6) и этим перевернув белую фишку в позиции (4, 5). Результатом стала бы ситуация, изображенная на рис. 15.4.

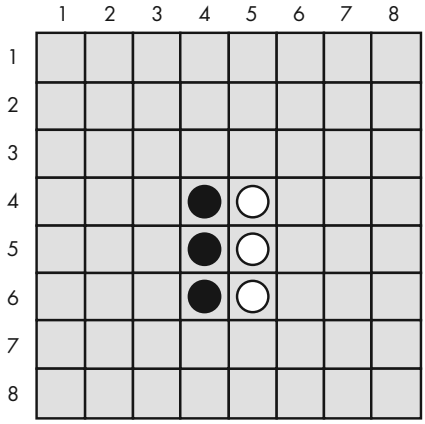


Рис. 15.4. Игрок черными выставил новую фишку, перевернув одну из фишек игрока белыми

Фишки во всех направлениях остаются перевернутыми, пока они находятся между новой фишкой игрока и существующей фишкой такого же цвета. На рис. 15.5 игрок белыми помещает фишку в позицию (3, 6) и переворачивает черные фишки в двух направлениях (обозначены линиями). Результат показан на рис. 15.6.

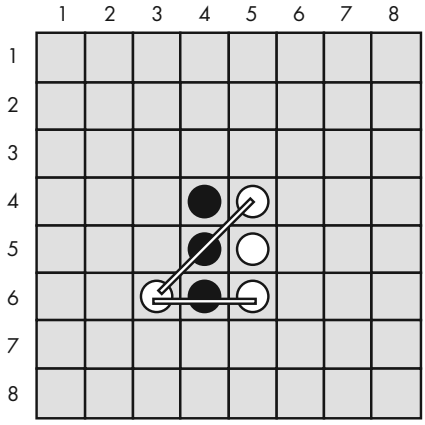


Рис. 15.5. Второй ход игрока белыми в позицию (3, 6) перевернет две фишки игрока черными

За один или два хода каждый игрок может быстро перевернуть много фишек. Игроки всегда должны делать ход, который переворачивает хотя бы одну фишку. Игра заканчивается либо когда игрок не может сделать ход, либо когда поле заполнено фишками. Побеждает тот игрок, у которого осталось больше фишек своего цвета.

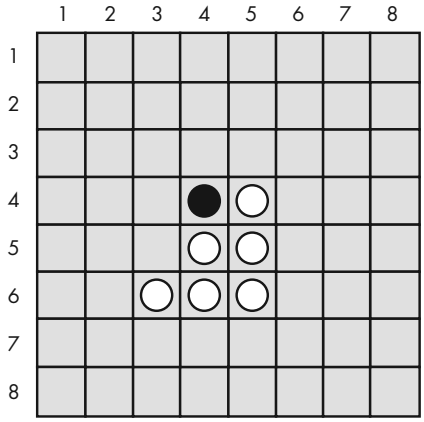


Рис. 15.6. Поле после второго хода игрока белыми

Искусственный интеллект, который мы создадим для этой игры, будет искать на поле любые угловые ходы, которые сможет сделать. Если доступных угловых ходов нет, компьютер выберет ход, который перевернет больше всего фишек.

Пример запуска игры «Реверси»

Вот что видит пользователь, запустив программу «Реверси». Текст, который вводит игрок, выделен полужирным шрифтом.

```
Приветствуем в игре "Реверси"!
Вы играете за X или O?
x
Человек ходит первым.
  12345678
 +--------+
1|        |1
2|        |2
3|        |3
4|   XO   |4
5|   OX   |5
6|        |6
7|        |7
8|        |8
 +--------+
  12345678
```

```
Ваш счет: 2. Счет компьютера: 2.
Укажите ход, текст "выход" для завершения игры или "подсказка" для вывода подсказки.
53
  12345678
 +--------+
1|        |1
2|        |2
3|    X   |3
4|   XX   |4
5|   OX   |5
6|        |6
7|        |7
8|        |8
 +--------+
  12345678
Ваш счет: 4. Счет компьютера: 1.
Нажмите клавишу Enter для просмотра хода компьютера.
```

--*пропуск*--

```
 12345678
 +--------+
1|XXXXOOOO|1
2|XXXXXOOO|2
3|XXXXXXOO|3
4|XXXXXXXO|4
5|XOXXOXXX|5
6|XOXOXOXX|6
7|XXOOOXXX|7
8|XXXXXXXX|8
 +--------+
  12345678
X набрал 46 очков. O набрал 18 очков.
Вы победили компьютер, обогнав его на 28 очков! Поздравления!
Хотите сыграть еще раз? (да или нет)
нет
```

Как видно из примера, автор довольно уверенно победил компьютер, 46 очков против 18. Чтобы помочь начинающему игроку, мы добавим в игру подсказки. Игрок может ввести слово `подсказка` во время своего хода, включив или выключив режим подсказок. Когда режим подсказок включен, все допустимые ходы игрока отобразятся на игровом поле в виде точек (`.`), например, так:

```
  12345678
 +--------+
1|      . |1
2|  X O.  |2
3|   XO.  |3
4|  XXO.  |4
5|   OO.  |5
6|   ..O  |6
7|      . |7
8|        |8
 +--------+
  12345678
```

Как видите, игрок может поместить фишку в позиции (6, 1), (6, 2), (4, 6), (7, 7) и несколько других, основываясь на выданных ему подсказках.

Исходный код игры «Реверси»

«Реверси» — гигантская программа по сравнению с нашими предыдущими играми, она содержит почти 300 строк кода! Но не беспокойтесь: многие из них — это комментарии или пустые строки, оставленные, чтобы разделить код и сделать его более удобочитаемым.

Как и в других наших программах, сначала мы создадим несколько функций, которые будут вызываться в основном коде игры. Около 250 первых строк кода предназначены для этих вспомогательных функций, а последние 30 строк реализуют саму игру «Реверси».

В редакторе файлов создайте новый файл, выбрав команду меню **File ⇒ New File** (Файл ⇒ Новый файл). В открывшемся окне введите приведенный ниже исходный код и сохраните файл под именем *reversegam.py.* Нажмите клавишу **F5** и запустите программу. Если при выполнении программы

возникают ошибки, сравните код, который вы набрали, с оригинальным кодом с помощью онлайн-инструмента на сайте **inventwithpython.com/diff/**.

reversegam.py

```
1. # "Реверси": клон "Отелло".
2. import random
3. import sys
4. WIDTH = 8  # Игровое поле содержит 8 клеток по ширине.
5. HEIGHT = 8 # Игровое поле содержит 8 клеток по высоте.
6. def drawBoard(board):
7.     # Вывести игровое поле, переданное этой функции. Ничего не возвращать.
8.     print('  12345678')
9.     print(' +--------+')
10.     for y in range(HEIGHT):
11.         print('%s|' % (y+1), end='')
12.         for x in range(WIDTH):
13.             print(board[x][y], end='')
14.         print('|%s' % (y+1))
15.     print(' +--------+')
16.     print('  12345678')
17.
18. def getNewBoard():
19.     # Создать структуру данных нового чистого игрового поля.
20.     board = []
21.     for i in range(WIDTH):
22.         board.append([' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '])
23.     return board
24.
25. def isValidMove(board, tile, xstart, ystart):
26.     # Вернуть False, если ход игрока в клетку с координатами xstart, ystart — недопустимый.
27.  # Если это допустимый ход, вернуть список клеток, которые "присвоил" бы игрок, если бы сделал туда ход.
28.     if board[xstart][ystart] != ' ' or not isOnBoard(xstart, ystart):
29.         return False
30.
31.     if tile == 'X':
32.         otherTile = 'O'
33.     else:
34.         otherTile = 'X'
35.
36.     tilesToFlip = []
37.   for xdirection, ydirection in [[0, 1], [1, 1], [1, 0], [1, -1], [0, -1], [-1, -1], [-1, 0], [-1, 1]]:
```

```
38.         x, y = xstart, ystart
39.         x += xdirection # Первый шаг в направлении x
40.         y += ydirection # Первый шаг в направлении y
41.         while isOnBoard(x, y) and board[x][y] == otherTile:
42.             # Продолжать двигаться в этом направлении x и y.
43.             x += xdirection
44.             y += ydirection
45.             if isOnBoard(x, y) and board[x][y] == tile:
46.  # Есть фишки, которые можно перевернуть. Двигаться в обратном направлении до достижения исходной клетки, отмечая все фишки на этом пути.
47.                 while True:
48.                     x -= xdirection
49.                     y -= ydirection
50.                     if x == xstart and y == ystart:
51.                         break
52.                     tilesToFlip.append([x, y])
53.
54.     if len(tilesToFlip) == 0: # Если ни одна из фишек не перевернулась, это недопустимый
ход.
55.         return False
56.     return tilesToFlip
57.
58. def isOnBoard(x, y):
59.     # Вернуть True, если координаты есть на игровом поле.
60.     return x >= 0 and x <= WIDTH - 1 and y >= 0 and y <= HEIGHT — 1
61.
62. def getBoardWithValidMoves(board, tile):
63.    # Вернуть новое поле с точками, обозначающими допустимые ходы, которые может сделать игрок.
64.     boardCopy = getBoardCopy(board)
65.
66.     for x, y in getValidMoves(boardCopy, tile):
67.         boardCopy[x][y] = '.'
68.     return boardCopy
69.
70. def getValidMoves(board, tile):
71. # Вернуть список списков с координатами x и y допустимых ходов для данного игрока на данном игровом поле.
72.     validMoves = []
73.     for x in range(WIDTH):
74.         for y in range(HEIGHT):
75.         if isValidMove(board, tile, x, y) != False:
76.             validMoves.append([x, y])
77.     return validMoves
78.
```

```
79. def getScoreOfBoard(board):
80.     # Определить количество очков, подсчитав фишки. Вернуть словарь с ключами 'X' и 'O'.
81.     xscore = 0
82.     oscore = 0
83.     for x in range(WIDTH):
84.         for y in range(HEIGHT):
85.             if board[x][y] == 'X':
86.                 xscore += 1
87.             if board[x][y] == 'O':
88.                 oscore += 1
89.     return {'X':xscore, 'O':oscore}
90.
91. def enterPlayerTile():
92.     # Позволить игроку ввести выбранную фишку.
93.  # Возвращает список с фишкой игрока в качестве первого элемента и фишкой компьютера в качестве второго.
94.     tile = ''
95.     while not (tile == 'X' or tile == 'O'):
96.         print('Вы играете за X или O?')
97.         tile = input().upper()
98.
99.     # Первый элемент в списке — фишка игрока, второй элемент — фишка компьютера.
100.     if tile == 'X':
101.         return ['X', 'O']
102.     else:
103.         return ['O', 'X']
104.
105. def whoGoesFirst():
106.     # Случайно выбрать, кто ходит первым.
107.     if random.randint(0, 1) == 0:
108.         return 'Компьютер'
109.     else:
110.         return 'Человек'
111.
112. def makeMove(board, tile, xstart, ystart):
113.  # Поместить фишку на игровое поле в позицию xstart, ystart и перевернуть какую-либо фишку противника.
114.     # Вернуть False, если это недопустимый ход; вернуть True, если допустимый.
115.     tilesToFlip = isValidMove(board, tile, xstart, ystart)
116.
117.     if tilesToFlip == False:
118.         return False
119.
120.     board[xstart][ystart] = tile
```

```
121.     for x, y in tilesToFlip:
122.         board[x][y] = tile
123.     return True
124.
125. def getBoardCopy(board):
126.     # Сделать копию списка board и вернуть ее.
127.     boardCopy = getNewBoard()
128.
129.     for x in range(WIDTH):
130.         for y in range(HEIGHT):
131.             boardCopy[x][y] = board[x][y]
132.
133.     return boardCopy
134.
135. def isOnCorner(x, y):
136.     # Вернуть True, если указанная позиция находится в одном из четырех углов.
137.     return (x == 0 or x == WIDTH - 1) and (y == 0 or y == HEIGHT - 1)
138.
139. def getPlayerMove(board, playerTile):
140.     # Позволить игроку ввести свой ход.
141.     # Вернуть ход в виде [x, y] (или вернуть строки 'подсказка' или 'выход').
142.     DIGITS1TO8 = '1 2 3 4 5 6 7 8'.split()
143.     while True:
144.     print('Укажите ход, текст "выход" для завершения игры или "подсказка" для вывода подсказки.')
145.         move = input().lower()
146.         if move == 'выход' or move == 'подсказка':
147.             return move
148.
149.         if len(move) == 2 and move[0] in DIGITS1TO8 and move[1] in DIGITS1TO8:
150.             x = int(move[0]) - 1
151.             y = int(move[1]) - 1
152.             if isValidMove(board, playerTile, x, y) == False:
153.                 continue
154.             else:
155.                 break
156.         else:
157.             print('Это недопустимый ход. Введите номер столбца (1-8) и номер ряда (1-8).')
158.             print('К примеру, значение 81 перемещает в верхний правый угол.')
159.
160.     return [x, y]
161.
162. def getComputerMove(board, computerTile):
```

```
163.     # Учитывая данное игровое поле и данную фишку компьютера, определить,
164.     # куда сделать ход, и вернуть этот ход в виде списка [x, y].
165.     possibleMoves = getValidMoves(board, computerTile)
166.     random.shuffle(possibleMoves) # Сделать случайным порядок ходов
167.
168.     # Всегда делать ход в угол, если это возможно.
169.     for x, y in possibleMoves:
170.         if isOnCorner(x, y):
171.             return [x, y]
172.
173.     # Найти ход с наибольшим возможным количеством очков.
174.     bestScore = -1
175.     for x, y in possibleMoves:
176.         boardCopy = getBoardCopy(board)
177.         makeMove(boardCopy, computerTile, x, y)
178.         score = getScoreOfBoard(boardCopy)[computerTile]
179.         if score > bestScore:
180.             bestMove = [x, y]
181.             bestScore = score
182.     return bestMove
183.
184. def printScore(board, playerTile, computerTile):
185.     scores = getScoreOfBoard(board)
186.     print('Ваш счет: %s. Счет компьютера: %s.' % (scores[playerTile], scores[computerTile]))
187.
188. def playGame(playerTile, computerTile):
189.     showHints = False
190.     turn = whoGoesFirst()
191.     print(turn + ' ходит первым.')
192.
193.     # Очистить игровое поле и выставить стартовые фишки.
194.     board = getNewBoard()
195.     board[3][3] = 'X'
196.     board[3][4] = 'O'
197.     board[4][3] = 'O'
198.     board[4][4] = 'X'
199.
200.     while True:
201.         playerValidMoves = getValidMoves(board, playerTile)
202.         computerValidMoves = getValidMoves(board, computerTile)
203.
204.         if playerValidMoves == [] and computerValidMoves == []:
```

```
205.             return board # Ходов нет ни у кого, так что окончить игру.
206.
207.         elif turn == 'Человек': # Ход человека
208.             if playerValidMoves != []:
209.                 if showHints:
210.                     validMovesBoard = getBoardWithValidMoves(board, playerTile)
211.                     drawBoard(validMovesBoard)
212.                 else:
213.                     drawBoard(board)
214.                 printScore(board, playerTile, computerTile)
215.
216.                 move = getPlayerMove(board, playerTile)
217.                 if move == 'выход':
218.                     print('Благодарим за игру!')
219.                     sys.exit() # Завершить работу программы.
220.                 elif move == 'подсказка':
221.                     showHints = not showHints
222.                     continue
223.                 else:
224.                     makeMove(board, playerTile, move[0], move[1])
225.             turn = 'Компьютер'
226.
227.         elif turn == 'Компьютер': # Ход компьютера
228.             if computerValidMoves != []:
229.                 drawBoard(board)
230.                 printScore(board, playerTile, computerTile)
231.
232.                 input('Нажмите клавишу Enter для просмотра хода компьютера.')
233.                 move = getComputerMove(board, computerTile)
234.                 makeMove(board, computerTile, move[0], move[1])
235.             turn = 'Человек'
236.
237.
238.
239. print('Приветствуем в игре "Реверси"!')
240.
241. playerTile, computerTile = enterPlayerTile()
242.
243. while True:
244.     finalBoard = playGame(playerTile, computerTile)
245.
246.     # Отобразить итоговый счет.
```

```
247.     drawBoard(finalBoard)
248.     scores = getScoreOfBoard(finalBoard)
249.     print('X набрал %s очков. O набрал %s очков.' % (scores['X'], scores['O']))
250.     if scores[playerTile] > scores[computerTile]:
251.         print('Вы победили компьютер, обогнав его на %s очков! Поздравления!' %
(scores[playerTile] - scores[computerTile]))
252.     elif scores[playerTile] < scores[computerTile]:
253.         print('Вы проиграли. Компьютер победил вас, обогнав на %s очков.' %
(scores[computerTile] - scores[playerTile]))
254.     else:
255.         print('Ничья!')
256.
257.     print('Хотите сыграть еще раз? (да или нет)')
258.     if not input().lower().startswith('д'):
259.         break
```

Импорт модулей и создание констант

Как мы уже научились, начинаем программу с импорта модулей.

```
1. # "Реверси": клон "Отелло".
2. import random
3. import sys
4. WIDTH = 8  # Игровое поле содержит 8 клеток по ширине.
5. HEIGHT = 8 # Игровое поле содержит 8 клеток по высоте.
```

Код в строке 2 импортирует модуль `random` для получения доступа к его функциям `randint()` и `choice()`. Код в строке 3 импортирует модуль `sys` для получения доступа к его функции `exit()`.

Код в строках 4 и 5 задают две константы, `WIDTH` и `HEIGHT`, которые используются для настройки игрового поля.

Структура данных игрового поля

Давайте разберемся со структурой данных игрового поля. Она представляет собой список списков, точно так же, как в игре «Охотник за сокровищами» из главы 13. Список списков создается за тем, чтобы с помощью `board[x][y]` можно было представлять символ, находящийся в клетке, расположенной в позиции `x` по оси *x* (перемещение влево/вправо) и в позиции `y` по оси *y* (вверх/вниз).

Этим символом может быть либо `' '` (пробел, представляющий пустую клетку), `'.'` (точка, указывающая на возможный ход в режиме подсказок) или `'X'` или `'O'` (буквы, представляющие фишки). Параметр `board` обозначает эту структуру данных вида список списков.

Важно отметить, что если значения координат *x* и *y* игрового поля будут варьироваться от 1 до 8, то индексы структуры данных списка — от 0 до 7. Нам придется скорректировать код для учета этого.

Отображение на экране структуры данных игрового поля

Структура данных игрового поля — это всего лишь список в `Python`, но нам нужен более удобный способ представить ее на экране. Функция `drawBoard()` принимает структуру данных поля и отображает ее на экране, чтобы игрок знал, где находятся фишки.

```
6. def drawBoard(board):
7.     # Вывести игровое поле, переданное этой функции. Ничего не возвращать.
8.     print('  12345678')
9.     print(' +--------+')
10.     for y in range(HEIGHT):
11.         print('%s|' % (y+1), end='')
12.         for x in range(WIDTH):
13.             print(board[x][y], end='')
14.         print('|%s' % (y+1))
15.     print(' +--------+')
16.     print('  12345678')
```

Функция `drawBoard()` выводит игровое поле в текущем состоянии, основываясь на структуре данных в параметре `board`.

Код в строке 8 — первый из вызовов функции `print()`, выполняемый для каждого игрового поля; он выводит метки для оси *x* в верхней части поля. Код в строке 9 выводит верхнюю горизонтальную строку поля. Цикл `for` в строке кода 10 будет выполняться восемь раз, по одному разу для каждого ряда. Код в строке 11 выводит метку для оси *y* в левой части поля и содержит аргумент — ключевое слово `end=''`, чтобы вместо новой строки поля не выводить ничего.

Это связано с тем, что другой цикл в строке 12 (который также выполняется восемь раз, по одному для каждого столбца в ряде) выводит каждую клетку вместе с символом `X`, `O`, `.` или пробелом в зависимости от того, что хранится в `board[x][y]`. Вызов функции `print()` в строке кода 13 внутри этого цикла также содержит аргумент — ключевое слово `end=''`, чтобы символ

новой строчки поля не выводился. Это создаст на экране одну строку вида `'1|XXXXXXXX|1'` (если каждое из значений `board[x][y]` было `'X'`).

После выполнения внутреннего цикла в строках 15 и 16 вызывается функция `print()`, которая выводит нижнюю горизонтальную строчку и метки оси *x*.

Цикл `for` в строке 13 формирует игровое поле.

```
  12345678
 +--------+
1|XXXXXXXX|1
2|XXXXXXXX|2
3|XXXXXXXX|3
4|XXXXXXXX|4
5|XXXXXXXX|5
6|XXXXXXXX|6
7|XXXXXXXX|7
8|XXXXXXXX|8
 +--------+
  12345678
```

Конечно, вместо `X` некоторые клетки на поле будут отмечены знаком другого игрока (`O`), точкой (`.`), если включен режим подсказок, или пробелом для пустых клеток.

Создание структуры данных нового игрового поля

Функция `drawBoard()` выводит структуру данных поля на экран, но нам также нужно каким-то образом создать эти структуры данных. Функция `getNewBoard()` возвращает список восьми списков, каждый из которых содержит восемь строк `' '`. Они представляют собой чистое игровое поле.

```
18. def getNewBoard():
19.     # Создать структуру данных нового чистого игрового поля.
20.     board = []
21.     for i in range(WIDTH):
22.         board.append([' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '])
23.     return board
```

Код в строке 20 создает список, содержащий внутренние списки. Цикл `for` добавляет внутрь этого списка восемь списков. Эти внутренние списки содержат восемь строк для представления восьми пустых клеток на поле.

Вместе этот код создает поле с 64 пустыми клетками — начальный вид поля игры «Реверси».

Проверка допустимости хода

Имея структуру данных игрового поля, фишку игрока и координаты *x* и *y* его хода, функция `isValidMove()` должна вернуть значение `True`, если правила игры «Реверси» позволяют сделать ход по этим координатам, и `False`, если не позволяют. Чтобы ход был допустимым, он должен быть сделан в пределах игрового поля, а также перевернуть хотя бы одну из фишек противника.

Эта функция использует несколько координат *x* и *y* на поле, так что переменные `xstart` и `ystart` отслеживают координаты *x* и *y* исходного хода.

```
25. def isValidMove(board, tile, xstart, ystart):
26.     # Вернуть False, если ход игрока в клетку с координатами xstart, ystart — недопустимый.
27.     # Если это допустимый ход, вернуть список клеток, которые "присвоил" бы игрок, если бы
сделал туда ход.
28.     if board[xstart][ystart] != ' ' or not isOnBoard(xstart, ystart):
29.         return False
30.
31.     if tile == 'X':
32.         otherTile = 'O'
33.     else:
34.         otherTile = 'X'
35.
36.     tilesToFlip = []
```

Код в строке 28, используя функцию `isOnBoard()` (которую мы определим позже), проверяет, принадлежат ли координаты *x* и *y* игровому полю и пуста ли эта клетка. Эта функция проверяет, что значения координат *x* и *y* находятся в диапазоне между `0` и значениями `WIDTH` (ширины) или `HEIGHT` (высоты) игрового поля минус `1`.

Фишка игрока (человека или компьютера) содержится в параметре `tile`, но этой функции должна быть также известна фишка противника. Если фишка игрока — `X`, то, очевидно, фишка противника — `O`, и наоборот. Для этого в строках 31–34 мы используем инструкцию `if-else`.

Наконец, если заданные координаты *x* и *y* допустимы, функция `isValidMove()` возвращает список всех фишек противника, которые будут перевернуты этим ходом. Мы создаем новый пустой список, `tilesToFlip`, который будем использовать для хранения всех координат фишки.

Проверка каждого из восьми направлений

Чтобы ход был допустимым, игрок должен перевернуть хотя бы одну из фишек противника, зажав ее между новой фишкой и одной из своих старых фишек. Это значит, что новая фишка должна быть рядом с одной из фишек противника.

Цикл `for` в строке 37 выполняет перебор списка списков, содержащий направления, в которых программа будет проверять фишку противника.

```
37.   for xdirection, ydirection in [[0, 1], [1, 1], [1, 0], [1, -1], [0, -1], [-1, -1], [-1, 0], [-1, 1]]:
```

Игровое поле представляет собой декартову систему координат с направлениями *x* и *y*. Есть восемь направлений, которые нужно проверить: вверх, вниз, влево и вправо, а также четыре диагональных направления. Каждый из восьми двухэлементных списков в списке в строке 37 используется для проверки одного из этих направлений. Программа проверяет направление, прибавляя первое значение в двухэлементном списке к координате *x*, а второе значение — к координате *y*.

Поскольку координаты *x* увеличиваются по мере продвижения вправо, вы можете проверить правое направление, прибавив `1` к значению координаты *x*. Таким образом, список `[1, 0]` прибавляет `1` к значению координаты *x* и `0` к значению координаты *y*. Принцип проверки направления влево противоположен: вычитаете `1` (то есть прибавляете `-1`) из значения координаты *x*.

Но чтобы проверить диагональные направления, нужно прибавлять к значениям обеих координат или вычитать из них. Например, прибавление `1` к значению координаты *x* и прибавление `-1` к значению координаты *y* приведет к проверке диагонального направления вправо-вверх.

На рис. 15.7 показана диаграмма, позволяющая легче запомнить, какое направление представлено каким двухэлементным списком.

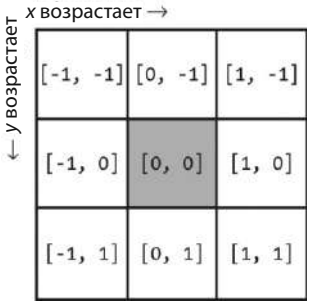


Рис. 15.7. Каждый двухэлементный список представляет одно из восьми направлений

Цикл `for` в строке 37 перебирает каждый из двухэлементных списков, чтобы проверить каждое направление. Внутри цикла `for` в строке 38 переменным `x` и `y` с помощью множественного присваивания назначены те же значения, что и переменным `xstart` и `ystart` соответственно. Переменным `xdirection` и `ydirection` присваиваются значения из одного из двухэлементных списков; они изменяют значения переменных `x` и `y` в соответствии с проверяемым направлением в той итерации цикла `for`.

```
37.  for xdirection, ydirection in [[0, 1], [1, 1], [1, 0], [1, -1], [0, -1], [-1, -1], [-1, 0], [-1, 1]]:
38.          x, y = xstart, ystart
39.          x += xdirection # Первый шаг в направлении x
40.          y += ydirection # Первый шаг в направлении y
```

Значения переменных `xstart` и `ystart` останутся такими же, чтобы программа могла запомнить, с какой клетки началась игра.

Помните, для того, чтобы ход был допустимым, он должен быть одновременно сделан в пределах игрового поля и рядом с одной из фишек другого игрока. (В противном случае не будет доступных для переворота фишек противника, а игрок должен перевернуть, по крайней мере, одну фишку, чтобы быть допустимым.) Код в строке 41 проверяет это условие, и, если оно не истинно, интерпретатор возвращается к инструкции `for` для выполнения проверки в следующем направлении.

```
41.          while isOnBoard(x, y) and board[x][y] == otherTile:
42.              # Продолжать двигаться в этом направлении x и y.
43.              x += xdirection
44.              y += ydirection
```

Но если первая проверенная клетка содержит фишку противника, программа должна продолжать проверять фишки противника в этом направлении, пока не достигнет одной из фишек игрока или конца поля. Следующая фишка в том же направлении проверяется с помощью повторного использования переменных `xdirection` и `ydirection`, чтобы присвоить переменным `x` и `y` следующие координаты на проверку. Так что программа изменяет значения переменных `x` и `y` в строках 43 и 44.

Определение наличия фишек, которые можно перевернуть

Затем мы проверяем, есть ли соседние фишки, которые можно перевернуть.

```
45.             if isOnBoard(x, y) and board[x][y] == tile:
46.                 # Есть фишки, которые можно перевернуть. Двигаться в обратном направлении до
достижения исходной клетки, отмечая все фишки на этом пути.
47.                 while True:
48.                     x -= xdirection
49.                     y -= ydirection
50.                     if x == xstart and y == ystart:
51.                         break
52.                     tilesToFlip.append([x, y])
```

Инструкция `if` в строке 45 проверяет, занята ли координата собственной фишкой игрока. Эта фишка будет служить отметкой конца «бутерброда», образованного фишками игрока, которые окружают фишки противника. Нам также необходимо записать координаты всех фишек противника, которые нужно перевернуть.

В строках 48 и 49 цикл `while` перемещает `x` и `y` в обратном направлении. До тех пор пока `x` и `y` не вернутся на исходную позицию `xstart` и `ystart`, из них вычитаются `xdirection` и `ydirection`, а каждая позиция `x` и `y` присоединяется к списку `tilesToFlip`. Когда `x` и `y` достигли клетки с координатами `xstart` и `ystart`, код в строке 51 прерывает выполнение цикла. Поскольку исходная клетка `xstart` и `ystart` пуста (мы удостоверились в этом в строках 28 и 29), условие цикла `while` в строке 41 примет значение `False`. Программа переходит к строке 37, а цикл `for` проверяет следующее направление.

Так цикл `for` проверяет все восемь направлений. После завершения этого цикла список `tilesToFlip` будет содержать координаты *x* и *y* всех фишек нашего оппонента, которые перевернулись бы, если бы игрок походил бы в `xstart`, `ystart`. Помните, функция `isValidMove()` только проверяет, был ли исходный ход допустимым; она не занимается реальным постоянным изменением структуры данных игрового поля.

Если ни одна из фишек противника не перевернулась ни в одном из восьми направлений, то `tilesToFlip` будет пустым списком.

```
54.     if len(tilesToFlip) == 0: # Если ни одна из фишек не перевернулась, это недопустимый ход.
55.         return False
56.     return tilesToFlip
```

Это признак того, что такой ход недопустим, и функция `isValidMove()` должна вернуть значение `False`. Иначе функция `isValidMove()` возвращает `tilesToFlip`.

Проверка допустимости координат

Функция `isOnBoard()` вызывается из функции `isValidMove()`. Она выполняет простую проверку, чтобы определить, находятся ли данные координаты *x* и *y* в пределах игрового поля. Например, клетки с координатой *x* `4` и координатой *y* `9999` не существует, так как координаты *y* заканчиваются `7`, что равно `WIDTH - 1` или `HEIGHT - 1`.

```
58. def isOnBoard(x, y):
59.     # Вернуть True, если координаты есть на игровом поле.
60.     return x >= 0 and x <= WIDTH - 1 and y >= 0 and y <= HEIGHT - 1
```

Вызов этой функции служит сокращением логического выражения в строке 72, которое проверяет, находятся ли значения и `x`, и `y` между `0` и `WIDTH - 1` или `0` и `HEIGHT - 1` (то есть в диапазоне от `0` до`7`).

Получение списка со всеми допустимыми ходами

Теперь давайте создадим режим подсказок, который будет отображать поле со всеми возможными ходами, отмеченными на нем. Функция `getBoardWithValidMoves()` возвращает структуру данных игрового поля, которая содержит точки (`.`) во всех клетках, ходы в которые допустимы.

```
62. def getBoardWithValidMoves(board, tile):
63.     # Вернуть новое поле с точками, обозначающими допустимые ходы, которые может сделать
игрок.
64.     boardCopy = getBoardCopy(board)
65.
66.     for x, y in getValidMoves(boardCopy, tile):
67.         boardCopy[x][y] = '.'
68.     return boardCopy
```

Эта функция создает копию структуры данных поля `board` с именем `boardCopy` (возвращаемую функцией `getBoardCopy()` в строке 64) вместо того, чтобы изменять переданную ей в параметре `board`. Код в строке 66 вызывает функцию `getValidMoves()`, чтобы получить список координат *x* и *y* со всеми допустимыми ходами, которые игрок может сделать. Копия игрового поля отмечается точками в соответствующих клетках и возвращается.

Функция getValidMoves() возвращает список двухэлементных списков, в которых содержатся координаты *x* и *y* всех допустимых ходов фишки tile для структуры данных поля в параметре board.

```
70. def getValidMoves(board, tile):
71.     # Вернуть список списков с координатами x и y допустимых ходов для данного игрока на
данном игровом поле.
72.     validMoves = []
73.     for x in range(WIDTH):
74.         for y in range(HEIGHT):
75.         if isValidMove(board, tile, x, y) != False:
76.             validMoves.append([x, y])
77.     return validMoves
```

Эта функция использует вложенные циклы (в строках 73 и 74) для проверки каждой координаты *x* и *y* (всех 64 координат), вызывая функцию isValidMove() в соответствующей клетке и проверяя, возвращает она значение False или список возможных ходов (в последнем случае ход является допустимым). Каждая допустимая координата *x* и *y* добавляется в список validMoves.

Вызов функции bool()

Возможно, вы заметили, что в строке 75 программа проверяет, возвращает ли функция isValidMove() значение False, несмотря на то, что она возвращает список. Чтобы понять, как это работает, вам нужно немного узнать о логических типах и функции bool().

Функция bool() похожа на функции int() и str(). Она возвращает логический тип переданного ей значения.

Большинство типов данных имеют одно значение, которое принимается False для этого типа данных. При этом любое другое значение считается True. Целое число 0, число с плавающей запятой 0,0, пустая строка, пустой список и пустой словарь принимают значение False, когда используются, например, в качестве условия для инструкции if или цикла. Все остальные значения — True. В интерактивной оболочке введите следующие команды:

```
>>> bool(0)
False
>>> bool(0.0)
False
```

```
>>> bool('')
False
>>> bool([])
False
>>> bool({})
False
>>> bool(1)
True
>>> bool('привет')
True
>>> bool([1, 2, 3, 4, 5])
True
>>> bool({'спам':'сыр', 'робик':'бобик'})
True
```

Условия автоматически интерпретируются как логические значения. Вот почему условие в строке 75 работает правильно. Вызов функции `isValidMove()` возвращает или логическое значение `False`, или непустой список.

Если вы представите, что все условие помещается внутри вызова функции `bool()`, тогда условие `False` в строке 75 становится `bool(False)` (что, разумеется, равно `False`). А условие непустого списка, переданного функции `bool()` в качестве параметра, вернет `True`.

Получение игрового счета

Функция `getScoreOfBoard()` использует вложенные циклы `for`, чтобы проверить все 64 позиции на поле и выяснить, какие на них есть фишки игрока (если есть).

```
79. def getScoreOfBoard(board):
80.     # Определить количество очков, подсчитав фишки. Вернуть словарь с ключами 'X' и 'O'.
81.     xscore = 0
82.     oscore = 0
83.     for x in range(WIDTH):
84.         for y in range(HEIGHT):
85.             if board[x][y] == 'X':
86.                 xscore += 1
87.             if board[x][y] == 'O':
88.                 oscore += 1
89.     return {'X':xscore, 'O':oscore}
```

В строке 86 код увеличивает значение переменной `xscore` для каждой фишки `X`. В строке 88 увеличивается `oscore` для каждой фишки `O`. Затем функция возвращает `xscore` и `oscore` в словаре.

Получение сделанного игроком выбора фишки

Функция `enterPlayerTile()` спрашивает игрока, какой фишкой он желает играть, *X* или *O*.

```
91. def enterPlayerTile():
92.     # Позволить игроку ввести выбранную фишку.
93. # Возвращает список с фишкой игрока в качестве первого элемента и фишкой компьютера в качестве второго.
94.     tile = ''
95.     while not (tile == 'X' or tile == 'O'):
96.         print('Вы играете за X или O?')
97.         tile = input().upper()
98.
99.     # Первый элемент в списке — фишка игрока, второй элемент — фишка компьютера.
100.     if tile == 'X':
101.         return ['X', 'O']
102.     else:
103.         return ['O', 'X']
```

Цикл `for` будет продолжать выполняться, пока игрок не введет прописную или строчную букву `X` или `O`. Затем функция `enterPlayerTile()` возвращает список из двух элементов, в котором выбранная игроком фишка является первым элементом, а фишка компьютера — вторым. Позже код в строке 241 вызывает функцию `enterPlayerTile()` и использует множественное присваивание, чтобы поместить эти два возвращенных элемента в две переменные.

Определение первого игрока

Функция `whoGoesFirst()` случайным образом определяет, кто делает ход первым, и возвращает либо строку `'Компьютер'`, либо строку `'Человек'`.

```
105. def whoGoesFirst():
106.     # Случайно выбрать, кто ходит первым.
107.     if random.randint(0, 1) == 0:
108.         return 'Компьютер'
109.     else:
110.         return 'Человек'
```

Помещение фишки на поле

Функция makeMove() вызывается, когда игрок хочет выставить фишку на поле и перевернуть другие фишки в соответствии с правилами игры.

```
112. def makeMove(board, tile, xstart, ystart):
113.     # Поместить фишку на игровое поле в позицию xstart,    ystart и перевернуть какую-либо
фишку противника.
114.     # Вернуть False, если это недопустимый ход; вернуть True, если допустимый.
115.     tilesToFlip = isValidMove(board, tile, xstart, ystart)
```

Эта функция на месте изменяет передаваемую ей структуру данных поля board. Изменения, внесенные в переменную board (поскольку это ссылка на список), будут внесены глобально.

Большая часть работы выполняется функцией isValidMove() в строке 115. Эта функция возвращает список координат *x* и *y* (в двухэлементном списке) фишек, которые нужно перевернуть. Помните, что если аргументы xstart и ystart укажут на недопустимый ход, функция isValidMove() вернет логическое значение False — это проверяется в строке 117.

```
117.     if tilesToFlip == False:
118.         return False
119.
120.     board[xstart][ystart] = tile
121.     for x, y in tilesToFlip:
122.         board[x][y] = tile
123.     return True
```

Если значение, возвращаемое функцией isValidMove() (теперь хранящееся в tilesToFlip) — False, то функция makeMove() в строке 118 также вернет False.

В противном случае функция isValidMove() возвращает список клеток на поле, в которых нужно перевернуть фишки (строка 'X' или 'O' в tile). Код в строке 20 назначает клетку, в которую переместился игрок. Цикл for в строке 121 назначает все фишки, которые находятся в списке tilesToFlip.

Создание копии структуры данных игрового поля

Функция getBoardCopy() отличается от getNewBoard(). Функция getNewBoard() создает структуру данных пустого игрового поля, которая содержит лишь пустые клетки и четыре стартовые фишки. Функция getBoardCopy() создает

структуру данных пустого поля, но затем копирует все позиции в параметре board с помощью вложенного цикла. ИИ использует функцию getBoardCopy(), чтобы была возможность вносить изменения в копию игрового поля, не изменяя исходное игровое поле. Эта методика также использовалась в игре «Крестики-нолики» в главе 10.

```
125. def getBoardCopy(board):
126.     # Сделать копию списка board и вернуть ее.
127.     boardCopy = getNewBoard()
128.
129.     for x in range(WIDTH):
130.         for y in range(HEIGHT):
131.             boardCopy[x][y] = board[x][y]
132.
133.     return boardCopy
```

Вызов функции getNewBoard() назначает boardCopy структурой данных нового игрового поля. Затем два вложенных цикла for копируют каждую из 64 фишек из board в копию структуры данных поля boardCopy.

Определение того, находится ли клетка в углу

Функция isOnCorner() возвращает значение True, если координаты соответствуют угловой клетке с координатами (0, 0), (7, 0), (0, 7) или (7, 7).

```
135. def isOnCorner(x, y):
136.     # Вернуть True, если указанная позиция находится в одном из четырех углов.
137.     return (x == 0 or x == WIDTH - 1) and (y == 0 or y == HEIGHT - 1)
```

В противном случае функция isOnCorner() возвращает False. Позже мы будем использовать эту функцию для программирования ИИ.

Получение хода игрока

Функция getPlayerMove() вызывается, чтобы позволить игроку ввести координаты следующего хода (и проверить, допустим ли он). Игрок также может ввести слово подсказка, чтобы включить (если выключен) или выключить (если включен) режим подсказок. Наконец, игрок может ввести слово выход для выхода из игры.

```
139. def getPlayerMove(board, playerTile):
140.     # Позволить игроку ввести свой ход.
141.     # Вернуть ход в виде [x, y] (или вернуть строки 'подсказка' или 'выход').
142.     DIGITS1TO8 = '1 2 3 4 5 6 7 8'.split()
```

Константа DIGITS1TO8 содержит список [1, 2, 3, 4, 5, 6, 7, 8]. Функция getPlayerMove() использует DIGITS1TO8 пару раз, а имя константы проще прочитать, чем полное значение списка. Нельзя использовать метод isdigit(), потому что это позволит вводить 0 и 9 — значения, не являющиеся допустимыми координатами на поле размером 8 × 8.

Цикл while продолжает выполнение до тех пор, пока игрок не введет допустимый ход.

```
143.     while True:
144.         print('Укажите ход, текст "выход" для завершения игры или "подсказка" для вывода
подсказки.')
145.         move = input().lower()
146.         if move == 'выход' or move == 'подсказка':
147.             return move
```

Код в строке 146 проверяет, хочет ли игрок выйти или переключить режим подсказок, а код в строке 147 возвращает соответственно строку 'выход' или 'подсказка'. Метод lower() вызывается в строке, возвращаемой функцией input(), поэтому игрок может ввести ПОДСКАЗКА или Выход, и код все равно «поймет» команду.

Код, который вызвал функцию getPlayerMove(), будет решать, что делать, если игрок хочет выйти или переключить режим подсказок. Если игрок вводит координаты для хода, инструкция if в строке 149 проверяет, допустим ли этот ход.

```
149.         if len(move) == 2 and move[0] in DIGITS1TO8 and move[1] in DIGITS1TO8:
150.             x = int(move[0]) - 1
151.             y = int(move[1]) - 1
152.             if isValidMove(board, playerTile, x, y) == False:
153.                 continue
154.             else:
155.                 break
```

Игра ожидает, что игрок введет координаты *x* и *y* своего хода в виде двух чисел без какого-либо символа или пробела между ними. Сначала код в строке 149 проверяет, чтобы размер строки, которую ввел игрок, был равен `2`. После этого он также проверяет, являются ли и `move[0]` (первый символ в строке), и `move[1]` (второй символ в строке) строками, доступными в списке `DIGITS1TO8`.

Помните, что структуры данных игрового поля имеют индексы от 0 до 7, а не от 1 до 8. Когда поле отображается в `drawBoard()`, код выводит от 1 до 8, потому что непрограммисты привыкли, что числа начинаются с 1, а не с 0. Поэтому для преобразования строк из элементов `move[0]` и `move[1]` в целые числа, строки 150 и 151 вычитают 1 из значений переменных `x` и `y`.

Даже если игрок ввел допустимый ход, код должен проверить, разрешен ли этот ход правилами игры «Реверси». Это осуществляется с помощью функции `isValidMove()`, которой передается структура данных игрового поля, фишка игрока и координаты *x* и *y* хода.

Если `isValidMove()` возвращает значение `False`, в строке 153 выполняется инструкция `continue`. Затем интерпретатор возвращается к началу цикла `while` и снова запрашивает у игрока допустимый ход. В противном случае игрок действительно ввел допустимый ход, и интерпретатор должен выйти из цикла `while`.

Если условие при инструкции `if` в строке 149 было ложным, значит, игрок не ввел допустимый ход. Строки 157 и 158 сообщают игроку, как правильно вводить ходы.

```
156.        else:
157.            print('Это недопустимый ход. Введите номер столбца (1-8) и номер ряда (1-8).')
158.            print('К примеру, значение 81 перемещает в верхний правый угол.')
```

После этого интерпретатор возвращается к инструкции `while` в строке 143, поскольку строка 158 — последняя строка не только в блоке `else`, но и в блоке `while`. Цикл `while` будет продолжать выполняться до тех пор, пока игрок не введет допустимый ход. Если игрок вводит координаты *x* и *y*, будет выполнен код в строке 160.

```
160.    return [x, y]
```

Наконец, если код в строке 160 выполняется, функция `getPlayerMove()` возвращает список из двух элементов с координатами *x* и *y* допустимого хода игрока.

Получение хода компьютера

С помощью функции getComputerMove() мы реализуем алгоритм ИИ.

```
162. def getComputerMove(board, computerTile):
163.     # Учитывая данное игровое поле и данную фишку компьютера, определить,
164.     # куда сделать ход, и вернуть этот ход в виде списка [x, y].
165.     possibleMoves = getValidMoves(board, computerTile)
```

Обычно вы используете результаты, полученные от функции getValidMoves(), для режима подсказок, который выведет . на поле, чтобы показать игроку все его возможные ходы. Но если функция getValidMoves() вызывается с фишкой ИИ компьютера (параметр computerTile), она также найдет все возможные ходы, которые может сделать *компьютер*. ИИ выберет лучший ход из этого списка.

Сначала функция random.shuffle() сделает порядок ходов в списке possibleMoves случайным.

```
166.     random.shuffle(possibleMoves) # Сделать случайным порядок ходов
```

Нужно перетасовать список possibleMoves, поскольку это сделает ИИ менее предсказуемым; в противном случае игрок мог бы просто запомнить ходы, необходимые для победы, ведь ответы компьютера всегда были бы одинаковыми. Давайте рассмотрим алгоритм.

Разработка стратегии с угловыми ходами

Угловые ходы — это неплохая идея для игры «Реверси», поскольку фишка в углу доски не может быть перевернута. Код в строке 169 перебирает каждый ход в списке possibleMoves. Если какой-либо из них является угловым, программа вернет эту клетку для хода компьютера.

```
168.     # Всегда делать ход в угол, если это возможно.
169.     for x, y in possibleMoves:
170.         if isOnCorner(x, y):
171.             return [x, y]
```

Так как possibleMoves — это список двухэлементных списков, мы будем использовать множественное присваивание в цикле for для назначения значе-

ний x и y. Если список possibleMoves содержит несколько угловых ходов, всегда используется первый из них. Но так как в строке 166 содержимое списка possibleMoves было перетасовано, первый угловой ход в списке определяется случайным образом.

Получения списка самых результативных ходов

Если угловые ходы отсутствуют, программа переберет весь список возможных ходов и выяснит, какой из них приводит к получению наибольшего количества очков.

Тогда переменной bestMove присваивается ход с наибольшим количеством очков, найденный до сих пор кодом, а переменной bestScore присваивается это количество очков. Так происходит до тех пор, пока не будет найден лучший ход.

```
173.    # Найти ход с наибольшим возможным количеством очков.
174.    bestScore = -1
175.    for x, y in possibleMoves:
176.        boardCopy = getBoardCopy(board)
177.        makeMove(boardCopy, computerTile, x, y)
178.        score = getScoreOfBoard(boardCopy)[computerTile]
179.        if score > bestScore:
180.            bestMove = [x, y]
181.            bestScore = score
182.    return bestMove
```

Сначала код в строке 174 присваивает переменной bestScore значение -1, чтобы первый ход, проверяемый кодом, был присвоен bestMove как первый лучший ход. Этим можно гарантировать, что bestMove будет присвоен один из ходов из possibleMoves, когда bestMove будет возвращен.

В строке 175 цикл for присваивает x и y каждый ход в possibleMoves. Перед моделированием хода код в строке 176 создает копию структуры данных игрового поля, вызывая функцию getBoardCopy(). Вам понадобится копия, которую вы можете изменять, не влияя на настоящую структуру данных поля, хранящуюся в переменной board.

Затем код в строке 177 вызывает функцию makeMove(), передавая копию поля (хранимую в boardCopy) вместо настоящего поля. Это моделирование ситуации на реальном игровом поле, будь сделан такой ход. Функция makeMove() будет заниматься выставлением фишки компьютера и переворачиванием фишек игрока на копии игрового поля.

Код в строке 178 вызывает функцию getScoreOfBoard() с копией поля, которая возвращает словарь, в котором ключами являются 'X' и 'O', а значения — это очки. Когда код в цикле находит ход, у которого больше очков, чем содержится в bestScore, код в строках с 179 по 181 сохраняют этот ход и это количество очков в качестве новых значений bestMove и bestScore. После того как цикл полностью перебрал список possibleMoves, возвращается bestMove.

К примеру, getScoreOfBoard() возвращает словарь {'X': 22, 'O': 8}, а значение computerTile — 'X'.

Тогда getScoreOfBoard(boardCopy)[computerTile] примет значение {'X': 22, 'O': 8}['X'], что затем примет значение 22. Если 22 больше, чем bestScore, тогда bestScore устанавливается равной 22, а bestMove устанавливается равной текущим значениям x и y.

К моменту завершения цикла for вы можете быть уверены, что bestScore содержит наивысшее значение очков, которое можно получить ходом, и этот ход хранится в bestMove.

Несмотря на то что код всегда выбирает первый из ходов в списке, выбор оказывается случайным, поскольку список был перетасован в строке 166. Это гарантирует, что ИИ не будет предсказуемым в случае наличия больше одного лучшего хода.

Вывод игрового счета на экран

Функция showPoints() вызывает функцию getScoreOfBoard(), а затем выводит результаты игры человека и компьютера.

```
184. def printScore(board, playerTile, computerTile):
185.     scores = getScoreOfBoard(board)
186.     print('Ваш счет: %s. Счет компьютера: %s.' % (scores[playerTile], scores[computerTile]))
```

Помните, что функция getScoreOfBoard() возвращает словарь с ключами 'X' и 'O' и значениями с очками игроков *X* и *O*.

Это все функции, которые используются в игре «Реверси». Код внутри функции playGame(), собственно, воплощает игру в жизнь, вызывая эти функции по мере необходимости.

Начало игры

Функция playGame() использует вызовы предварительно написанных нами функций для запуска игры.

```
188. def playGame(playerTile, computerTile):
189.     showHints = False
190.     turn = whoGoesFirst()
191.     print(turn + ' ходит первым.')
192.
193.     # Очистить игровое поле и выставить стартовые фишки.
194.     board = getNewBoard()
195.     board[3][3] = 'X'
196.     board[3][4] = 'O'
197.     board[4][3] = 'O'
198.     board[4][4] = 'X'
```

Функции playGame() передаются строки 'X' или 'O' в виде playerTile и computerTile. Строка 190 определяет, чей первый ход. Переменная turn содержит строку 'Компьютер' или 'Человек' для отслеживания того, чья очередь ходить. Код в строке 194 создает структуру данных пустого поля, в то время как строки с 195 по 198 устанавливают на поле начальные четыре фишки. Теперь можно начинать играть.

Проверка на попадание в тупик

Прежде чем получить ход игрока или компьютера, нужно проверить, могут ли они вообще сделать ход. Если нет, то игра заходит в тупик и должна окончиться. (Если допустимых ходов нет только у одной стороны, очередь переходит к другому игроку.)

```
200.     while True:
201.         playerValidMoves = getValidMoves(board, playerTile)
202.         computerValidMoves = getValidMoves(board, computerTile)
203.
204.         if playerValidMoves == [] and computerValidMoves == []:
205.             return board # Ходов нет ни у кого, так что окончить игру.
```

Код в строке 200 начинает основной цикл для управления очередностью ходов игрока и компьютера. Пока этот цикл продолжает выполняться, игра будет работать. Но прежде чем начать выполнять эти очереди, строки 201 и 202 получают списки допустимых ходов и проверяют, может ли каждая из сторон сделать ход. Если оба этих списка пустые, значит, ни один из игроков не может сделать ход. Строка 205 выходит из функции playGame(), возвращая итоговое поле, завершая игру.

Выполнение хода игроком

Если игра не попадает в тупик, программа определяет, принадлежит ли ход игроку, проверяя, принимает ли переменная `turn` значение строки `'Человек'`.

```
207.        elif turn == 'Человек': # Ход человека
208.            if playerValidMoves != []:
209.                if showHints:
210.                    validMovesBoard = getBoardWithValidMoves(board, playerTile)
211.                    drawBoard(validMovesBoard)
212.                else:
213.                    drawBoard(board)
214.                printScore(board, playerTile, computerTile)
```

В строке 207 начинается блок `elif`, содержащий код, который выполняется, если это очередь игрока. (Блок `elif`, который начинается в строке 227, содержит код для очереди компьютера.)

Весь этот код будет работать только в том случае, если у игрока есть допустимый ход, что определяет строка 208, проверяя, не пуст ли список `playerValidMoves`. Мы отображаем поле на экране, вызывая функцию `drawBoard()` в строке 211 или 213.

Если включен режим подсказок (то есть `showHints` принимает значение `True`), тогда структура данных поля должна отобразить `.` на каждой допустимой для хода игрока клетке. Это осуществляется с помощью функции `getBoardWithValidMoves()`. Ей передается структура данных игрового поля, а возвращает она копию, которая содержит еще и точки (`.`). Код в строке 211 передает это поле функции `drawBoard()`.

Если режим подсказок выключен, тогда вместо этого строка 213 передает `board` в качестве параметра `drawBoard()`.

После вывода игрового поля также необходимо вывести текущие очки. Делается это путем вызова функции `printScore()` в строке 214.

Затем игроку необходимо ввести свой ход. Этим занимается функция `getPlayerMove()` — возвращаемое ей значение представляет собой двухэлементный список координат *x* и *y* хода игрока.

```
216.                move = getPlayerMove(board, playerTile)
```

К моменту определения функции `getPlayerMove()` мы уже убедились в допустимости хода игрока.

Функция getPlayerMove() может вернуть строки 'выход' или 'подсказка' вместо хода. Строки с 217 по 222 работают с такими случаями:

```
217.                if move == 'выход':
218.                    print('Благодарим за игру!')
219.                    sys.exit() # Завершить работу программы.
220.                elif move == 'подсказка':
221.                    showHints = not showHints
222.                    continue
223.                else:
224.                    makeMove(board, playerTile, move[0], move[1])
225.            turn = 'Компьютер'
```

Если игрок вместо хода ввел слово выход, функция getPlayerMove() вернет строку 'выход'. В этом случае код в строке 219 вызывает функцию sys.exit() для завершения работы программы.

Если игрок ввел слово подсказка, функция getPlayerMove() вернет строку 'подсказка'. В этом случае нужно включить режим подсказок (если он выключен) либо выключить, если он включен).

Инструкция присваивания showHints = not showHints в строке 221 работает с обоими этими случаями, потому что not False принимает значение True, а not True — то же, что False. Затем инструкция continue переносит выполнение в начало цикла (переменная turn не изменила значение, так что это по-прежнему будет очередь игрока). В противном случае, если игрок не вышел и не переключил режим подсказок, код в строке 224 вызывает функцию makeMove(), чтобы осуществить ход игрока.

Наконец, код в строке 225 присваивает переменной turn значение 'Компьютер'. Поток выполнения пропускает блок else и достигает конца блока while, поэтому интерпретатор возвращается к инструкции while в строке 200. На этот раз, однако, будет очередь компьютера.

Выполнение хода компьютером

Если переменная turn содержит строку 'Компьютер', тогда будет выполняться код, отвечающий за ход компьютера. Он аналогичен коду для очереди игрока, но имеет отличия.

```
227.        elif turn == 'Компьютер': # Ход компьютера
228.            if computerValidMoves != []:
```

```
229.                drawBoard(board)
230.                printScore(board, playerTile, computerTile)
231.
232.                input('Нажмите клавишу Enter для просмотра хода компьютера.')
233.                move = getComputerMove(board, computerTile)
234.                makeMove(board, computerTile, move[0], move[1])
```

После отображения игрового поля с помощью функции `drawBoard()` программа также выводит текущие очки, вызывая функцию `showPoints()` в строке 230.

Код в строке 232 вызывает функцию `input()`, чтобы приостановить сценарий и дать игроку взглянуть на поле. Это очень похоже на то, как функция `input()` использовалась для приостановки программы-шутника в главе 4. Вместо того чтобы вызывать функцию `print()` для вывода строки на экран перед вызовом функции `input()`, вы можете сделать то же самое, передав строку для вывода функции `input()`.

После того как игрок посмотрел на поле и нажал клавишу **Enter**, код в строке 233 вызывает функцию `getComputerMove()`, чтобы получить координаты *x* и *y* следующего хода компьютера. Эти координаты сохраняются в переменных `x` и `y` с помощью множественного присваивания.

Наконец, значения переменных `x` и `y` вместе со структурой данных игрового поля и фишкой компьютера передаются функции `makeMove()`. Она помещает фишку компьютера на игровое поле в `board`, отражая ход компьютера. Вызов функции `getComputerMove()` в строке 233 позволяет получить ход компьютера (и сохраняет его в переменных `x` и `y`). Функция `makeMove()` в строке 234 делает ход на игровом поле.

Затем код в строке 235 присваивает переменной `turn` строковое значение `'Человек'`.

```
235.            turn = 'Человек'
```

После строки 235 в блоке `while` больше нет кода, поэтому интерпретатор возвращается к инструкции `while` в строке 200.

Игровой цикл

Это все функции, которые используются в игре «Реверси». Начиная со строки 239, основная часть программы запускает игру, вызывая функцию `playGame()`, но она также отображает окончательный результат и спрашивает игрока, желает ли он сыграть снова.

```
239. print('Приветствуем в игре "Реверси"!')
240.
241. playerTile, computerTile = enterPlayerTile()
```

Программа начинается с приветствия игрока в строке 239 и предложения выбрать *X* или *O*. В строке 241 используется множественное присваивание, чтобы установить переменные `playerTile` и `computerTile` равными двум значениям, которые возвращаются `enterPlayerTile()`.

Цикл `while` в строке 243 проводит каждую игру.

```
243. while True:
244.     finalBoard = playGame(playerTile, computerTile)
245.
246.     # Отобразить итоговый счет.
247.     drawBoard(finalBoard)
248.     scores = getScoreOfBoard(finalBoard)
249.     print('X набрал %s очков. O набрал %s очков.' % (scores['X'], scores['O']))
250.     if scores[playerTile] > scores[computerTile]:
251.         print('Вы победили компьютер, обогнав его на %s очков! Поздравления!' %
(scores[playerTile] - scores[computerTile]))
252.     elif scores[playerTile] < scores[computerTile]:
253.         print('Вы проиграли. Компьютер победил вас, обогнав на %s очков.' %
(scores[computerTile] - scores[playerTile]))
254.     else:
255.         print('Ничья!')
```

Он начинается с вызова `playGame()`. Этот вызов не вернется, пока игра не будет окончена. Структура данных поля, возвращаемая функцией `playGame()`, будет передана функции `getScoreOfBoard()` для подсчета фишек *X* и *O* и определения итогового результата. Код в строке 249 отображает этот результат.

Если фишек игрока на доске больше, чем фишек компьютера, код в строке 251 поздравляет игрока с победой. Если победил компьютер, код в строке 253 сообщает игроку, что он проиграл. В противном случае код в строке 255 сообщает игроку, что игра завершилась вничью.

Предложение сыграть снова

После окончания игры появляется запрос, хочет ли пользователь сыграть снова.

```
257.     print('Хотите сыграть еще раз? (да или нет)')
258.     if not input().lower().startswith('д'):
259.         break
```

Если игрок не вводит ответ, начинающийся с буквы *д*, например, `да` или `ДА`, либо `Д`, тогда условие в строке 258 принимает значение `True`, а код в строке 259 прерывает цикл `while`, который начался в строке 243, что приводит к завершению игры. В противном случае этот цикл `while` запускается как обычно, и снова вызывается функция `playGame()`, чтобы начать следующую игру.

Заключение

Искусственный интеллект игры «Реверси» может показаться практически непобедимым. Это не потому, что компьютер умнее, чем мы, просто он намного быстрее! Стратегия, которой он следует, проста: если можете, делайте ход в угол, а в противном случае делайте ход, который перевернет больше всего фишек. Человек мог бы это сделать, но на выяснение того, сколько фишек будет перевернуто каждым допустимым ходом, ушло бы много времени. Для компьютера же такие расчеты не представляют труда.

Эта игра похожа на программу «Охотник за сокровищами», потому что использует сетку для игрового поля. Она также напоминает игру «Крестики-нолики», потому что здесь есть ИИ, планирующий лучший для компьютера ход. В этой главе вы изучили новые понятия: пустые списки, пустые строки, а также целое число `0`, принимающие значение `False` в контексте условия. Не считая этого, в данной игре использовались только уже знакомые вам методы программирования!

В главе 16 вы узнаете, как заставить искусственные интеллекты играть в компьютерные игры друг против друга.

16

ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ ИГРЫ «РЕВЕРСИ»

Алгоритм ИИ игры «Реверси» из главы 15 прост, но он побеждает автора почти в каждой игре. Поскольку компьютер способен быстро обрабатывать инструкции, он может легко проверить каждую возможную позицию на игровом поле и выбрать ход, который принесет больше всего очков. Если бы автор пытался отыскать лучший ход таким способом, это отняло бы уйму времени.

Программа «Реверси» содержала две функции, `getPlayerMove()` и `getComputerMove()`, которые возвращали выбранный ход как двухэлементный список в формате `[x, y]`. У обеих этих функций также были одинаковые параметры, структура данных игрового поля и один тип фишки, но возвращаемые ходы поступали из разных источников — либо от игрока, либо от алгоритма игры.

Что произойдет, если мы заменим вызов `getPlayerMove()` вызовом `getComputerMove()`? Тогда игроку вообще не нужно будет вводить ход; этот выбор будет сделан за него. Компьютер будет играть против самого себя!

В этой главе мы создадим три новые программы, в которых компьютер играет против себя; каждая из этих программ будет основана на игре «Реверси» из главы 15:

- **Модель 1**: файл *AISim1.py* представляет собой измененную версию файла *reversegam.py*.
- **Модель 2**: файл *AISim2.py* представляет собой измененную версию файла *AISim1.py*.
- **Модель 3**: файл *AISim3.py* представляет собой измененную версию файла *AISim2.py*.

Небольшие изменения каждой из перечисленных программ покажут вам, как превратить игру «игрок против компьютера» в модель «компьютер против компьютера». Итоговая программа, *AISim3.py*, большую часть своего кода заимствует из файла *reverseegam.py*, но служит совершенно другой цели. Моделирование не дает нам играть в «Реверси», но позволяет узнать больше о самой игре.

Вы можете либо ввести эти изменения самостоятельно, либо загрузить готовые файлы с веб-сайта https://eksmo.ru/files/python_games_sweigart.zip.

В ЭТОЙ ГЛАВЕ РАССМАТРИВАЮТСЯ СЛЕДУЮЩИЕ ТЕМЫ:

- Модели
- Процентные соотношения
- Целочисленное деление
- Функция `round()`
- Игры типа «компьютер против компьютера»

Компьютер против компьютера

Чтобы компьютер смог играть против себя, мы внесем в программу *reversegam.py* несколько простых изменений. Функции `getPlayerMove()` и `getComputerMove()` принимают структуру данных поля и фишку игрока, а затем возвращают ход, который следует сделать. Вот почему функция `getComputerMove()` может заменить `getPlayerMove()`, и программа по-прежнему будет работать. В программе *AISim1.py* функция `getComputerMove()` вызывается и для игрока `X`, и для игрока `O`.

Мы также уберем вывод игрового поля с делающимися на нем ходами. Поскольку человек не может считывать выводы полей так же быстро, как компьютер делает ходы, нецелесообразно выводить каждый ход — так что мы просто выводим итоговое поле.

Эти изменения минимальны, поэтому программа все равно будет выдавать `Человек ходит первым.` и другие сообщения, несмотря на то, что компьютер играет за обе стороны.

Пример запуска модели 1

Вот что видит пользователь при запуске программы *AISim1.py*. Текст, введенный игроком, выделен полужирным шрифтом.

```
Приветствуем в игре "Реверси"!
Компьютер ходит первым.
```

```
  12345678
 +--------+
1|XOOOOOOO|1
2|XXOOXOOO|2
3|OOXXOOOO|3
4|OOXOOOOO|4
5|OOOOXOXO|5
6|OOOOXOOO|6
7|OOXXXXXO|7
8|OXXOOOOO|8
 +--------+
  12345678
X набрал 17 очков. O набрал 47 очков.
Вы проиграли. Компьютер победил вас, обогнав на 30 очков.
Хотите сыграть еще раз? (да или нет)
нет
```

Исходный код модели 1

Сохраните старый файл *reverseegam.py* под именем *AISim1.py* следующим образом:

1. Выберите команду меню **File** ⇒ **Save as** (Файл ⇒ Сохранить как).
2. Сохраните этот файл под именем *AISim1.py*, чтобы вы могли вносить изменения, не затрагивая программу *reverseegam.py*. (На данном этапе *reverseegam.py* и *AISim1.py* все еще содержат один и тот же код.)
3. Внесите изменения в *AISim1.py* и сохраните этот файл, чтобы зафиксировать любые изменения. (*AISim1.py* будет содержать новые изменения, а *reverseegam.py* будет содержать исходный, неизмененный код.)

Так вы создадите копию исходного кода «Реверси» в виде нового файла, в который вы можете вносить изменения, сохранив оригинальную игру (вы можете захотеть снова в нее сыграть, чтобы протестировать ее). Например, замените код в строке 216 в файле *AISim1.py* следующим (изменение выделено полужирным шрифтом):

```
216.                move = getComputerMove(board, playerTile)
```

Теперь запустите программу. Обратите внимание, что игра по-прежнему спрашивает, что вы хотите выбрать — *X* или *O*, но не попросит ввести координаты ходов. Когда вы заменяете функцию `getPlayerMove()` функцией `getComputerMove()`, вы больше не вызываете код, который принимает инфор-

мацию, вводимую игроком. Игрок по-прежнему нажимает клавишу **Enter** после исходных ходов компьютера (это требуется из-за кода `input('Нажмите клавишу Enter для просмотра хода компьютера.')` в строке 232), но программа играет самостоятельно!

Внесем в файл *AISim1.py* еще кое-какие изменения. Измените указанные ниже строки кода, выделенные полужирным шрифтом. Изменения начинаются со строки 209. Большинство этих изменений просто «отключат» строки кода, *закомментировав* их, чтобы те не выполнялись.

Если при выполнении программы возникают ошибки, сравните код, который вы набрали, с оригинальным кодом с помощью онлайн-инструмента на сайте **inventwithpython.com/diff/**.

AISim1.py

```
207.         elif turn == 'Человек': # Player's turn
208.             if playerValidMoves != []:
209.                 #if showHints:
210.                 #    validMovesBoard = getBoardWithValidMoves(board, playerTile)
211.                 #    drawBoard(validMovesBoard)
212.                 #else:
213.                     #drawBoard(board)
214.                 #printScore(board, playerTile, computerTile)
215.
216.                 move = getComputerMove(board, playerTile)
217.                 #if move == 'выход':
218.                 #    print('Благодарим за игру!')
219.                 #    sys.exit() # Завершить работу программы.
220.                 #elif move == 'подсказка':
221.                 #    showHints = not showHints
222.                 #    continue
223.                 #else:
224.                 makeMove(board, playerTile, move[0], move[1])
225.             turn = 'Компьютер'
226.
227.         elif turn == 'Компьютер': # Ход компьютера
228.             if computerValidMoves != []:
229.                 #drawBoard(board)
230.                 #printScore(board, playerTile, computerTile)
```

```
231.
232.                #input('Нажмите клавишу Enter для просмотра хода компьютера.')
233.                move = getComputerMove(board, computerTile)
234.                makeMove(board, computerTile, move[0], move[1])
235.            turn = 'Человек'
236.
237.
238.
239. print('Приветствуем в игре "Реверси"!')
240.
241. playerTile, computerTile = ['X', 'O'] #enterPlayerTile()
```

Удаление приглашений для игрока и добавление игрока-компьютера

Как видно из кода, программа *AISim1.py* в основном тождественна исходной программе «Реверси», за исключением того, что вызов `getPlayerMove()` был заменен вызовом `getComputerMove()`. Также были внесены некоторые изменения в текст вывода на экране, чтобы за ходом игры было проще следить. После старта программы на всю игру уходит менее секунды!

И снова большинство изменений заключается в комментировании некоторых строк кода. Поскольку компьютер играет против самого себя, программе больше не нужно запускать код для получения ходов игрока или отображения состояния игрового поля. Все это пропущено, поэтому поле отображается только в самом конце игры. Код закомментирован, а не удален по той причине, что если нам будет нужно повторно использовать код позже, его будет проще восстановить, убрав символы `#`.

Я закомментировал строки кода с 209 по 214, поскольку больше не нужно выводить поле для игрока, уже не принимающего участие в игре. То же я проделал со строками с 217 по 223, ведь не нужно проверять, вводит ли игрок слово `выход` или переключает режим подсказок. Но нужно убрать четыре пробела перед строкой 224, так как она была частью блока `else`, который я только что отключил. Строки 229–232 также выводят игровое поле, поэтому я закомментировал и эти строки.

Единственный новый код — в строках 216 и 241. В строке 216 мы просто заменяем вызов `getPlayerMove()` вызовом функции `getComputerMove()`, как говорилось ранее. В строке 241 вместо того, чтобы спрашивать игрока, желает он играть за *X* или *O*, мы просто присваиваем значение `'X'` переменной `playerTile` и `'O'` переменной `computerTile`. (Учитывая, что за обоих этих игроков будет играть компьютер, вы можете переименовать `playerTile` в `computerTile2` или `secondComputerTile`.) Теперь, когда у нас есть программа, в которой компьютер

играет против самого себя, можно продолжить изменять ее код, чтобы реализовать дополнительные интересные возможности.

Компьютер против компьютера — несколько партий

Если бы мы создали новый алгоритм, то могли бы противопоставить его ИИ, реализованному с помощью функции `getComputerMove()`, и посмотреть, какой из них лучше. Однако прежде чем мы сделаем это, нам нужен способ оценивать игроков. Мы не можем оценить, какой ИИ лучше, основываясь только на результатах одной партии, поэтому необходимо, чтобы ИИ сыграли друг против друга более одного раза. Для этого мы внесем некоторые изменения в исходный код. Выполните следующие шаги, чтобы создать программу *AISim2.py*:

1. Выберите команду меню **File** ⇒ **Save as** (Файл ⇒ Сохранить как).
2. Сохраните этот файл под именем *AISim2.py*, чтобы вы могли вносить изменения, не затрагивая программу *AISim1.py*. (На данном этапе *AISim1.py* и *AISim2.py* все еще содержат один и тот же код.)

Пример запуска модели 2

Вот что видит пользователь, когда запускает программу *AISim2.py*:

```
Приветствуем в игре "Реверси"!
#1: X набрал 38 очков. O набрал 26 очков.
#2: X набрал 26 очков. O набрал 38 очков.
#3: X набрал 25 очков. O набрал 39 очков.
#4: X набрал 33 очков. O набрал 31 очков.
#5: X набрал 23 очков. O набрал 41 очков.
#6: X набрал 33 очков. O набрал 31 очков.

--пропуск--

#249: X набрал 30 очков. O набрал 34 очков.
#250: X набрал 26 очков. O набрал 38 очков.
Количество выигрышей X: 115 (46.0%)
Количество выигрышей O: 129 (51.6%)
Количество ничьих:   6 (2.4%)
```

Поскольку алгоритмы предполагают случайность, ваш запуск не будет содержать точно такие же числа.

Исходный код модели 2

Измените код в *AISim2.py*, чтобы он соответствовал приведенному ниже коду. Убедитесь, что меняете код строка за строкой в порядке очереди. Если при выполнении программы возникают ошибки, сравните код, который вы набрали, с оригинальным кодом с помощью онлайн-инструмента на сайте **inventwithpython.com/diff/**.

AISim2.py

```
235.            turn = 'Человек'
236.
237. NUM_GAMES = 250
238. xWins = oWins = ties = 0
239. print('Приветствуем в игре "Реверси"!')
240.
241. playerTile, computerTile = ['X', 'O'] #enterPlayerTile()
242.
243. for i in range(NUM_GAMES): #while True:
244.     finalBoard = playGame(playerTile, computerTile)
245.
246.     # Отобразить итоговый счет.
247.     #drawBoard(finalBoard)
248.     scores = getScoreOfBoard(finalBoard)
249.     print('#%s: X набрал %s очков. O набрал %s очков.' % (i + 1, scores['X'], scores['O']))
250.     if scores[playerTile] > scores[computerTile]:
251.  xWins+=1#print('Вы победили компьютер, обогнав его на %s очков! Поздравления!' % (scores[playerTile] - scores[computerTile]))
252.     elif scores[playerTile] < scores[computerTile]:
253.   oWins += 1 #print('Вы проиграли. Компьютер победил вас, обогнав на %s очков.' % (scores[computerTile] - scores[playerTile]))
254.     else:
255.         ties += 1 #print('Ничья!')
256.
257.     #print('Хотите сыграть еще раз? (да или нет)')
258.     #if not input().lower().startswith('д'):
259.     #    break
260.
261. print('Количество выигрышей X: %s (%s%%)' % (xWins, round(xWins / NUM_GAMES * 100, 1)))
262. print('Количество выигрышей O: %s (%s%%)' % (oWins, round(oWins / NUM_GAMES * 100, 1)))
263. print('Количество ничьих:   %s (%s%%)' % (ties, round(ties / NUM_GAMES * 100, 1)))
```

Если у вас возникают проблемы с внесением изменений, можно загрузить готовый файл *AISim2.py* с веб-сайта https://eksmo.ru/files/python_games_sweigart.zip.

Отслеживание результатов партий

Основная информация, которую мы хотим получить в процессе моделирования, — это то, сколько побед одержал игрок *X*, сколько побед игрок *O*, и сколько ничьих состоялось за определенное количество партий. Все это можно отслеживать с помощью четырех переменных, которые создаются в строках 237 и 238.

```
237. NUM_GAMES = 250
238. xWins = oWins = ties = 0
```

Константа `NUM_GAMES` определяет, сколько партий будет сыграно. Были добавлены переменные `xWins`, `oWins` и `ties`, чтобы отслеживать победы *X*, победы *O* и ничьи. Вы можете связать воедино инструкции присваивания, чтобы присвоить переменной `ties` значение `0`, переменной `oWins` — значение `ties`, и переменной `xWins` — значение `oWins`. Эта инструкция присваивает всем трем переменным значение `0`.

Константа `NUM_GAMES` используется в цикле `for`, который заменяет игровой цикл в строке 243.

```
243. for i in range(NUM_GAMES): #while True:
```

Цикл `for` запускает игру столько раз, сколько указано в константе `NUM_GAMES`. Код заменяет цикл `while`, который выполнялся до тех пор, пока игрок не сообщал, что не хочет играть еще раз.

В строке 250 инструкция `if` сравнивает счет двух игроков, а строки с 251 по 255 в блоках `if-elif-else` инкрементируют значения переменных `xWins`, `oWins` и `ties` в конце каждой партии, прежде чем вернуться назад для начала новой партии.

```
250.     if scores[playerTile] > scores[computerTile]:
251.         xWins += 1 #print('Вы победили компьютер, обогнав его на %s очков! Поздравления!' %
(scores[playerTile] - scores[computerTile]))
252.     elif scores[playerTile] < scores[computerTile]:
253.         oWins += 1 #print('Вы проиграли. Компьютер победил вас, обогнав на %s очков.' %
(scores[computerTile] - scores[playerTile]))
254.     else:
255.         ties += 1 #print('Ничья!')
```

Мы убираем сообщения, первоначально выводящиеся внутри блока, поэтому теперь выводится лишь однострочный итог результатов каждой партии. Позже в коде мы будем использовать переменные `xWins`, `oWins` и `ties`, чтобы проанализировать, как компьютер выступил против самого себя.

Отключение вызовов функции print()

Я также закомментировал код в строках 247 и 257–259. Благодаря этому я убрал из программы бóльшую часть вызовов функций `print()` и `drawBoard()`. Нам не нужно наблюдать за каждой из партий, поскольку их очень много. Программа по-прежнему запускает каждую партию целиком с использованием написанного нами ИИ, но показывает только итоговый счет. После завершения всех партий программа показывает, сколько партий выиграла каждая сторона, а код в строках с 251 по 253 выводят некоторую информацию об проведенных партиях.

Вывод данных на экран замедляет работу компьютера, но теперь, когда вы удалили соответствующий код, компьютер может отыграть целую партию «Реверси» примерно за одну-две секунды. Каждый раз, когда программа выводила одну из тех строк с итоговым счетом, она обрабатывала всю игру (по отдельности проверяя около 50 или 60 ходов, чтобы выбрать тот, который приносит наибольшее количество очков). Теперь, когда компьютеру не нужно делать столько работы, он может работать намного быстрее.

Числа, которые программа выводит в конце, являются *статистическими данными*, которые используются для обобщения итоговых результатов партий. В этом случае мы показали итоговые очки игроков в каждой партии, а также процентное соотношение побед игроков и ничьих между ними.

Оценка искусственных интеллектов в процентном соотношении

Процентное соотношение — это часть от общего количества. Значение процентного соотношения может варьироваться в диапазоне от 0 до 100%. Представьте, что у вас есть пирог: 100% — это весь пирог, 50% — его половина, а 0% — пирога вообще нет.

Мы можем рассчитывать процентные соотношения с помощью деления. Чтобы получить процентное соотношение, разделите часть, которая у вас есть, на общее количество, а затем умножьте результат на 100. Например, если *X* выиграл 50 партий из 100, можно составить выражение `50 / 100`, равняющееся `0.5`. Умножьте это на `100`, чтобы получить процентное соотношение (в данном случае 50 процентов).

Если *X* выиграл 100 партий из 200, можно составить выражение `100 / 200`, что также равно `0.5`. Когда вы умножите `0.5` на `100`, чтобы узнать процентное

соотношение, получите 50 процентов. Выигрыш 100 партий из 200 — это то же процентное соотношение, что и выигрыш 50 партий из 100.

В строках с 261 по 263 мы используем процентные соотношения для вывода информации об итогах партий.

```
261. print('Количество выигрышей X: %s (%s%%)' % (xWins, round(xWins / NUM_GAMES * 100, 1)))
262. print('Количество выигрышей O: %s (%s%%)' % (oWins, round(oWins / NUM_GAMES * 100, 1)))
263. print('Количество ничьих:   %s (%s%%)' % (ties, round(ties / NUM_GAMES * 100, 1)))
```

Каждая инструкция `print()` имеет метку, которая сообщает пользователю, относятся выводимые данные к победам *X*, победам *O* или ничьим. Я применил метод интерполяции строк для вставки количества выигранных партий и ничьих, а также вычисленного процентного соотношения выигрышей и ничьих в общем количестве партий. Но, как видите, здесь не просто делятся значения переменных `xWins`, `oWins` или `ties` на общее количество партий и умножаются на `100`. Это потому что для каждого процентного соотношения нужно вывести только один дробный символ после запятой, что нельзя осуществить путем обычного деления.

Деление приводит к получению числа с плавающей запятой

Когда вы используете оператор деления (`/`), выражение всегда будет вычисляться в число с плавающей запятой. Например, выражение `10 / 2` вычисляется как число с плавающей запятой `5.0`, а не целочисленное значение `5`.

Важно помнить об этом, поскольку прибавление целого числа к числу с плавающей запятой с помощью оператора сложения `+` также всегда будет приводить к получению числа с плавающей запятой. Например, выражение `3 + 4.0` равно числу с плавающей запятой `7.0`, а не целому числу `7`.

Введите следующий код в интерактивной оболочке:

```
>>> spam = 100 / 4
>>> spam
25.0
>>> spam = spam + 20
>>> spam
45.0
```

В этом примере тип данных значения переменной `spam` всегда является числом с плавающей запятой. Вы можете передать значение с плавающей

запятой функции int(), которая возвращает целочисленную форму значения с плавающей запятой. Но эта функция всегда будет округлять значение в меньшую сторону. Например, каждое из выражений int(4.0), int(4.2) и int(4.9) принимает значение 4, но не 5. Но в программе *AISim2.py* нам нужно округлить каждую процентную долю до десятых. Поскольку для этого мы не можем просто выполнить деление, нужно воспользоваться функцией round().

Функция round()

Функция round() округляет число с плавающей запятой до ближайшего целого числа. В интерактивной оболочке введите следующие команды:

```
>>> round(10.0)
10
>>> round(10.2)
10
>>> round(8.7)
9
>>> round(3.4999)
3
>>> round(2.5422, 2)
2.54
```

Функция round() также имеет необязательный второй параметр, с помощью которого вы можете указать, до какого знака требуется округлить число. Благодаря этому округленное число может быть числом с плавающей запятой, а не целым. Например, выражение round(2.5422, 2) вычисляется в значение 2.54, а round(2.5422, 3) — в значение 2.542. В строках 261–263 программы *AISim2.py* мы используем эту функцию с параметром 1, чтобы округлить соотношение выигранных или сыгранных вничью партий до одного знака после запятой, что позволяет нам получить точные процентные соотношения.

Сравнение различных алгоритмов ИИ

Внеся лишь несколько изменений, мы можем заставить компьютер играть против себя сотни раз. Прямо сейчас каждый игрок побеждает примерно в половине партий, так как они оба используют один и тот же алгоритм выбора ходов. Но если мы добавим разные алгоритмы, то увидим, какой ИИ чаще будет выигрывать чаще.

Давайте добавим функции с новыми алгоритмами. Но сначала в окне с файлом *AISim2.py* выберите команду меню **File** ⇒ **Save as** (Файл ⇒ Сохранить как), чтобы сохранить сценарий под именем *AISim3.py*.

Мы переименуем функцию `getComputerMove()` в `getCornerBestMove()`, так как этот алгоритм сначала пытается осуществлять угловые ходы, а затем выбирает ход, который переворачивает больше всего фишек. Мы назовем эту стратегию *алгоритмом лучшего углового хода*. Также добавим несколько других функций, реализующих различные стратегии, включая *алгоритм худшего хода*, который ищет ход с худшим по счету результатом; *алгоритм случайного хода*, который ищет любой допустимый ход; и *алгоритм лучшего углового-граничного хода*, работающий так же, как и алгоритм лучшего углового хода, за исключением того, что после углового хода и перед ходом, переворачивающим больше всего фишек, он ищет граничный ход.

В файле *AISim3.py* вызов `getComputerMove()` в строке 257 будет заменен на функцию `getCornerBestMove()`, а функция `getComputerMove()` в строке 274 станет `getWorstMove()` — функцией, которую мы напишем для алгоритма худшего хода. Таким образом, мы заставим соревноваться обычный алгоритм лучшего углового хода и алгоритм, который целенаправленно выбирает ход, переворачивающий *меньше всего* фишек.

Исходный код модели 3

Измените код в файле *AISim3.py*, чтобы он соответствовал приведенному ниже коду. Убедитесь, что меняете код строка за строкой в порядке очереди. Если при выполнении программы возникают ошибки, сравните код, который вы набрали, с оригинальным кодом с помощью онлайн-инструмента на сайте **inventwithpython.com/diff/**.

AISim3.py

```
162. def getCornerBestMove(board, computerTile):

--пропуск–
1

184. def getWorstMove(board, tile):
185.     # Вернуть ход, который переворачивает меньше всего фишек.
186.     possibleMoves = getValidMoves(board, tile)
187.     random.shuffle(possibleMoves) # Сделать случайным порядок ходов.
188.
189.     # Найти ход с наименьшим возможным количеством очков.
190.     worstScore = 64
```

```
191.     for x, y in possibleMoves:
192.         boardCopy = getBoardCopy(board)
193.         makeMove(boardCopy, tile, x, y)
194.         score = getScoreOfBoard(boardCopy)[tile]
195.         if score < worstScore:
196.             worstMove = [x, y]
197.             worstScore = score
198.
199.     return worstMove
200.
201. def getRandomMove(board, tile):
202.     possibleMoves = getValidMoves(board, tile)
203.     return random.choice(possibleMoves)
204.
205. def isOnSide(x, y):
206.     return x == 0 or x == WIDTH - 1 or y == 0 or y == HEIGHT — 1
207.
208. def getCornerSideBestMove(board, tile):
209.     # Вернуть угловой ход, граничный ход или лучший ход.
210.     possibleMoves = getValidMoves(board, tile)
211.     random.shuffle(possibleMoves) # Сделать случайным порядок ходов.
212.
213.     # Всегда делать ход в угол, если это возможно.
214.     for x, y in possibleMoves:
215.     if isOnCorner(x, y):
216.         return [x, y]
217.
218.     # Если сделать ход в угол нельзя, вернуть граничный ход.
219.     for x, y in possibleMoves:
220.         if isOnSide(x, y):
221.             return [x, y]
222.
223.     return getCornerBestMove(board, tile) # Делать то, что делал бы обычный ИИ.
224.
225. def printScore(board, playerTile, computerTile):
```

--*пропуск*--

```
257.                 move = getCornerBestMove(board, playerTile)
```

--*пропуск*--

```
274.                 move = getWorstMove(board, computerTile)
```

Запуск файла *AISim3.py* выводит то же самое, что и *AISim2.py*, за исключением того, что теперь будут использоваться разные алгоритмы.

Принципы работы искусственных интеллектов в модели 3

Функции `getCornerBestMove()`, `getWorstMove()`, `getRandomMove()` и `getCornerSideBestMove()` похожи друг на друга, но используют несколько отличающиеся игровые стратегии. Одна из них применяет новую функцию `isOnSide()`. Она аналогична нашей функции `isOnCorner()`, но проверяет клетки вдоль сторон поля, прежде чем выбрать самый результативный ход.

ИИ лучшего углового хода

У нас уже есть код для ИИ, который выбирает угловой ход, а затем выбирает лучший ход из возможных, ведь это именно то, что делает функция `getComputerMove()`. Мы можем просто изменить имя `getComputerMove()` на что-то более описательное, так что измените строку 162 и переименуйте нашу функцию в `getCornerBestMove()`.

```
162. def getCornerBestMove(board, computerTile):
```

Поскольку функции `getComputerMove()` больше не существует, нам необходимо обновить код в строке 257, указав там имя `getCornerBestMove()`.

```
257.                move = getCornerBestMove(board, playerTile)
```

Это все, что нам нужно было сделать для данного ИИ, поэтому давайте двигаться дальше.

ИИ худшего хода

ИИ худшего хода просто находит ход с наименьшим количеством очков и возвращает его. Его код очень похож на тот, который мы использовали для поиска хода с наибольшим количеством очков в нашем исходном алгоритме `getComputerMove()`.

```
184. def getWorstMove(board, tile):
185.     # Вернуть ход, который переворачивает меньше всего фишек.
186.     possibleMoves = getValidMoves(board, tile)
187.     random.shuffle(possibleMoves) # Сделать случайным порядок ходов.
188.
189.     # Найти ход с наименьшим возможным количеством очков.
```

```
190.     worstScore = 64
191.     for x, y in possibleMoves:
192.         boardCopy = getBoardCopy(board)
193.         makeMove(boardCopy, tile, x, y)
194.         score = getScoreOfBoard(boardCopy)[tile]
195.         if score < worstScore:
196.             worstMove = [x, y]
197.             worstScore = score
198.
199.     return worstMove
```

Алгоритм `getWorstMove()` начинается аналогичным кодом в строках 186 и 187, но затем со строки 190 появляются отличия. Мы назначили переменную `worstScore` (худший результат) вместо `bestScore` (лучший результат) и присвоили ей значение 64, так как это общее число позиций на поле и наибольшее количество очков, которые можно было бы заработать, если бы все поле было заполнено. Строки с 191 по 194 такие же, как и исходный алгоритм, но затем код в строке 195 проверяет, меньше ли значение переменной `score` чем `worstScore`, вместо того, чтобы проверять, больше ли значение переменной `score`. Если значение переменной `score` меньше, тогда `worstMove` заменяется ходом на поле, который в настоящее время тестирует алгоритм, и значение переменной `worstScore` также обновляется. Потом функция возвращает значение переменной `worstMove`.

Наконец, функция `getComputerMove()` в строке 274 должна быть заменена функцией `getWorstMove()`.

```
274.                 move = getWorstMove(board, computerTile)
```

Внесите эти изменения и запустите программу, и `getCornerBestMove()` и `getWorstMove()` будут играть друг против друга.

ИИ случайного хода

ИИ случайного хода просто находит все возможные допустимые ходы, а затем случайным образом выбирает один из них.

```
201. def getRandomMove(board, tile):
202.     possibleMoves = getValidMoves(board, tile)
203.     return random.choice(possibleMoves)
```

Здесь используется функция getValidMoves(), как и во всех остальных ИИ, а затем применяется функция choice(), чтобы в списке вернуть один из возможных ходов.

Проверка на граничные ходы

Прежде чем перейти к алгоритмам, давайте рассмотрим одну новую вспомогательную функцию, добавленную нами. Вспомогательная функция isOnSide() подобна функции isOnCorner(), только она проверяет, приходится ли ход на стороны игрового поля.

```
205. def isOnSide(x, y):
206.     return x == 0 or x == WIDTH - 1 or y == 0 or y == HEIGHT - 1
```

Она содержит одно логическое выражение, которое проверяет, равны ли значения координат x или y, переданных ей в качестве аргументов, 0 или 7. Любая координата со значением 0 или 7 находится на краю поля.

Мы будем использовать эту функцию далее в ИИ лучшего углового-граничного хода.

ИИ лучшего углового-граничного хода

Этот ИИ работает во многом так же, как ИИ лучшего углового хода, поэтому мы можем повторно использовать часть уже введенного кода. Мы определяем этот ИИ в функции getCornerSideBestMove().

```
208. def getCornerSideBestMove(board, tile):
209.     # Вернуть угловой ход, граничный ход или лучший ход.
210.     possibleMoves = getValidMoves(board, tile)
211.     random.shuffle(possibleMoves) # Сделать случайным порядок ходов.
212.
213.     # Всегда делать ход в угол, если это возможно.
214.     for x, y in possibleMoves:
215.     if isOnCorner(x, y):
216.         return [x, y]
217.
218.     # Если сделать ход в угол нельзя, вернуть граничный ход.
219.     for x, y in possibleMoves:
220.         if isOnSide(x, y):
221.             return [x, y]
222.
223.     return getCornerBestMove(board, tile) # Делать то, что делал бы обычный ИИ.
```

Строки 210 и 211 такие же, как и в ИИ лучшего углового хода, а строки 214–216 идентичны нашему алгоритму, проверяющему наличие углового хода в исходном ИИ `getComputerMove()`. Если угловой ход недоступен, тогда код в строках 219–221 проверяют наличие граничного хода с помощью вспомогательной функции `isOnSide()`. В случае если было проверено наличие всех угловых и граничных ходов, а хода еще нет, мы повторно используем нашу функцию `getCornerBestMove()`. Поскольку ранее угловые ходы найдены не были, их все еще не будет, когда код достигнет функции `getCornerBestMove()`, так что эта функция просто найдет ход с наибольшим количеством очков и вернет его.

В таблице 16.1 рассматриваются созданные нами новые алгоритмы.

Таблица 16.1. Функции, используемые для искусственных интеллектов игры «Реверси»

Функция	Описание
`getCornerBestMove()`	Сделать угловой ход, если возможно. Если нет, найти ход с наибольшим количеством очков
`getCornerSideBestMove()`	Сделать угловой ход, если возможно. Если нет, выбрать граничную клетку. Если граничных ходов нет, использовать обычный алгоритм `getCornerBestMove()`
`getRandomMove()`	Случайным образом выбрать допустимый ход
`getWorstMove()`	Выбрать позицию, ход на которую перевернет меньше всего фишек

Теперь, когда у нас есть все алгоритмы, мы можем «натравить» их друг на друга.

Сравнение искусственных интеллектов

В нашей программе ИИ лучшего углового хода играет против ИИ худшего хода. Мы можем запустить программу для моделирования, насколько хорошо ИИ действуют друг против друга, и проанализировать результаты с помощью статистики.

В дополнение к этим двум ИИ мы создали несколько других, которые не вызвали. Эти ИИ содержатся в коде, но не используются, поэтому, если мы хотим увидеть, как они выступят в матче друг с другом, нужно будет отредактировать код для их вызова. Так как у нас уже настроено одно сравнение, давайте посмотрим, как ИИ худшего хода будет справляться с ИИ лучшего углового хода.

ИИ худшего хода против ИИ лучшего углового хода

Запустите программу, чтобы выставить функцию `getCornerBestMove()` против `getWorstMove()`. Неудивительно, что стратегия переворачивания наименьшего количества фишек за ход привела к поражению в большинстве партий.

```
Количество выигрышей X: 206 (82.4%)
Количество выигрышей O: 41 (16.4%)
Количество ничьих:   3 (1.2%)
```

А вот что *удивительно*, так это то, что порой стратегия худшего хода все же работает! Вместо выигрыша в 100% случаев, алгоритм функции `getCornerBestMove()` побеждает лишь примерно в 80% случаев. Примерно 1 раз в 5 партий он уступает!

В этом сила моделирования: вы можете найти новые идеи, на осознание которых ушло бы гораздо больше времени, если бы вы играли самостоятельно. Компьютер намного быстрее!

ИИ случайного хода против ИИ лучшего углового хода

Давайте испробуем другую стратегию. В строке 274 замените вызов функции `getWorstMove()` вызовом `getRandomMove()`.

```
274.                move = getRandomMove(board, computerTile)
```

Когда вы запустите программу сейчас, вывод будет выглядеть примерно так:

```
Приветствуем в игре "Реверси"!
#1: X набрал 32 очков. O набрал 32 очков.
#2: X набрал 44 очков. O набрал 20 очков.
#3: X набрал 31 очков. O набрал 33 очков.
#4: X набрал 45 очков. O набрал 19 очков.
#5: X набрал 49 очков. O набрал 15 очков.

--пропуск--

#249: X набрал 20 очков. O набрал 44 очков.
#250: X набрал 38 очков. O набрал 26 очков.
Количество выигрышей X: 195 (78.0%)
Количество выигрышей O: 48 (19.2%)
Количество ничьих:   7 (2.8%)
```

Алгоритм случайного хода `getRandomMove()` справился немного лучше, чем алгоритм худшего хода. Это логично, потому что разумный выбор обычно лучше, чем выбор хода наугад, но выбор наугад немного лучше, чем целенаправленный выбор худшего хода.

ИИ лучшего углового-граничного хода против ИИ лучшего углового хода

Выбор угловой клетки, если она доступна, — хорошая мысль, потому что фишка в углу не может быть перевернута. Помещение фишки в граничные клетки кажется неплохой идеей, так как существует меньше способов окружить и перевернуть такую фишку. Но разве это преимущество оправдывает отказ от ходов, которые переворачивают больше фишек? Давайте выясним, столкнув алгоритм лучшего углового угла и алгоритм лучшего углового-граничного хода.

Измените алгоритм в строке 274, чтобы использовать функцию `getCornerSideBestMove()`.

```
274.                move = getCornerSideBestMove(board, computerTile)
```

Затем снова запустите программу.

```
Приветствуем в игре "Реверси"!
#1: X набрал 27 очков. O набрал 37 очков.
#2: X набрал 39 очков. O набрал 25 очков.
#3: X набрал 41 очков. O набрал 23 очков.

--пропуск--

#249: X набрал 48 очков. O набрал 16 очков.
#250: X набрал 38 очков. O набрал 26 очков.
Количество выигрышей X: 152 (60.8%)
Количество выигрышей O: 89 (35.6%)
Количество ничьих:   9 (3.6%)
```

Ух ты! Это неожиданно. Оказывается, выбор граничных клеток вместо клетки, переворачивающей больше фишек, — плохая стратегия. Преимущество граничной клетки не перевешивает ущерб от переворачивания меньшего количества фишек противника. Можем ли мы быть уверены в этих результатах? Давайте запустим программу еще раз, но сыграем 1000 партий, заменив код в строке 278 в файле *AISim3.py* на `NUM_GAMES = 1000`. Теперь программа, наверное, будет работать на протяжении нескольких минут, вам же понадобилось бы несколько *недель*, чтобы сделать то же самое вручную!

Вы увидите, что более точные статистические данные, полученные после 1000 партий, в целом согласующиеся статистике 250 партий. Кажется, выбор хода, переворачивающего наибольшее количество фишек, — более удачная идея, чем выбор граничной клетки.

Таким образом, с помощью программирования мы выяснили, какая игровая стратегия работает лучше всего. Когда вы слышите, что ученые используют компьютерные модели, имеется в виду именно это. Они используют модели для воссоздания какого-то реального процесса, а затем проводят тесты в этой модели, чтобы узнать больше о реальном мире.

Заключение

В этой главе не описывалась новая игра, а были сымитированы различные стратегии для игры «Реверси». Если бы мы считали, что граничные ходы в «Реверси» — это хорошая идея, нам пришлось бы провести недели, даже месяцы, вручную играя в «Реверси» и внимательно записывая результаты, чтобы проверить, так ли это на самом деле. Но если нам известен способ научить компьютер самостоятельно играть в «Реверси», тогда мы можем сделать так, чтобы он испытывал различные стратегии за нас. Только подумайте, компьютер выполняет миллионы строк нашей программы на Python за считанные секунды! Эксперименты с моделями игры «Реверси» могут помочь вам узнать больше о том, как играть в нее в реальной жизни.

Вообще-то эта глава могла бы послужить основой для хорошего научного проекта. Вы могли бы исследовать, какой набор ходов приводит к наибольшему числу выигрышей в сравнении с другими наборами ходов, и могли бы составить гипотезу о том, какая стратегия лучшая. После тестирования нескольких моделей, вы могли бы определить наверняка, какая стратегия работает лучше всего. С программированием вы можете сделать проект на основе моделирования любой настольной игры! И все это потому, что вы знаете, как поручить компьютеру делать это, шаг за шагом, строка за строкой. Вы умеете общаться на языке компьютера, заставлять его обрабатывать для вас большие объемы данных и выполнять сложные числовые расчеты.

Это была последняя текстовая игра в этой книге. Конечно, такие игры могут быть интересны, хоть они и просты. Но в большинстве современных игр используется графика, звук и анимация, что делает игру более захватывающей. В следующих главах этой книги вы узнаете, как создавать игры с графикой, используя модуль Python под названием `pygame`.

17

СОЗДАНИЕ ГРАФИКИ

До сих пор все наши игры были исключительно текстовые. Текст отображался на экране при выводе, текст набирался игроком при вводе. Использование только текста упрощает процесс обучения программированию. В этой главе мы создадим более занимательные программы с графикой, используя модуль `pygame`.

Главы 17–20 научат вас использовать модуль `pygame` для разработки игр с графикой, анимацией, звуком и вводом с помощью мыши. В этих главах мы напишем исходный код для простых программ, отражающих методы `pygame`. Затем, в главе 21? мы соберем все выученные ранее идеи для создания игры.

В ЭТОЙ ГЛАВЕ РАССМАТРИВАЮТСЯ СЛЕДУЮЩИЕ ТЕМЫ:

- Установка `pygame`
- Цвета и шрифты в `pygame`
- Сглаженная графика
- Атрибуты
- Типы данных `pygame.font.Font`, `pygame.Surface`, `pygame.Rect` и `pygame.PixelArray`
- Функции-конструкторы
- Функции рисования модуля `pygame`
- Метод `blit()` для объектов поверхности
- События

Установка pygame

Модуль pygame помогает разработчикам создавать игры, упрощая отображение графики на экране компьютера или процесс добавления музыки в программы. Модуль не поставляется с Python, но, как и сам Python, его можно установить бесплатно. Для этого нужно запустить *оболочку командной строки* (а не среду разработки Python!) от имени администратора (командная строка в Windows и программа Терминал (Terminal) в macOS/Linux) и выполнить следующую команду:

```
>>> pip3 install pygame
```

Система pip самостоятельно скачает и установит подходящую версию pygame. Если возникает ошибка установки, может потребоваться указать полный путь к исполняемому файлу системы pip. К примеру, в Windows это делается так:

```
>>> C:\Python36\Scripts\pip3 install pygame
```

Чтобы проверить правильность установки, введите в интерактивной оболочке Python следующую команду:

```
>>> import pygame
```

Если после нажатия клавиши **Enter** ничего не происходит, значит, pygame был установлен успешно. Если появляется ошибка ImportError: No module named pygame, попробуйте переустановить pygame (и убедитесь, что вы правильно ввели команду import pygame).

Примечание. При написании кода программ Python не сохраняйте файл под именем *pygame.py*. Если вы это сделаете, строка import pygame импортирует ваш файл вместо настоящего модуля pygame и код перестанет работать.

Привет, pygame!

Для начала мы напишем новую pygame-программу «Привет, мир!», подобную той, которую вы создавали в начале книги. На этот раз вы будете использовать модуль pygame, чтобы надпись «Привет, мир!» появилась в графи-

ческом окне, а не в виде текста. В этой главе мы будем использовать модуль pygame только для того, чтобы отображать в окне фигуры и линии, но скоро вам пригодятся эти навыки, когда вы будете создавать свою первую анимационную игру.

Модуль pygame плохо совместим с интерактивной оболочкой, поэтому писать программы с помощью pygame вы можете только в редакторе.

Кроме того, в программах pygame не используются функции print() или input(). Там нет вводимого или выводимого текста. Вместо этого pygame показывает вывод путем отображения графики и текста в отдельном окне. Ввод в модуле pygame осуществляется с клавиатуры и мыши посредством *событий*, описанных в разделе «События и игровой цикл» в конце этой главы.

Пример запуска pygame-программы «Привет, мир!»

Когда вы запускаете графическую программу «Привет, мир!», должно появиться новое окно, показанное на рис. 17.1.

Рис. 17.1. Pygame-программа «Привет, мир!»

Самое удобное в использовании окна вместо консоли заключается в том, что текст может появляться в любом месте окна, а не только после текста, который вы вывели до этого. Шрифт может быть любого цвета и размера. Окно представляет собой что-то вроде холста, и вы можете рисовать на нем все, что пожелаете.

Исходный код pygame-программы «Привет, мир!»

В редакторе файлов создайте новый файл, выбрав команду меню **File** ⇒ **New File** (Файл ⇒ Новый файл). В открывшемся окне введите приведенный ниже исходный код и сохраните файл под именем *pygameHelloWorld.py*. Затем нажмите клавишу **F5** и запустите программу. Если при выполнении программы возникают ошибки, сравните код, который вы набрали, с оригинальным кодом с помощью онлайн-инструмента на сайте **inventwithpython.com/diff/**.

pygameHelloWorld.py

```
import pygame, sys
from pygame.locals import *

# Настройка pygame.
pygame.init()

# Настройка окна.
windowSurface = pygame.display.set_mode((500, 400), 0, 32)
pygame.display.set_caption('Привет, мир!')

# Назначение цветов.
BLACK = (0, 0, 0)
WHITE = (255, 255, 255)
RED = (255, 0, 0)
GREEN = (0, 255, 0)
BLUE = (0, 0, 255)

# Назначение шрифтов.
basicFont = pygame.font.SysFont(None, 48)

# Настройка текста.
text = basicFont.render('Привет, мир!', True, WHITE, BLUE)
textRect = text.get_rect()
textRect.centerx = windowSurface.get_rect().centerx
textRect.centery = windowSurface.get_rect().centery

# Нанесение на поверхность белого фона.
```

```
28. windowSurface.fill(WHITE)
29.
30. # Нанесение на поверхность зеленого многоугольника.
31. pygame.draw.polygon(windowSurface, GREEN, ((146, 0), (291, 106), (236, 277), (56, 277), (0, 106)))
32.
33. # Нанесение на поверхность синих линий.
34. pygame.draw.line(windowSurface, BLUE, (60, 60), (120, 60), 4)
35. pygame.draw.line(windowSurface, BLUE, (120, 60), (60, 120))
36. pygame.draw.line(windowSurface, BLUE, (60, 120), (120, 120), 4)
37.
38. # Нанесение на поверхность синего круга.
39. pygame.draw.circle(windowSurface, BLUE, (300, 50), 20, 0)
40.
41. # Нанесение на поверхность красного эллипса.
42. pygame.draw.ellipse(windowSurface, RED, (300, 250, 40, 80), 1)
43.
44. # Нанесение на поверхность фонового прямоугольника для текста.
45.pygame.draw.rect(windowSurface,RED,(textRect.left-20,textRect.top-20,textRect.width+40,textRect.height+40))
46.
47. # Получение массива пикселов поверхности.
48. pixArray = pygame.PixelArray(windowSurface)
49. pixArray[480][380] = BLACK
50. del pixArray
51.
52. # Нанесение текста на поверхность.
53. windowSurface.blit(text, textRect)
54.
55. # Отображение окна на экране.
56. pygame.display.update()
57.
58. # Запуск игрового цикла.
59. while True:
60.     for event in pygame.event.get():
61.         if event.type == QUIT:
62.             pygame.quit()
63.             sys.exit()
```

Импорт модуля pygame

Давайте рассмотрим каждую из этих строк кода и узнаем, что они делают.

Сначала вам нужно импортировать модуль pygame, чтобы получить возможность вызывать его функции. Вы можете импортировать несколько модулей в одной строке, разделяя имена модулей запятыми. Код в строке 1 импортирует модули pygame и sys.

```
1. import pygame, sys
2. from pygame.locals import *
```

Код во второй строке импортирует модуль pygame.locals. Этот модуль содержит много констант, которые вы будете использовать с pygame, таких как QUIT (помогает выйти из программы) и K_ESCAPE (обрабатывает нажатие клавиши **Esc**).

Код в строке 2 также позволяет вам использовать модуль pygames.locals без дополнительного указания кода pygames.locals. перед каждым методом, константой или чем-либо еще, что вы вызываете из модуля.

Если вместо кода import sys указать from sys import *, можно вызывать просто функцию exit() вместо sys.exit(). Но в большинстве случаев лучше всего использовать полное имя функции, чтобы знать, в каком модуле она находится.

Инициализация pygame

Код любой pygame-программы должен содержать вызов pygame.init() после импорта модуля pygame, но перед вызовом любых других функций pygame.

```
4. # Настройка pygame.
5. pygame.init()
```

Эта операция инициализирует pygame, то есть подготавливает модуль к использованию. Вам не нужно знать, что делает init(); просто запомните, что нужно выполнить такой вызов, прежде чем использовать любые другие функции pygame.

Настройка окна pygame

Код в строке 8 создает окно *графического интерфейса пользователя (англ. GUI, graphical user interface)*, вызывая метод set_mode() в модуле pygame.display.

(display — это модуль в составе модуля pygame. Даже у модуля pygame есть свои собственные модули!)

```
7. # Настройка окна.
8. windowSurface = pygame.display.set_mode((500, 400), 0, 32)
9. pygame.display.set_caption('Привет, мир!')
```

Эти методы помогают настроить окно, в котором будет работать pygame. Как и в игре «Охотник за сокровищами», эти окна используют систему координат, но их система координат организована в пикселях.

Пиксель — наименьшая точка на экране вашего компьютера. Пиксель может быть окрашен в любой цвет. Все пиксели на экране работают вместе, отображая изображения, которые вы видите. Мы создадим окно размером 500 пикселей в ширину и 400 пикселей в высоту с помощью *кортежа*.

Кортежи

Кортежи подобны спискам, за исключением того, что кортежи используют круглые скобки вместо квадратных и, как и строки, кортежи нельзя изменять. Для примера введите следующие команды в интерактивной оболочке:

```
   >>> spam = ('Жизнь', 'Вселенная', 'Бытие', 42)
❶  >>> spam[0]
   'Жизнь'
   >>> spam[3]
   42
❷  >>> spam[1:3]
   ('Вселенная', 'Бытие')
❸  >>> spam[3] = 'привет'
❹  Traceback (most recent call last):
       File "<pyshell#4>", line 1, in <module>
           spam[3] = 'привет'
   TypeError: 'tuple' object does not support item assignment
```

Как видно из примера, если требуется получить только один элемент кортежа ❶ или диапазон элементов ❷, то по-прежнему используете квадратные скобки, как и в случае со списком. Однако если вы попытаетесь заменить элемент с индексом 3 строкой 'привет' ❸, Python выдаст ошибку ❹.

Мы будем использовать кортежи для настройки окон pygame. У метода pygame.display.set_mode() есть три параметра. Первый — это кортеж из двух целых чисел для ширины и высоты окна в пикселях. Чтобы создать окно размером 500×400 пикселей, нужно использовать кортеж (500, 400) для первого аргумента функции set_mode(). Второй и третий параметры касаются проблем, выходящих за рамки этой книги. Просто передайте 0 и 32 соответственно.

Объекты поверхности

Функция set_mode() возвращает объект pygame.Surface (который мы для краткости будем называть объектом Surface). *Объект* — это экземпляр некоторого типа данных, имеющего свои методы. Например, строки в Python являются объектами, потому что они содержат данные (саму строку) и у них есть методы (например, lower() и split()). Объект Surface представляет окно.

Переменные хранят ссылки на объекты так же, как они хранят ссылки для списков и словарей (см. раздел «Ссылки на список» в главе 10).

Метод set_caption() в строке 9 назначает заголовок окна — 'Привет, мир!'. Заголовок находится в верхнем левом углу окна.

Работа с цветом

У пикселей есть три основных цвета: красный, зеленый и синий. Объединяя эти три цвета в различных пропорциях (что и делает монитор вашего компьютера), вы можете получить любой оттенок. В pygame оттенки представлены кортежами из трех целых чисел. Они называются *цветами RGB*, и в нашей программе мы будем их использовать для окрашивания пикселей. Поскольку неудобно указывать кортеж из трех чисел каждый раз, когда требуется использовать определенный оттенок, мы создадим константы для хранения кортежей, названные по цвету, который представляет соответствующий кортеж.

```
11. # Назначение цветов.
12. BLACK = (0, 0, 0)
13. WHITE = (255, 255, 255)
14. RED = (255, 0, 0)
15. GREEN = (0, 255, 0)
16. BLUE = (0, 0, 255)
```

Первое значение в кортеже определяет, какое количество красного содержится в данном оттенке. Значение 0 сообщает о том, что в оттенке нет красного, а значение 255 указывает на максимальное количество красного цвета. Второе

значение предназначено для зеленого цвета, третье — для синего. Эти три целых числа образуют кортеж RGB. Например, кортеж (`0, 0, 0`) не содержит ни красного, ни зеленого, ни синего цветов. Полученный цвет полностью черный, как в строке 12. Кортеж (`255, 255, 255`) содержит максимальное количество красного, зеленого и синего цветов, что в сочетании дает белый цвет, как в строке 13.

Мы также будем использовать красный, зеленый и синий цвета, которые назначаются в строках с 14 по 16. Кортеж (`255, 0, 0`) содержит максимальное количество красного, и поскольку там нет ни зеленого, ни синего, то в результате будет получен цвет. Аналогично (`0, 255, 0`) — зеленый цвет, а (`0, 0, 255`) — синий.

Вы можете смешивать количества красного, зеленого и синего цветов и получать любой оттенок. Таблица 17.1 представляет некоторые часто используемые цвета и их соответствующие значения RGB. На веб-сайте **htmlcolorcodes.com** вы найдете дополнительную информацию о RGB-цветах.

Таблица 17.1. Цвета и соответствующие значения RGB

Цвет	Значение RGB
Черный	`(0, 0, 0)`
Синий	`(0, 0, 255)`
Серый	`(128, 128, 128)`
Зеленый	`(0, 128, 0)`
Лаймовый	`(0, 255, 0)`
Пурпурный	`(128, 0, 128)`
Красный	`(255, 0, 0)`
Бирюзовый	`(0, 128, 128)`
Белый	`(255, 255, 255)`
Желтый	`(255, 255, 0)`

Мы будем использовать только пять определенных нами цветов. Но в своих программах вы можете использовать любой из этих цветов или даже составлять собственные оттенки.

Вывод текста в окне `pygame`

Вывод текста в окне немного отличается от использования функции `print()` в текстовых играх. Чтобы вывести текст в окне, сначала нужно произвести кое-какие настройки.

Использование шрифтов для оформления текста

Шрифт — это полный набор букв, цифр и специальных символов, выполненных в одном стиле. Каждый раз, как нам нужно будет вывести текст

в окне pygame, мы будем использовать шрифты. На рис. 17.2 показано одно и то же предложение с использованием разных шрифтов.

Рис. 17.2. Примеры различных шрифтов

В наших предыдущих играх мы инструктировали Python просто выводить текст. Цвет, размер и шрифт для отображения этого текста полностью определялись настройками операционной системы. Программа Python не влияла на шрифт. Однако модуль pygame может вывести текст любым шрифтом. Код в строке 19 создает объект pygame.font.Font (для краткости будем называть его объектом Font), вызывая функцию pygame.font.SysFont() с двумя параметрами.

```
18. # Назначение шрифтов.
19. basicFont = pygame.font.SysFont(None, 48)
```

Первый параметр — это название шрифта, но мы передадим значение None, чтобы использовался системный шрифт по умолчанию. Второй параметр — размер шрифта (который измеряется в единицах, называемых *пунктами*). Мы выведем в окне надпись 'Привет, мир!', используя шрифт по умолчанию размером 48 пунктов. Создание изображения из букв для текста «Привет, мир!» называется *рендерингом*.

Рендеринг объекта Font

Объект Font, который вы сохранили в переменной basicFont, имеет метод под названием render(). Этот метод вернет объект Surface с текстом, ото-

браженном на нем. Первым параметром метода `render()` является строка с текстом для отображения. Второй параметр — логическое значение, указывающее, следует ли производить *сглаживание* шрифта. Сглаживание слегка размывает текст, чтобы придать ему более гладкий вид. Рисунок 17.3 показывает, как выглядит линия со сглаживанием и без него.

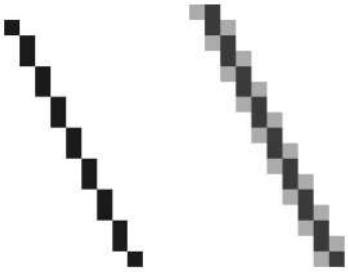


Рис. 17.3. Увеличенное изображение линии до и после сглаживания

Чтобы использовать сглаживание, мы передадим значение `True` в строке 22.

```
21. # Настройка текста.
22. text = basicFont.render('Привет, мир!', True, WHITE, BLUE)
```

Третий и четвертый параметры в строке 22 являются кортежами RGB. Третий параметр — это цвет шрифта текста (в данном случае это белый), а четвертый — фоновый цвет позади текста (синий). Мы присваиваем объект `Font` переменной `text`.

После настройки объекта `Font` нам нужно поместить его в определенную позицию в окне.

Настройка местоположения текста с помощью атрибутов `Rect`

Тип данных `pygame.Rect` (сокращенно `Rect`) представляет прямоугольные области, у которых заданы размер и положение на экране.

Чтобы создать новый объект `Rect`, вызовите функцию `pygame.Rect()`. Обратите внимание, что функция `pygame.Rect()` имеет то же имя, что и тип данных `pygame.Rect`. Функции, которые имеют то же имя, что и их тип данных, и создают объекты или значения своего типа данных, называются *функциями-конструкторами*. Параметрами для функции `pygame.Rect()` выступают целые числа, обозначающие координаты *x* и *y* верхнего левого угла, а также ширина и высота, и указываемые в пикселях. Синтаксис функции с параметрами выглядит следующим образом: `pygame.Rect(`*`лево, верх, ширина, высота`*`)`.

Когда мы создали объект Font, объект Rect для него уже был создан, поэтому нам нужно лишь запросить его. Для этого мы используем метод get_rect() в переменной text и присваиваем Rect переменной textRect.

```
23. textRect = text.get_rect()
24. textRect.centerx = windowSurface.get_rect().centerx
25. textRect.centery = windowSurface.get_rect().centery
```

Подобно тому, как методы являются функциями, связанными с объектом, *атрибуты* — это переменные, связанные с объектом. Тип данных Rect имеет много атрибутов, которые описывают прямоугольник, который они представляют. Чтобы установить местоположение textRect в окне, нам нужно присвоить значения x и y его центра координатам в пикселях в этом окне. Поскольку у каждого объекта Rect уже есть атрибуты, которые хранят координаты x и y центра Rect, называемые centerx и centery, соответственно, все, что нам нужно сделать, — это назначить значения этих координат.

Мы хотим поместить textRect в центр окна, поэтому нужно получить объект windowSurface Rect, его атрибуты centerx и centery, а затем присвоить их соответствующим атрибутам объекта в textRect. Я сделал это в строках 24 и 25.

Мы можем использовать множество других атрибутов Rect. Таблица 17.2 перечисляет атрибуты объекта Rect с именем myRect.

Таблица 17.2. Атрибуты объекта Rect

Атрибут pygame.Rect	Описание
myRect.left	Целочисленное значение координаты *x* левой стороны прямоугольника
myRect.right	Целочисленное значение координаты *x* правой стороны прямоугольника
myRect.top	Целочисленное значение координаты *y* верхней стороны прямоугольника
myRect.bottom	Целочисленное значение координаты *y* нижней стороны прямоугольника
myRect.centerx	Целочисленное значение координаты *x* центра прямоугольника
myRect.centery	Целочисленное значение координаты *y* центра прямоугольника
myRect.width	Целочисленное значение ширины прямоугольника
myRect.height	Целочисленное значение высоты прямоугольника
myRect.size	Кортеж из двух целых чисел: (width, height)
myRect.topleft	Кортеж из двух целых чисел: (left, top)
myRect.topright	Кортеж из двух целых чисел: (right, top)
myRect.bottomleft	Кортеж из двух целых чисел: (left, bottom)
myRect.bottomright	Кортеж из двух целых чисел: (right, bottom)
myRect.midleft	Кортеж из двух целых чисел: (left, centery)
myRect.midright	Кортеж из двух целых чисел: (right, centery)
myRect.midtop	Кортеж из двух целых чисел: (centerx, top)
myRect.midbottom	Кортеж из двух целых чисел: (centerx, bottom)

Самое замечательное при работе с объектами Rect заключается в том, что если вы измените какой-либо из этих атрибутов, все остальные атрибуты автоматически изменятся сами. Например, если вы создадите объект Rect шириной и высотой по 20 пикселей с верхним левым углом в координатах (30, 40), тогда координата *x* правой стороны автоматически будет установлена равной 50 (поскольку 20 + 30 = 50).

Или, если вы вместо этого замените атрибут left строкой кода myRect.left = 100, тогда pygame автоматически изменит атрибут right на 120 (поскольку 20 + 100 = 120). Все остальные атрибуты для этого объекта Rect также обновятся.

БОЛЬШЕ О МЕТОДАХ, МОДУЛЯХ И ТИПАХ ДАННЫХ

Внутри модуля pygame есть модули font и surface, а внутри *этих* модулей — типы данных Font и Surface. Разработчики модуля pygame сделали так, чтобы имена модулей начинались со строчной буквы, а типов данных — с прописной, чтобы было проще различать типы данных и модули.

Обратите внимание, что и объект Font (хранящийся в переменной text в строке 23), и объект Surface (хранящийся в переменной windowSurface в строке 24) имеют метод get_rect(). Технически это два разных метода, но разработчики pygame дали им одно и то же имя, так как они делают одно и то же: возвращают объекты Rect, которые представляют размер и положение объекта Font или Surface.

Заливка цветом объекта Surface

В нашей программе мы заполним белым цветом всю поверхность, хранящуюся в переменной windowSurface. Функция fill() полностью зальет поверхность цветом, который вы передадите в качестве параметра. Помните, что мы присвоили переменной WHITE значение (255, 255, 255) в строке 13.

```
27. # Нанесение на поверхность белого фона.
28. windowSurface.fill(WHITE)
```

Обратите внимание, что в pygame окно на экране не изменится при вызове метода fill() или любой другой функции рисования. Вместо этого они изменят объект Surface. Для того чтобы увидеть изменения, вы должны будете вывести на экран новый объект Surface с помощью функции pygame.display.update().

Это связано с тем, что изменение объекта `Surface` осуществляется в памяти компьютера намного быстрее, чем изменение изображения на экране. Гораздо эффективнее производить вывод на экран один раз после того, как все функции рисования были применены к объекту `Surface`.

Функции рисования pygame

Мы уже научились заполнять окно цветом и добавлять текст, но в `pygame` также есть функции, позволяющие рисовать фигуры и линии. Вы можете комбинировать эти фигуры в своей графической игре, получая различные изображения.

Рисование многоугольника

Функция `pygame.draw.polygon()` позволяет нарисовать любую многоугольную фигуру, которую вы ей передадите. *Многоугольник* представляет собой многогранную фигуру со сторонами, образованными прямыми. Круги и эллипсы не являются многоугольниками, поэтому для этих фигур мы будем использовать другие функции.

Аргументы функции указываются в следующем порядке:

1. Объект `Surface`, на который будет наноситься многоугольник.
2. Цвет многоугольника.
3. Кортеж из кортежей, представляющий координаты *x* и *y* рисуемых точек по порядку. Последний кортеж автоматически соединяется с первым для завершения фигуры.
4. Опционально, целое число для толщины контура многоугольника. Без этого многоугольник будет залит целиком.

В строке 31 мы рисуем зеленый многоугольник на белом объекте `Surface`.

```
30. # Нанесение на поверхность зеленого многоугольника.
31. pygame.draw.polygon(windowSurface, GREEN, ((146, 0), (291, 106), (236, 277), (56, 277), (0, 106)))
```

Рис. 17.4. Примеры многоугольников

Так как нам нужно залить многоугольник целиком, мы не передаем необязательный параметр толщины контура. На рис. 17.4 показаны некоторые примеры многоугольников.

Рисование линии

Функция `pygame.draw.line()` проводит линию от одной точки на экране до другой. Параметры для `pygame.draw.line()` указываются в следующем порядке:

1. Объект `Surface`, на котором будет проведена линия.
2. Цвет линии.
3. Кортеж из двух целых чисел для координат *x* и *y* одного конца линии.
4. Кортеж из двух целых чисел для координат *x* и *y* другого конца линии.
5. Опционально целое число для толщины линии в пикселях.

В строках с 34 по 36 мы трижды вызываем функцию `pygame.draw.line()`.

```
33. # Нанесение на поверхность синих линий.
34. pygame.draw.line(windowSurface, BLUE, (60, 60), (120, 60), 4)
35. pygame.draw.line(windowSurface, BLUE, (120, 60), (60, 120))
36. pygame.draw.line(windowSurface, BLUE, (60, 120), (120, 120), 4)
```

Если вы не укажете параметр `width`, он примет значение по умолчанию, равное `1`. В строках 34 и 36 для `width` мы передаем значение `4`, так что линии будут иметь толщину в 4 пикселя. Три вызова `pygame.draw.line()` в строках 34, 35 и 36 позволяют отобразить синюю букву Z на объекте `Surface`.

Рисование круга

Функция `pygame.draw.circle()` рисует на объектах `Surface` круги. Ее параметры указываются в следующем порядке:

1. Объект `Surface`, на котором нужно нарисовать круг.
2. Цвет круга.
3. Кортеж из двух целых чисел для координат *x* и *y* центра круга.
4. Целое число радиуса (то есть размера) круга.
5. Опционально целое число толщины контура. Толщина `0` означает, что круг будет залит цветом без контура.

Код в строке 39 рисует синий круг на объекте `Surface`.

```
38. # Нанесение на поверхность синего круга.
39. pygame.draw.circle(windowSurface, BLUE, (300, 50), 20, 0)
```

Центр этого круга имеет координату *x*, равную 300 и координату *y*, равную 50. Радиус круга составляет 20 пикселей, круг заполнен синим цветом.

Рисование эллипса

Функция `pygame.draw.ellipse()` напоминает функцию `pygame.draw.circle()`, но вместо этого она рисует эллипс — фигуру, которая выглядит как сплющенный круг. Параметры функции `pygame.draw.ellipse()` указываются в следующем порядке:

1. Объект `Surface`, на который наносится эллипс.
2. Цвет эллипса.
3. Кортеж из четырех целых чисел для левого верхнего угла объекта `Rect` эллипса, а также ширины и высоты эллипса.
4. Опционально целое число толщины контура. Толщина `0` означает, что эллипс будет залит цветом без контура.

Код в строке 42 рисует красный эллипс на объекте `Surface`.

```
41. # Нанесение на поверхность красного эллипса.
42. pygame.draw.ellipse(windowSurface, RED, (300, 250, 40, 80), 1)
```

Координата *x* верхнего левого угла эллипса равна 300, а координата *y* — 250. Ширина фигуры 40 пикселей, высота — 80 пикселей. Контур эллипса имеет толщину в 1 пиксель.

Рисование прямоугольника

Функция `pygame.draw.rect()` создает прямоугольники. Параметры функции `pygame.draw.rect()` указываются в следующем порядке:

1. Объект `Surface`, на котором нужно нарисовать прямоугольник.
2. Цвет прямоугольника.
3. Кортеж из четырех целых чисел для координат *x* и *y* верхнего левого угла, ширины и высоты прямоугольника. Вместо кортежа из четырех целых чисел вы также можете передать в качестве третьего параметра объект `Rect`.

В программе «Привет, мир!» отображаемый прямоугольник на 20 пикселей будет выступать за пределы всех границ текста. Помните, что в строке 23 мы создали переменную `textRect`, содержащую наш текст. В строке 45 мы присваиваем левой и верхней точкам прямоугольника соответствующие значения `textRect` минус 20 (вычитаем, поскольку координаты уменьшаются по мере продвижения влево и вверх).

```
44. # Нанесение на поверхность фонового прямоугольника для текста.
45. pygame.draw.rect(windowSurface, RED, (textRect.left - 20, textRect.top - 20, textRect.width + 40, textRect.height + 40))
```

Ширина и высота прямоугольника равны ширине и высоте textRect плюс 40. Мы используем 40, а не 20, поскольку лево и верх были перемещены к началу координат на 20 пикселей, поэтому вам нужно компенсировать это пространство.

Окрашивание пикселей

Код в строке 48 создает объект pygame.PixelArray (для краткости, объект PixelArray). Объект PixelArray — это список списков цветовых кортежей, представляющий объект Surface, который вы ему передали.

Объект PixelArray позволяет управлять каждым пикселем, поэтому прекрасно подойдет, если вам нужно вывести на экран глубоко детализированные или специфические изображения вместо простых больших форм.

Мы будем использовать PixelArray для окраски одного пикселя на объекте windowSurface в черный цвет. Этот пиксель можно заметить в правом нижнем углу окна при запуске pygame-программы «Привет, мир!».

Код в строке 48 передает объект windowSurface вызову pygame.PixelArray(), поэтому присвоение значения константы BLACK переменной pixArray[480][380] в строке 49 окрасит пиксель с координатами (480, 380) в черный цвет.

```
47. # Получение массива пикселов поверхности.
48. pixArray = pygame.PixelArray(windowSurface)
49. pixArray[480][380] = BLACK
```

Модуль pygame автоматически внесет в объект windowSurface это изменение.

Первый индекс в объекте PixelArray соответствует координате *x*. Второй индекс — координате *y*. Объекты PixelArray упрощают процесс окрашивания отдельных пикселей на объекте Surface в определенный цвет.

Каждый раз, когда вы создаете объект PixelArray из объекта Surface, этот объект Surface блокируется. Это означает, что на этом объекте Surface не могут быть осуществлены вызовы метода blit() (описано ниже). Чтобы разблокировать объект Surface, нужно удалить объект PixelArray с помощью инструкции del.

```
50. del pixArray
```

Если вы забудете это сделать, то получите сообщение об ошибке: pygame.error: Surfaces must not be locked during blit.

Метод blit() для объектов Surface

Метод blit() будет рисовать содержимое одного объекта Surface на другом объекте Surface. Все текстовые объекты, созданные методом render(), существуют на их собственном объекте Surface. Все методы рисования pygame могут указывать конкретный объект Surface, на котором нужно рисовать фигуры или линии, но наш текст был сохранен в переменной text, а не нарисован на объекте windowSurface. Чтобы нанести text на тот объект Surface, где мы хотим его видеть, нужно использовать метод blit().

```
52. # Нанесение текста на поверхность.
53. windowSurface.blit(text, textRect)
```

Код в строке 53 рисует объект Surface 'Привет, мир!' из переменной text (определенной в строке 22) на объекте Surface, хранящемся в переменной windowSurface.

Второй параметр метода blit() указывает позицию на windowSurface, в котором должна быть нарисована поверхность text. Объект Rect, который вы получили из вызова метода text.get_rect() в строке 23, передается в качестве этого параметра.

Вывод объекта Surface на экран

Поскольку в pygame на экран ничего не выводится до тех пор, пока не вызывается функция pygame.display.update(), мы вызываем ее в строке 56 для отображения нашего обновленного объекта Surface.

```
55. # Отображение окна на экране.
56. pygame.display.update()
```

Для экономии памяти не стоит обновлять экран после вызова каждой отдельной функции рисования; вместо этого лучше обновить его лишь раз, после того, как были вызваны все функции рисования.

События и игровой цикл

Все наши текстовые игры осуществляли полный вывод, пока не достигли бы вызова функции input(). Тогда программа остановилась бы и ждала, когда пользователь что-то наберет и нажмет клавишу **Enter**. Но pygame-программы

постоянно находятся в *игровом цикле*, выполняя каждую строку кода в этом цикле около 100 раз в секунду.

Игровой цикл постоянно проверяет наличие новых событий, обновляет состояние окна и отображает окно на экране. *События* генерируются `pygame` каждый раз, когда пользователь нажимает клавишу на клавиатуре, кнопку мыши, перемещает курсор или выполняет иные распознаваемые программой действия, которые должны повлиять на что-то в игре. `Event` — объект типа данных `pygame.event.Event`.

Строка 59 — начало игрового цикла.

```
58. # Запуск игрового цикла.
59. while True:
```

Условие для инструкции `while` установлено истинным, чтобы этот цикл выполнялся бесконечно. Цикл прервется исключительно в том случае, если событие приведет к завершению работы программы.

Получение объектов `Event`

Функция `pygame.event.get()` проверяет любые новые объекты `pygame.event.Event` (сокращенно объекты `Event`), созданные с момента последнего вызова `pygame.event.get()`. Эти события возвращаются как список объектов `Event`, которые программа затем выполнит, чтобы произвести некоторые действия в ответ на событие. Все объекты `Event` имеют атрибут с именем `type`, который сообщает нам тип события. В этой главе нам нужно использовать только тип события `QUIT`, который сигнализирует о выходе пользователя из программы.

```
60.     for event in pygame.event.get():
61.         if event.type == QUIT:
```

В строке 60 мы используем цикл `for` для итерации по каждому объекту `Event` в списке, возвращаемом `pygame.event.get()`. Если атрибут `type` события равен константе `QUIT`, содержащейся в модуле `pygame.locals`, который мы импортировали в начале программы, тогда понятно, что было сгенерировано событие `QUIT`.

Модуль `pygame` генерирует событие `QUIT`, когда пользователь закрывает окно программы или когда компьютер выключается и пытается завершить все запущенные программы. Далее мы сообщим программе что делать, когда она обнаруживает событие `QUIT`.

Выход из программы

Если событие QUIT было сгенерировано, программа последовательно вызовет функции pygame.quit() и sys.exit().

```
62.         pygame.quit()
63.         sys.exit()
```

Функция pygame.quit() является, по сути, противоположностью init(). Вы должны вызвать ее перед выходом из программы. Если забудете, процесс IDLE может зависнуть после завершения работы вашей программы. Инструкции в строках 62 и 63 завершают работу pygame и программы.

Заключение

В этой главе мы рассмотрели множество новых тем, которые позволят нам сделать гораздо более занимательные игры. Взамен обычной работы с текстом и вызовов функций print() и input(), pygame-программа выглядит как окно (созданное методом pygame.display.set_mode()), в котором мы можем отображать разные объекты. Функции рисования модуля pygame позволяют отображать в этом окне фигуры разного цвета. Кроме того, вы можете создавать текст различного размера. Такие рисунки могут находиться в любых позициях внутри окна, в отличие от текста, создаваемого функцией print().

pygame-программы могут быть намного интереснее текстовых игр. Теперь давайте узнаем, как создавать игры с анимированной графикой.

18

АНИМИРОВАННАЯ ГРАФИКА

Теперь, когда у нас есть некоторые навыки работы с модулем `pygame`, мы напишем программу с анимацией блоков, перемещающихся в окне. У этих блоков разные цвета и размеры, а перемещаются они только по диагонали. Для создания анимации мы будем на несколько пикселей перемещать блоки при каждом переборе игрового цикла. Тогда возникнет эффект, будто блоки двигаются по экрану.

В ЭТОЙ ГЛАВЕ РАССМАТРИВАЮТСЯ СЛЕДУЮЩИЕ ТЕМЫ:

- Анимация объектов в игровом цикле
- Изменение направления движения объекта

Пример запуска игры программы

Когда вы запустите программу, ее окно будет выглядеть примерно так, как показано на рис. 18.1. Блоки будут отскакивать от границ окна.

Исходный код программы

В редакторе файлов создайте новый файл, выбрав команду меню **File** ⇒ **New File** (Файл ⇒ Новый файл). В открывшемся окне введите приведенный ниже исходный код и сохраните файл под именем *animation.py*. Затем нажмите клавишу **F5**

и запустите программу. Если при выполнении программы возникают ошибки, сравните код, который вы набрали, с оригинальным кодом с помощью онлайн-инструмента на сайте **inventwithpython.com/diff/**.

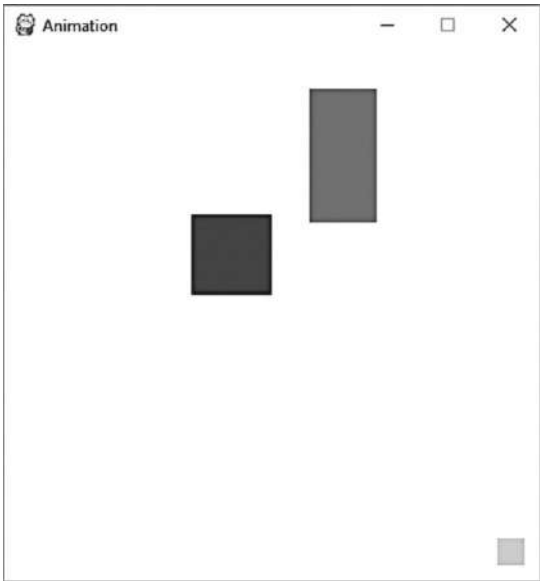


Рис. 18.1. Снимок программы с анимированными блоками

animation.py

```
1. import pygame, sys, time
2. from pygame.locals import *
3.
4. # Установка pygame.
5. pygame.init()
6.
7. # Настройка окна.
8. WINDOWWIDTH = 400
9. WINDOWHEIGHT = 400
10. windowSurface = pygame.display.set_mode((WINDOWWIDTH, WINDOWHEIGHT), 0, 32)
11. pygame.display.set_caption('Анимация')
12.
13. # Создание переменных направления.
14. DOWNLEFT = 'downleft'
15. DOWNRIGHT = 'downright'
16. UPLEFT = 'upleft'
17. UPRIGHT = 'upright'
```

```
18.
19. MOVESPEED = 4
20.
21. # Настройка цвета.
22. WHITE = (255, 255, 255)
23. RED = (255, 0, 0)
24. GREEN = (0, 255, 0)
25. BLUE = (0, 0, 255)
26.
27. # Создание структуры данных блока.
28. b1 = {'rect':pygame.Rect(300, 80, 50, 100), 'color':RED, 'dir':UPRIGHT}
29. b2 = {'rect':pygame.Rect(200, 200, 20, 20), 'color':GREEN, 'dir':UPLEFT}
30. b3 = {'rect':pygame.Rect(100, 150, 60, 60), 'color':BLUE, 'dir':DOWNLEFT}
31. boxes = [b1, b2, b3]
32.
33. # Запуск игрового цикла.
34. while True:
35.     # Проверка наличия события QUIT.
36.     for event in pygame.event.get():
37.         if event.type == QUIT:
38.             pygame.quit()
39.             sys.exit()
40.
41.     # Создание на поверхности белого фона.
42.     windowSurface.fill(WHITE)
43.
44.     for b in boxes:
45.         # Перемещение структуры данных блока.
46.         if b['dir'] == DOWNLEFT:
47.             b['rect'].left -= MOVESPEED
48.             b['rect'].top += MOVESPEED
49.         if b['dir'] == DOWNRIGHT:
50.             b['rect'].left += MOVESPEED
51.             b['rect'].top += MOVESPEED
52.         if b['dir'] == UPLEFT:
53.             b['rect'].left -= MOVESPEED
54.             b['rect'].top -= MOVESPEED
55.         if b['dir'] == UPRIGHT:
56.             b['rect'].left += MOVESPEED
57.             b['rect'].top -= MOVESPEED
58.
```

```
59.         # Проверка, переместился ли блок за пределы окна.
60.         if b['rect'].top < 0:
61.             # Прохождение блока через верхнюю границу.
62.             if b['dir'] == UPLEFT:
63.                 b['dir'] = DOWNLEFT
64.             if b['dir'] == UPRIGHT:
65.                 b['dir'] = DOWNRIGHT
66.         if b['rect'].bottom > WINDOWHEIGHT:
67.             # Прохождение блока через нижнюю границу.
68.             if b['dir'] == DOWNLEFT:
69.                 b['dir'] = UPLEFT
70.             if b['dir'] == DOWNRIGHT:
71.                 b['dir'] = UPRIGHT
72.         if b['rect'].left < 0:
73.             # Прохождение блока через левую границу.
74.             if b['dir'] == DOWNLEFT:
75.                 b['dir'] = DOWNRIGHT
76.             if b['dir'] == UPLEFT:
77.                 b['dir'] = UPRIGHT
78.         if b['rect'].right > WINDOWWIDTH:
79.             # Прохождение блока через правую границу.
80.             if b['dir'] == DOWNRIGHT:
81.                 b['dir'] = DOWNLEFT
82.             if b['dir'] == UPRIGHT:
83.                 b['dir'] = UPLEFT
84.
85.         # Создание блока на поверхности.
86.         pygame.draw.rect(windowSurface, b['color'], b['rect'])
87.
88.     # Вывод окна на экран.
89.     pygame.display.update()
90.     time.sleep(0.02)
```

Перемещение и контроль отскока блоков

В этой программе у нас будет три блока разного цвета, движущихся и отскакивающих от границ окна. В следующих главах мы используем эту программу в качестве основы для создания игры, в которой одним из блоков можно будет управлять. Для этого сначала нужно определиться с желаемым способом перемещения блоков.

Каждый блок будет двигаться в одном из четырех диагональных направлений. Когда блок сталкивается с границей окна, он должен отскакивать и двигаться в противоположном диагональном направлении. Блоки будут отскакивать так, как показано на рис. 18.2.

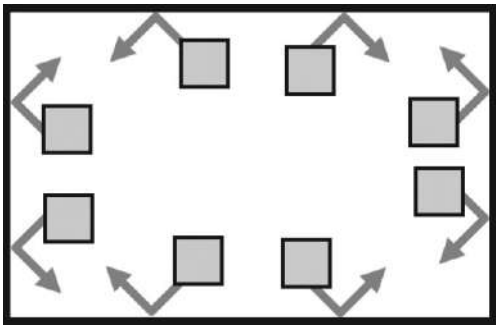


Рис. 18.2. Как будут отскакивать блоки

Новое направление, в котором движется блок после отскока, зависит от двух факторов: в каком направлении он двигался перед отскоком и от какой границы окна отскочил. Существуют восемь возможных способов отскока — по два разных способа для каждой из четырех границ. Например, если блок движется вниз и направо, а затем отскакивает от нижней границы окна, нужно, чтобы новым направлением блока было вверх и вправо.

Мы можем использовать объект `Rect` для представления положения и размера блока, кортеж из трех целых чисел для представления цвета блока и одного целого числа, представляющего, в каком из четырех диагональных направлений в данный момент двигается блок.

Игровой цикл установит координаты x и y блока в объекте `Rect` и при каждой итерации отобразит на экране все блоки в их текущем положении. По мере работы цикла блоки будут постепенно перемещаться по экрану, чтобы это выглядело, будто они плавно двигаются и отскакивают.

Создание констант

Строки с 1 по 5 устанавливают наши модули и инициализируют `pygame`, аналогично тому, что мы делали в главе 17.

```
1. import pygame, sys, time
2. from pygame.locals import *
3.
4. # Установка pygame.
5. pygame.init()
```

```
6.
7. # Настройка окна.
8. WINDOWWIDTH = 400
9. WINDOWHEIGHT = 400
10. windowSurface = pygame.display.set_mode((WINDOWWIDTH, WINDOWHEIGHT), 0, 32)
11. pygame.display.set_caption('Анимация')
```

В строках 8 и 9 мы определяем две константы для ширины и высоты окна, после чего в строке 10 используем эти константы для задания переменной `windowSurface`, которая будет представлять окно `pygame`. Код в строке 11 содержит функцию `set_caption()`, позволяющую вывести название в заголовке окна — `'Анимация'`.

В этой программе вы увидите, что значения ширины и высоты окна используются не только для вызова функции `set_mode()`. Мы будем использовать константы на случай, если вы когда-либо захотите изменить размер окна, — тогда вам нужно будет изменить код только в строках 8 и 9. Поскольку во время выполнения программы ширина и высота окна никогда не меняются, константы — неплохая мысль.

Константы для направлений

Мы будем использовать константы для каждого из четырех направлений, в которых могут перемещаться блоки.

```
13. # Создание переменных направления.
14. DOWNLEFT = 'downleft'
15. DOWNRIGHT = 'downright'
16. UPLEFT = 'upleft'
17. UPRIGHT = 'upright'
```

Для этих направлений вы могли бы использовать любое значение вместо константы. Например, можно использовать строку `'downleft'` непосредственно для представления диагонального направления вниз и влево, и повторять строку каждый раз, когда вам нужно указать это направление. Однако если вы хоть раз опечатаетесь в значении `'downleft'`, то получите ошибку, из-за которой ваша программа будет вести себя странно, даже несмотря на то что сбоя не случится.

Если вместо этого вы использовали константы и случайно ошиблись в имени переменной, Python обнаружит, что переменной с таким именем не

существует, и устроит сбой программы, выдав ошибку. Это все равно будет довольно значительным дефектом, но, по крайней мере, вы сразу узнаете об этом и сможете все исправить.

Мы также создаем константу, чтобы определить, как быстро блоки должны двигаться.

```
19. MOVESPEED = 4
```

Значение 4 константы MOVESPEED сообщает программе, на сколько пикселей должен перемещаться каждый блок при каждой итерации игрового цикла.

Константы для цвета

Код в строках 22–25 устанавливает константы для цветов. Помните, pygame использует кортежи из трех целых значений для совокупностей красного, зеленого и синего цветов, называемых моделью RGB. Целые числа находятся в диапазоне от 0 до 255.

```
21. # Настройка цвета.
22. WHITE = (255, 255, 255)
23. RED = (255, 0, 0)
24. GREEN = (0, 255, 0)
25. BLUE = (0, 0, 255)
```

Константы используются для удобства чтения кода, так же, как и в pygame-программе «Привет, мир!».

Создание структуры данных блока

Теперь определим блоки. Чтобы было проще, создадим словарь в виде структуры данных (см. раздел «Словари» в главе 9) для представления каждого перемещающегося блока. Словарь будет иметь ключи 'rect' (с объектом Rect в качестве значения), 'color' (с кортежем из трех целых чисел в качестве значения) и 'dir' (с одной из констант направления в качестве значения). Пока что мы создадим только три блока, но вы можете создать больше блоков, определив больше структур данных. При помощи кода анимации, который мы будем использовать позже, вы можете анимировать столько блоков, сколько определите при создании своих структур данных.

Переменная b1 будет хранить одну из этих структур данных блока.

```
27. # Создание структуры данных блока.
28. b1 = {'rect':pygame.Rect(300, 80, 50, 100), 'color':RED, 'dir':UPRIGHT}
```

Верхний левый угол этого блока расположен в позиции с координатами x=300 и y=80. Его ширина составляет 50 пикселей, а высота — 100 пикселей. Блок окрашен в красный (RED), а начальное направление — UPRIGHT (вправо вверх). Код в строках 29 и 30 создает еще две аналогичные структуры данных для блоков, у которых разные размеры, положения, цвета и направления.

```
29. b2 = {'rect':pygame.Rect(200, 200, 20, 20), 'color':GREEN, 'dir':UPLEFT}
30. b3 = {'rect':pygame.Rect(100, 150, 60, 60), 'color':BLUE, 'dir':DOWNLEFT}
31. boxes = [b1, b2, b3]
```

Если вам понадобилось извлечь блок или значение из списка, это можно сделать с помощью индексов и ключей. Ввод значения boxes[0] обращается к структуре данных словаря в b1. Если бы мы ввели boxes[0]['color'], выражение обратилось бы к ключу 'color' в b1, поэтому значение boxes[0]['color'] равнялось бы (255, 0, 0). Вы можете ссылаться на любое из значений в любой из структур данных блока, используя переменную boxes. Три словаря b1, b2 и b3 затем сохраняются в списке в переменной boxes.

Игровой цикл

Игровой цикл управляет анимацией движущихся блоков. Эффект анимации создается за счет того, что мы показываем друг за другом изображения с небольшими различиями. В нашей анимации изображениями выступят движущиеся блоки, а небольшие различия будут состоять в положениях каждого блока. Каждый блок на изображении будет перемещаться на 4 пикселя. Изображения будут чередоваться так быстро, что создастся впечатление плавного перемещения по экрану. Если блок столкнется с границей окна, тогда игровой цикл заставит его отскочить, изменив направление.

Теперь, когда мы немного знаем о том, как будет работать игровой цикл, давайте же его напишем!

Обработка решения игрока завершить игру

Когда игрок закрывает окно и завершает работу, мы должны прекратить выполнение программы так же, как мы делали это с pygame-программой «Привет, мир!». Нам нужно осуществить это в игровом цикле, так что наша про-

грамма постоянно проверяет, произошло ли событие QUIT. Код в строке 34 запускает цикл, а код в строках 36–39 руководит процессом выхода.

```
33. # Запуск игрового цикла.
34. while True:
35.     # Проверка наличия события QUIT.
36.     for event in pygame.event.get():
37.         if event.type == QUIT:
38.             pygame.quit()
39.             sys.exit()
```

После этого следует убедиться, что windowSurface уже можно отображать. Позже мы отобразим каждый блок на windowSurface с помощью метода rect(). При каждой итерации игрового цикла код «перерисовывает» все содержимое окна, заменяя на нем блоки на новые на расстоянии в несколько пикселей от старых. При этом мы не отображаем заново весь объект Surface; взамен мы просто добавляем отображение объекта Rect на windowSurface. Но когда игровой цикл повторяется, «перерисовывая» каждый объект Rect, он не стирает старую версию Rect. Если мы просто позволим циклу отображать на экране объекты Rect, то получим их след вместо гладкой анимации. Чтобы этого избежать, необходимо очищать окно при каждой итерации игрового цикла.

Для этого код в строке 42 заполняет Surface белым цветом, чтобы все, что было отображено там, стерлось.

```
41.     # Создание на поверхности белого фона.
42.     windowSurface.fill(WHITE)
```

Без вызова windowSurface.fill(WHITE) для заливки белым всего окна перед отображением прямоугольников в новых положениях, вы увидите след объектов Rect. Если хотите испробовать это и посмотреть, что произойдет, можете закомментировать строку 42, указав символ # в ее начале.

Как только windowSurface заполняется, можно приступать к отображению всех наших объектов Rect.

Перемещение каждого блока

Чтобы обновлять положение каждого блока, мы будем перебирать список boxes внутри игрового цикла.

```
44.     for b in boxes:
```

Чтобы упростить код для ввода, внутри цикла `for` вы будете ссылаться на текущий блок как на `b`. Нам нужно изменить каждый блок в зависимости от направления, в котором он уже перемещается, поэтому мы будем использовать инструкции `if`, чтобы выяснить направление движения блока, проверяя ключ `dir` внутри структуры данных блока. Затем мы изменим положение блока в зависимости от этого направления.

```
45.         # Перемещение структуры данных блока.
46.         if b['dir'] == DOWNLEFT:
47.             b['rect'].left -= MOVESPEED
48.             b['rect'].top += MOVESPEED
49.         if b['dir'] == DOWNRIGHT:
50.             b['rect'].left += MOVESPEED
51.             b['rect'].top += MOVESPEED
52.         if b['dir'] == UPLEFT:
53.             b['rect'].left -= MOVESPEED
54.             b['rect'].top -= MOVESPEED
55.         if b['dir'] == UPRIGHT:
56.             b['rect'].left += MOVESPEED
57.             b['rect'].top -= MOVESPEED
```

Новое значение атрибутов `left` и `top` каждого блока зависит от направления его движения. Если это направление `DOWNLEFT` или `DOWNRIGHT`, нужно *увеличить* значение атрибута `top`. Если это направление `UPLEFT` или `UPRIGHT`, нужно *уменьшить* значение атрибута `top`.

Если направление блока — `DOWNRIGHT` или `UPRIGHT`, нужно увеличить значение атрибута `left`. А если `DOWNLEFT` или `UPLEFT` — нужно уменьшить значение атрибута `left`.

Значения этих атрибутов будут увеличиваться или уменьшаться на величину целого числа, хранящегося в переменной `MOVESPEED`, то есть там, где хранится количество пикселей, на которые перемещаются блоки при каждой итерации игрового цикла. Мы присваиваем переменной `MOVESPEED` значение в строке 19.

Например, если значение `b['dir']` установлено равным `'downleft'`, `b['rect'].left` — равным `40` и `b['rect'].top` — `100`, тогда условие в строке 46 будет истинным. Если переменной `MOVESPEED` присвоено значение `4`, тогда код в строках 47 и 48 изменит объект `Rect` так, что `b['rect'].left` станет `36`, а `b['rect'].top` — `104`. Затем изменение значения `Rect` приводит к тому, что код в строке 86 «опускает» прямоугольник немного вниз и влево от его предыдущего положения.

Управление отскакиванием блока

После того как код в строках 44–57 переместил блок, нужно проверить, зашел ли он за границу окна. Если так и есть, нужно заставить блок отскочить. В контексте кода это значит, что цикл `for` присвоит новое значение ключу блока `'dir'`. Блок будет двигаться в новом направлении при следующей итерации игрового цикла. Благодаря этому кажется, что блок отскочил от границы окна.

В инструкции `if` в строке 60 мы определяем то, что блок переместился за верхнюю границу окна, если верхний атрибут объекта `Rect` блока меньше `0`.

```
59.         # Проверка, переместился ли блок за пределы окна.
60.         if b['rect'].top < 0:
61.             # Прохождение блока через верхнюю границу.
62.             if b['dir'] == UPLEFT:
63.                 b['dir'] = DOWNLEFT
64.             if b['dir'] == UPRIGHT:
65.                 b['dir'] = DOWNRIGHT
```

В таком случае направление будет изменено в зависимости от направления движения блока. Если блок перемещался в направлении UPLEFT (*влево вверх*), то теперь он будет перемещаться DOWNLEFT (*влево вниз*); если он перемещался UPRIGHT (*вправо вверх*), теперь он будет перемещаться DOWNRIGHT (*вправо вниз*).

Код в строках 66–71 занимается ситуацией, в которой блок зашел за нижнюю границу окна.

```
66.         if b['rect'].bottom > WINDOWHEIGHT:
67.             # Прохождение блока через нижнюю границу.
68.             if b['dir'] == DOWNLEFT:
69.                 b['dir'] = UPLEFT
70.             if b['dir'] == DOWNRIGHT:
71.                 b['dir'] = UPRIGHT
```

Эти строки проверяют, является ли значение атрибута `bottom` (не атрибута `top`) *больше* значения переменной `WINDOWHEIGHT`. Помните, что координаты *y* начинаются с `0` в верхней части окна и увеличиваются до значения переменной `WINDOWHEIGHT` в самом низу.

Код в строках 72–83 управляют поведением блоков, когда те отскакивают от границ.

```
72.         if b['rect'].left < 0:
73.             # Прохождение блока через левую границу.
74.             if b['dir'] == DOWNLEFT:
75.                 b['dir'] = DOWNRIGHT
76.             if b['dir'] == UPLEFT:
77.                 b['dir'] = UPRIGHT
78.         if b['rect'].right > WINDOWWIDTH:
79.             # Прохождение блока через правую границу.
80.             if b['dir'] == DOWNRIGHT:
81.                 b['dir'] = DOWNLEFT
82.             if b['dir'] == UPRIGHT:
83.                 b['dir'] = UPLEFT
```

Код в строках 78–83 аналогичен коду в строках 72–77, но проверьте, зашла ли правая сторона блока за правую границу окна. Помните, что координаты *x* начинаются с `0` на левой границе окна и увеличиваются до значения переменной `WINDOWWIDTH` на правой границе.

Отображение в окне блоков в новых положениях

При каждом перемещении блоков мы должны отображать их в новых положениях на `windowSurface`, вызывая функцию `pygame.draw.rect()`.

```
85.         # Создание блока на поверхности.
86.         pygame.draw.rect(windowSurface, b['color'], b['rect'])
```

Вам нужно передать объект `windowSurface` функции, потому что прямоугольник отображается на объекте `Surface`. Передайте функции `b['color']`, поскольку это цвет прямоугольника. Наконец, передайте `b['rect']`, так как это объект `Rect`, содержащий положение и размер отображаемого прямоугольника.

Строка 86 — последняя в цикле `for`.

Отображение окна на экране

После завершения цикла `for`, каждый блок в списке `boxes` будет обрисован.

Теперь нам нужно вызвать функцию `pygame.display.update()`, чтобы отобразить на экране `windowSurface`.

```
88.     # Вывод окна на экран.
89.     pygame.display.update()
90.     time.sleep(0.02)
```

Компьютер может перемещать и отображать блоки, и заставлять их отскакивать настолько быстро, что если бы программа работала на полной скорости, все блоки слились бы в пятно. Чтобы замедлить выполнение программы до скорости, на которой можно различать блоки, нужно добавить код `time.sleep(0.02)`. Вы можете закомментировать строку кода `time.sleep(0.02)` и запустить программу, чтобы увидеть, как она работает без нее. Вызов функции `time.sleep()` приостанавливает программу на 0,02 с (то есть 20 мс) между каждым перемещением блоков.

После этой строки интерпретатор возвращается к началу игрового цикла и снова начинает весь процесс. Таким образом, блоки постоянно перемещаются на небольшие расстояния, отскакивают от границ окна и отображаются на экране в новых положениях.

Заключение

В этой главе был представлен совершенно новый способ создания компьютерных программ.

Программы из предыдущих глав останавливались и дожидались, пока игрок введет текст. Однако в случае с нашей «анимационной» программой происходит постоянное обновление структуры данных без ожидания ввода от игрока.

Помните, в играх «Виселица» и «Крестики-нолики» у нас были структуры данных, которые представляли состояние игрового поля. Те структуры данных передавались функции `drawBoard()` и отображались на экране, что схоже с программой из этой главы. Переменная `boxes` содержит список структур данных, представляющих блоки, которые отображаются внутри игрового цикла.

Но как при отсутствии вызовов `input()` получить данные ввода от игрока? В главе 19 мы рассмотрим принцип, благодаря которому программы узнают, когда игрок нажимает клавиши на клавиатуре. Мы также изучим новую концепцию, называемую обнаружением столкновения.

19

ОБНАРУЖЕНИЕ СТОЛКНОВЕНИЙ

Обнаружение столкновений подразумевает вычисление момента касания (то есть столкновения) двух объектов друг с другом. Это часто может потребоваться в играх. Например, если игрок касается врага, он может терять здоровье, а касаясь монеты, должен автоматически ее подбирать. Обнаружение столкновений помогает определить, стоит ли персонаж в игре на твердой почве, или же под ним пустое пространство, и ему следует упасть.

В наших играх обнаружение столкновений будет использоваться для определения, накладываются ли два прямоугольника друг на друга. Такой базовый метод будет охватывать пример программы из данной главы. Мы также рассмотрим, как наши программы, использующие модуль `pygame`, могут принимать от игрока входные данные посредством клавиатуры и мыши. Это немного сложнее, чем вызов функции `input()`, который мы осуществляли в наших текстовых программах. Но использование клавиатуры является гораздо более интерактивным процессом в программах с графическим интерфейсом (GUI), а применение мыши в ранее описанных текстовых играх вообще невозможно. Эти две концепции сделают ваши игры более захватывающими!

В ЭТОЙ ГЛАВЕ РАССМАТРИВАЮТСЯ СЛЕДУЮЩИЕ ТЕМЫ:

- Объекты `clock`
- Ввод с клавиатуры в `pygame`
- Ввод мышью в `pygame`
- Обнаружение столкновений
- Перебор элементов списка без его изменения

Пример работы программы

В этой программе игрок использует клавиши ←, ↑, ↓ и → на клавиатуре, чтобы перемещать по экрану черный блок. На экране также появляются маленькие зеленые блоки, представляющие собой «еду», которые блок «съедает», когда касается их. Чтобы создать новые блоки «еды», игрок может щелкнуть мышью в любом месте окна. Кроме того, клавиша **Esc** завершает работу программы, а клавиша **X** перемещает игрока в случайное место на экране.

На рис. 19.1 показано, как будет выглядеть запущенная программа.

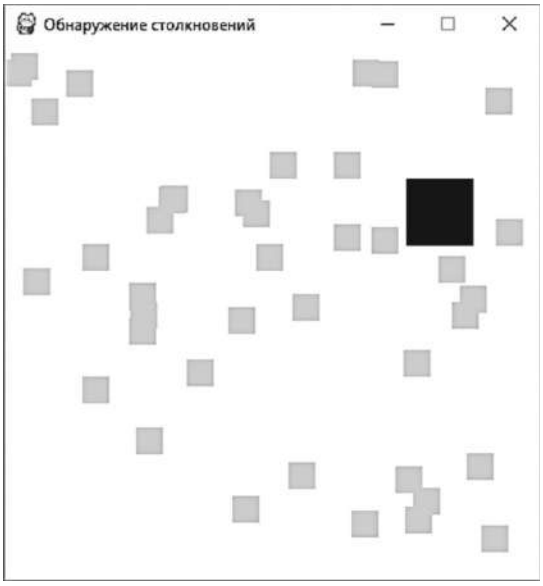


Рис. 19.1. Снимок `pygame`-программы с функцией обнаружения столкновений

Исходный код программы

В редакторе файлов создайте новый файл, выбрав команду меню **File** ⇒ **New File** (Файл ⇒ Новый файл). В открывшемся окне введите приведенный ниже исходный код и сохраните файл под именем *collisionDetection.py*. Затем нажмите клавишу **F5** и запустите программу. Если при выполнении программы возникают ошибки, сравните код, который вы набрали, с оригинальным кодом с помощью онлайн-инструмента на сайте **inventwithpython.com/diff/**.

```
1. import pygame, sys, random
2. from pygame.locals import *
3.
4. # Установка pygame.
5. pygame.init()
6. mainClock = pygame.time.Clock()
7.
8. # Настройка окна.
9. WINDOWWIDTH = 400
10. WINDOWHEIGHT = 400
11. windowSurface = pygame.display.set_mode((WINDOWWIDTH, WINDOWHEIGHT), 0, 32)
12. pygame.display.set_caption('Обнаружение столкновений')
13.
14. # Настройка цветов.
15. BLACK = (0, 0, 0)
16. GREEN = (0, 255, 0)
17. WHITE = (255, 255, 255)
18.
19. # Создание структур данных игрока и "еды".
20. foodCounter = 0
21. NEWFOOD = 40
22. FOODSIZE = 20
23. player = pygame.Rect(300, 100, 50, 50)
24. foods = []
25. for i in range(20):
26.   foods.append(pygame.Rect(random.randint(0, WINDOWWIDTH - FOODSIZE), random.randint(0, WINDOWHEIGHT - FOODSIZE), FOODSIZE, FOODSIZE))
27.
28. # Создание переменных перемещения.
29. moveLeft = False
30. moveRight = False
31. moveUp = False
32. moveDown = False
33.
34. MOVESPEED = 6
35.
36.
37. # Запуск игрового цикла.
38. while True:
39.     # Проверка событий.
```

```
40.     for event in pygame.event.get():
41.         if event.type == QUIT:
42.             pygame.quit()
43.             sys.exit()
44.         if event.type == KEYDOWN:
45.             # Изменение переменных клавиатуры.
46.             if event.key == K_LEFT or event.key == K_a:
47.                 moveRight = False
48.                 moveLeft = True
49.             if event.key == K_RIGHT or event.key == K_d:
50.                 moveLeft = False
51.                 moveRight = True
52.             if event.key == K_UP or event.key == K_w:
53.                 moveDown = False
54.                 moveUp = True
55.             if event.key == K_DOWN or event.key == K_s:
56.                 moveUp = False
57.                 moveDown = True
58.         if event.type == KEYUP:
59.             if event.key == K_ESCAPE:
60.                 pygame.quit()
61.                 sys.exit()
62.             if event.key == K_LEFT or event.key == K_a:
63.                 moveLeft = False
64.             if event.key == K_RIGHT or event.key == K_d:
65.                 moveRight = False
66.             if event.key == K_UP or event.key == K_w:
67.                 moveUp = False
68.             if event.key == K_DOWN or event.key == K_s:
69.                 moveDown = False
70.             if event.key == K_x:
71.                 player.top = random.randint(0, WINDOWHEIGHT - player.height)
72.                 player.left = random.randint(0, WINDOWWIDTH - player.width)
73.
74.         if event.type == MOUSEBUTTONUP:
75.             foods.append(pygame.Rect(event.pos[0], event.pos[1], FOODSIZE, FOODSIZE))
76.
77.     foodCounter += 1
78.     if foodCounter >= NEWFOOD:
79.         # Добавление новой "еды".
80.         foodCounter = 0
```

```
81.     foods.append(pygame.Rect(random.randint(0, WINDOWWIDTH - FOODSIZE), random.randint(0, WINDOWHEIGHT - FOODSIZE), FOODSIZE, FOODSIZE))
82.
83.     # Создание на поверхности белого фона.
84.     windowSurface.fill(WHITE)
85.
86.     # Перемещение игрока.
87.     if moveDown and player.bottom < WINDOWHEIGHT:
88.         player.top += MOVESPEED
89.     if moveUp and player.top > 0:
90.         player.top -= MOVESPEED
91.     if moveLeft and player.left > 0:
92.         player.left -= MOVESPEED
93.     if moveRight and player.right < WINDOWWIDTH:
94.         player.right += MOVESPEED
95.
96.     # Отображение игрока на поверхности.
97.     pygame.draw.rect(windowSurface, BLACK, player)
98.
99.     # Проверка, не пересекся ли игрок с какими-либо блоками "еды".
100.     for food in foods[:]:
101.         if player.colliderect(food):
102.             foods.remove(food)
103.
104.     # Отображение "еды".
105.     for i in range(len(foods)):
106.         pygame.draw.rect(windowSurface, GREEN, foods[i])
107.
108.     # Вывод окна на экран.
109.     pygame.display.update()
110.     mainClock.tick(40 )
```

Импорт модулей

Pygame-программа с функцией обнаружения столкновений импортирует те же модули, что и программа из главы 18, а также модуль random.

```
1. import pygame, sys, random
2. from pygame.locals import *
```

Использование объекта Clock для управления скоростью работы программы

Инструкции в строках 5–17 в основном выполняют те же действия, что и в программе из главы 18: они инициализируют pygame, устанавливают переменные WINDOWHEIGHT и WINDOWWIDTH и назначают константы цвета и направления.

Однако код в строке 6 вам незнаком.

```
6. mainClock = pygame.time.Clock()
```

В программе из главы 18 вызов функции time.sleep(0.02) замедлял работу программы, чтобы она не выполнялась слишком быстро. В то время как этот вызов приостанавит работу на 0,02 секунды на всех компьютерах, скорость выполнения остальной части программы зависит от скорости работы самого компьютера. Если мы хотим, чтобы наша программа работала на любом компьютере с одинаковой скоростью, нам понадобится функция, делающая на быстрых компьютерах более продолжительную паузу, чем на медленных.

Объект pygame.time.Clock может приостановить работу на необходимое количество времени на любом компьютере. Код в строке 110 вызывает функцию mainClock.tick(40) внутри игрового цикла. Этот вызов метода tick() объекта Clock ждет достаточно времени, чтобы цикл выполнялся со скоростью около 40 итераций в секунду, независимо от скорости компьютера, вследствие чего игра никогда не будет работать быстрее, чем вы ожидаете. Вызов tick() должен быть осуществлен в игровом цикле только один раз.

Настройка окна и структур данных

Код в строках 19 — 22 создают несколько переменных для блоков «еды», появляющихся на экране.

```
19. # Создание структур данных игрока и "еды".
20. foodCounter = 0
21. NEWFOOD = 40
22. FOODSIZE = 20
```

Переменная foodCounter начнет со значения 0, NEWFOOD — с 40, а переменная FOODSIZE — со значения 20. Мы увидим, как они будут использоваться позже, когда создадим «еду».

Код в строке 23 создает объект `pygame.Rect` для обозначения местоположения игрока.

```
23. player = pygame.Rect(300, 100, 50, 50)
```

Переменная `player` содержит объект `pygame.Rect`, который представляет размер и положение блока. Блок игрока будет перемещаться так же, как блоки в программе из главы 18 (см. раздел «Перемещение каждого блока» в главе 18), но в этой программе игрок сможет управлять его перемещением.

Затем мы создали код для отслеживания блоков «еды».

```
24. foods = []
25. for i in range(20):
26.   foods.append(pygame.Rect(random.randint(0, WINDOWWIDTH - FOODSIZE), random.randint(0, WINDOWHEIGHT - FOODSIZE), FOODSIZE, FOODSIZE))
```

Программа будет отслеживать каждый блок «еды» с помощью списка объектов `Rect` в `foods`. Код в строках 25 и 26 создает 20 блоков «еды», случайно разбросанных по экрану. Можно использовать функцию `random.randint()` для генерации случайных координат *x* и *y*.

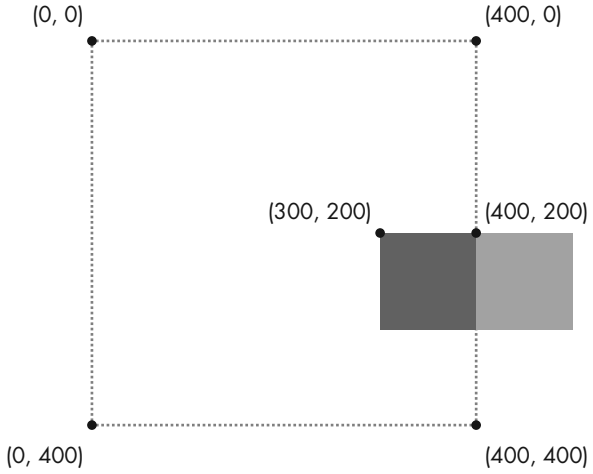


Рис. 19.2. Установление значения 400 координаты верхнего левого угла блока размером 100×100 в окне размером 400×400 поместит блок за пределы окна. Чтобы этого не произошло, координата левого угла должна равняться 300

В строке 26 программа вызывает функцию-конструктор `pygame.Rect()`, чтобы вернуть новый объект `pygame.Rect`. Он будет представлять положение и размер нового блока «еды». Первые два параметра `pygame.Rect()` — это ко-

ординаты *x* и *y* верхнего левого угла. Нужно, чтобы случайная координата находилась между 0 и размером окна минус размер блока «еды». Если вы установите значение случайной координаты между 0 и размером окна, то блок «еды» может быть целиком вытолкнут за пределы окна, как на рис. 19.2.

Третий и четвертый параметры для pygame.Rect() — ширина и высота блока «еды». Ширина и высота являются значениями в константе FOODSIZE.

Третий и четвертый параметры pygame.Rect() — ширина и высота блока «еды». Ширина и высота являются значениями в константе FOODSIZE.

Создание переменных для отслеживания перемещения

Начиная со строки 29, код создает определенные переменные, которые отслеживают движение блока игрока для каждого направления, в котором он может перемещаться.

```
28. # Создание переменных перемещения.
29. moveLeft = False
30. moveRight = False
31. moveUp = False
32. moveDown = False
```

Четыре переменные имеют логические значения, изначально равные False, для отслеживания того, какая клавиша со стрелкой была нажата. Например, когда игрок для перемещения блока нажимает клавишу ←, переменная moveLeft принимает значение True. Когда он отпускает эту клавишу, переменная moveLeft возвращается к значению False.

Код в строках 34–43 почти идентичен коду в предыдущих pygame-программах. Эти строки обрабатывают начало игрового цикла и ситуацию, когда игрок выходит из программы. Я пропущу объяснение данного кода, так как делал это в предыдущей главе.

Обработка событий

Модуль pygame способен генерировать события в ответ на пользовательский ввод с клавиатуры или мыши. Ниже перечислены события, которые могут быть возвращены pygame.event.get():

- **QUIT**. Генерируется, когда игрок закрывает окно.
- **KEYDOWN**. Генерируется, когда игрок нажимает клавишу. Имеет атрибут key, сообщающий, какая клавиша была нажата. Также имеет атрибут

mod, который сообщает, удерживалась ли клавиша **Shift**, **Ctrl**, **Alt** или какая-либо другая во время нажатия игровой клавиши.

- **KEYUP**. Генерируется, когда игрок отпускает клавишу. Имеет атрибуты key и mod, аналогичные тем, которые есть у KEYDOWN.
- **MOUSEMOTION**. Генерируется каждый раз, когда курсор мыши перемещается по окну. Имеет атрибут pos (сокращение от position — *положение*), который возвращает кортеж (x, y) для координат позиции в окне, где находится курсор. Атрибут rel также возвращает кортеж (x, y), но он возвращает координаты относительно последнего события MOUSEMOTION. Например, если мышь перемещается налево на 4 пикселя с (200, 200) до (196, 200), тогда rel вернет кортеж (-4, 0). Атрибут button возвращает кортеж из трех целых чисел. Первое целое число в кортеже представляет левую кнопку мыши, второе — среднюю (если таковая существует), а третье — правую кнопку. Эти целые числа будут равны 0, если при движении мыши кнопка не нажата, и 1, если кнопка нажата.
- **MOUSEBUTTONDOWN**. Генерируется при нажатии кнопки мыши внутри окна. Это событие имеет атрибут pos, который является кортежем (x, y) для координат местоположения мыши непосредственно при нажатии кнопки. Есть также атрибут button, представляющий собой целое число от 1 до 5. Он сообщает, какая кнопка мыши была нажата, как описано в таблице 19.1.
- **MOUSEBUTTONUP**. Генерируется при отпускании кнопки мыши. Имеет те же атрибуты, что и MOUSEBUTTONDOWN.

Когда генерируется событие MOUSEBUTTONDOWN, оно имеет атрибут button. Атрибут button — это значение, связанное с различными типами кнопок, которые могут быть у мыши. Например, левая кнопка имеет значение 1, а правая — значение 3. В таблице 19.1 перечислены все атрибуты button для событий мыши, но обратите внимание, что к мыши могут быть применены не все значения button, указанные здесь.

Таблица 19.1. Значения атрибутов button

Значение button	Кнопка (или колесико) мыши
1	Левая кнопка
2	Средняя кнопка
3	Правая кнопка
4	Колесо прокрутки прокручено вверх
5	Колесо прокрутки прокручено вниз

Мы будем использовать эти события, чтобы позволить игроку управлять блоком с помощью событий KEYDOWN и щелчков мышью.

Обработка события KEYDOWN

Код для обработки событий нажатия и отпускания клавиш начинается в строке 44. Он включает в себя тип события KEYDOWN.

```
44.         if event.type == KEYDOWN:
```

Если тип события — KEYDOWN, тогда объект Event содержит атрибут key, указывающий, какая клавиша была нажата. Когда игрок нажимает клавишу со стрелкой или клавишу **W**, **A**, **S** или **D** (эти клавиши расположены таким же образом, что и клавиши со стрелками, но в левой части клавиатуры), нужно, чтобы блок перемещался. Мы будем использовать инструкции if для проверки нажатой клавиши, чтобы указать, в каком направлении должен двигаться блок.

Код в строке 46 сравнивает этот атрибут key с K_LEFT и K_a, константами pygame.locals, представляющими на клавиатуре клавиши ← и **A**, соответственно. Код в строках 46–57 выполняет проверку для каждой из клавиш со стрелкой и **W**, **A**, **S** или **D**:

```
45.             # Изменение переменных клавиатуры.
46.             if event.key == K_LEFT or event.key == K_a:
47.                 moveRight = False
48.                 moveLeft = True
49.             if event.key == K_RIGHT or event.key == K_d:
50.                 moveLeft = False
51.                 moveRight = True
52.             if event.key == K_UP or event.key == K_w:
53.                 moveDown = False
54.                 moveUp = True
55.             if event.key == K_DOWN or event.key == K_s:
56.                 moveUp = False
57.                 moveDown = True
```

Когда нажата одна из этих клавиш, код сообщает Python, что нужно присвоить значение True соответствующей переменной перемещения. Python также присвоит значение False переменной перемещения для обратного направления.

Например, при нажатии клавиши ← программа выполняет код в строках 47 и 48. В этом случае Python присвоит переменной moveLeft значение True, а переменной moveRight — False (несмотря на то, что переменной moveRight уже

может быть присвоено значение `False`, Python снова сделает его `False`, просто для пущей уверенности).

В строке 46 переменная `event.key` может равняться либо `K_LEFT`, либо `K_a`. Значение `event.key` устанавливается равным тому же значению, что и `K_LEFT`, если нажата клавиша ←, или тому же значению, что и `K_a`, если нажата клавиша **A**.

Благодаря выполнению кода в строках 47 и 48, если нажатие клавиши — `K_LEFT` или `K_a`, клавиши ← и **A** выполняют одно и то же действие. Клавиши **W**, **A**, **S** и **D** используются в качестве альтернативы для изменения переменных перемещения, позволяя игроку использовать по желанию левую руку вместо правой. Оба набора клавиш изображены на рис. 19.3.

Рис. 19.3. Клавиши **W**, **A**, **S** или **D** можно запрограммировать выполнять те же операции, что выполняют клавиши ←, ↑, ↓ и →

Константы для клавиш с буквами и цифрами интуитивно понятны: константа клавиши с буквой «A» — это `K_a`, константа клавиши с буквой «B» — `K_b` и так далее. Константа клавиши с цифрой 3 — `K_3`. В таблице 19.2 перечислены часто используемые константы для других клавиш на клавиатуре.

Таблица 19.2. Константы для клавиш на клавиатуре

Константа pygame	**Клавиша на клавиатуре**
`K_LEFT`	**Стрелка влево**
`K_RIGHT`	**Стрелка вправо**
`K_UP`	**Стрелка вверх**
`K_DOWN`	**Стрелка вниз**
`K_ESCAPE`	**Esc**
`K_BACKSPACE`	**Backspace**
`K_TAB`	**Tab**
`K_RETURN`	**Return** или **Enter**
`K_SPACE`	**Пробел**
`K_DELETE`	**Del**
`K_LSHIFT`	Левая клавиша **Shift**
`K_RSHIFT`	Правая клавиша **Shift**
`K_LCTRL`	Левая клавиша **Ctrl**
`K_RCTRL`	Правая клавиша **Ctrl**

Константа pygame	Клавиша на клавиатуре
K_LALT	Левая клавиша **Alt**
K_RALT	Правая клавиша **Alt**
K_HOME	**Home**
K_END	**End**
K_PAGEUP	**PgUp**
K_PAGEDOWN	**PgDn**
K_F1	**F1**
K_F2	**F2**
K_F3	**F3**
K_F4	**F4**
K_F5	**F5**
K_F6	**F6**
K_F7	**F7**
K_F8	**F8**
K_F9	**F9**
K_F10	**F10**
K_F11	**F11**
K_F12	**F12**

Обработка события KEYUP

Когда игрок отпускает клавишу, генерируется событие KEYUP.

```
58.         if event.type == KEYUP:
```

Если клавиша, которую отпустили, была **Esc**, тогда Python должен завершить программу. Помните, что в модуле pygame вы должны вызвать функцию pygame.quit() перед вызовом функции sys.exit(), что мы и делаем в строках 59–61.

```
59.             if event.key == K_ESCAPE:
60.                 pygame.quit()
61.                 sys.exit()
```

Код в строках 62–69 присваивает переменной перемещения значение False, если была отпущена клавиша соответствующего направления.

```
62.             if event.key == K_LEFT or event.key == K_a:
63.                 moveLeft = False
64.             if event.key == K_RIGHT or event.key == K_d:
65.                 moveRight = False
66.             if event.key == K_UP or event.key == K_w:
67.                 moveUp = False
68.             if event.key == K_DOWN or event.key == K_s:
69.                 moveDown = False
```

Присваивание переменной перемещения значения `False` посредством события `KEYUP` приводит к тому, что блок прекращает перемещение.

Телепортация игрока

Вы также можете добавить в игру возможность телепортации. Если игрок нажимает клавишу **X**, код в строках 71 и 72 помещает блок игрока в случайную позицию в окне.

```
70.             if event.key == K_x:
71.                 player.top = random.randint(0, WINDOWHEIGHT - player.height)
72.                 player.left = random.randint(0, WINDOWWIDTH - player.width)
```

Код в строке 70 проверяет, нажал ли игрок клавишу **X**. Затем код в строке 71 устанавливает значение случайной координаты *x*, в которую телепортируется игрок, в промежутке между `0` и высотой окна минус высота блока игрока. Код в строке 72 выполняет аналогичный код для координаты *y*. Это позволяет игроку перемещаться по окну, нажимая клавишу **X**, но он не сможет выбирать, куда телепортироваться, — это место выбирается полностью случайным образом.

Добавление новых блоков «еды»

Существуют два способа, с помощью которых игрок может добавить на экран новые блоки «еды».

Он может щелкнуть мышью в позиции, где хочет создать новый блок «еды», или подождать, пока игровой цикл не повторит `NEWFOOD` количество раз, и тогда в окне будет случайно создан новый блок «еды».

Сначала рассмотрим, как «еда» добавляется с помощью мыши.

```
74.         if event.type == MOUSEBUTTONUP:
75.             foods.append(pygame.Rect(event.pos[0], event.pos[1], FOODSIZE, FOODSIZE))
```

Ввод с мыши управляется событиями точно так же, как ввод с клавиатуры. Событие `MOUSEBUTTONUP` происходит, когда игрок отпускает кнопку мыши после того, как нажал ее.

В строке 75 координата *x* сохраняется в переменной `event.pos[0]`, а координата *y* — в переменной `event.pos[1]`. Код в строке 75 создает новый объект `Rect` для представления нового блока «еды» и помещает его там, где произошло событие `MOUSEBUTTONUP`. При добавлении нового объекта `Rect` в список `foods`, код отображает на экране новый блок «еды».

Помимо добавления вручную на усмотрение игрока, блоки «еды» генерируются автоматически с помощью кода в строках 77–81.

```
77.     foodCounter += 1
78.     if foodCounter >= NEWFOOD:
79.         # Добавление новой "еды".
80.         foodCounter = 0
81.     foods.append(pygame.Rect(random.randint(0, WINDOWWIDTH - FOODSIZE), random.randint(0, WINDOWHEIGHT - FOODSIZE), FOODSIZE, FOODSIZE))
```

Переменная `foodCounter` отслеживает, как часто следует добавлять «еду». При каждой итерации игрового цикла значение переменной `foodCounter` в строке 77 увеличивается на `1`.

Когда значение переменной `foodCounter` становится больше или равно константе `NEWFOOD`, оно сбрасывается, и кодом в строке 81 создается новый блок «еды». Вы можете изменить скорость, с которой добавляются новые блоки «еды», путем настройки `NEWFOOD` ранее в строке 21.

Код в строке 84 просто заполняет поверхность окна белым цветом, о чем мы говорили в разделе «Обработка решения игрока завершить игру» предыдущей главы, поэтому сейчас перейдем к обсуждению того, как игрок перемещается по экрану.

Перемещение игрока по окну

Мы присвоили переменным перемещения (`moveDown`, `moveUp`, `moveLeft` и `moveRight`) значения `True` или `False` в зависимости от того, какие клавиши нажал игрок. Теперь нам нужно переместить блок игрока, а он представлен объектом `pygame.Rect`, хранящимся в переменной `player`. Мы сделаем это, установив координаты *x* и *y* объекта `player`.

```
86.     # Перемещение игрока.
87.     if moveDown and player.bottom < WINDOWHEIGHT:
88.         player.top += MOVESPEED
89.     if moveUp and player.top > 0:
90.         player.top -= MOVESPEED
91.     if moveLeft and player.left > 0:
92.         player.left -= MOVESPEED
93.     if moveRight and player.right < WINDOWWIDTH:
94.         player.right += MOVESPEED
```

Если переменной `moveDown` присвоено значение `True` (а нижняя часть блока игрока не вышла за пределы нижней границы окна), тогда код в строке 88 перемещает блок игрока вниз, добавляя значение `MOVESPEED` к текущему атрибуту игрока `top`. Код в строках 89–94 делает то же самое для трех других направлений.

Отображение блока игрока в окне

Код в строке 97 отображает блок игрока в окне.

```
96.     # Отображение игрока на поверхности.
97.     pygame.draw.rect(windowSurface, BLACK, player)
```

После перемещения блока код в строке 97 отображает его в новом положении. Переменная `WindowSurface`, переданная в качестве первого параметра, сообщает Python, на каком из объектов `Surface` отобразить прямоугольник. Переменная `BLACK`, в которой хранится значение `(0, 0, 0)`, сообщает Python, что нужно отобразить черный прямоугольник. Объект `Rect`, хранящийся в переменной `player`, указывает Python положение и размер прямоугольника, который нужно отобразить.

Проверка на столкновения

Прежде чем отображать блоки «еды», программе необходимо проверить, не пересекся ли блок игрока каким-либо из блоков «еды». Если пересекся, то этот блок «еды» следует удалить из списка `foods`. Таким образом, Python не будет отображать блоки «еды», которые игрок уже «съел».

Мы будем использовать `colliderect()` – метод обнаружения столкновений, который содержат все объекты `Rect` в строке 101.

```
99.     # Проверка, не пересекся ли игрок с какими-либо блоками "еды".
100.     for food in foods[:]:
101.         if player.colliderect(food):
102.             foods.remove(food)
```

При каждой итерации цикла `for` текущий блок «еды» из списка `foods` (множественное число) помещается в переменную `food` (единственное число). Метод `colliderect()` для объектов `pygame.Rect` передается объекту `pygame.Rect` блока игрока в качестве аргумента и возвращает `True`, если два прямоугольника сталкиваются, и `False` — если не сталкиваются. В случае значения `True` код в строке 102 удаляет перекрывающийся блок «еды» из списка `foods`.

НЕ ИЗМЕНЯЙТЕ СПИСОК ПРИ ЕГО ПЕРЕБОРЕ

Обратите внимание, что этот цикл `for` немного отличается от циклов `for`, которые мы встречали раньше. Если вы внимательно взглянете на строку 100, то заметите, что на самом деле цикл перебирает не `foods`, а `foods[:]`.

Давайте вспомним, как работают срезы. `foods[:2]` — копия списка с элементами от начала и до (но не включая) элемента с индексом `2`. `foods[:]` даст вам копию списка с элементами с самого начала и до конца. В принципе, `foods[:]` создает новый список с копией всех элементов в списке `foods`. Это более короткий способ создать копию списка, чем, скажем, тот, что основан на использовании функции `getBoardCopy()` в игре «Крестики-нолики» из главы 10.

Вы не можете добавлять или удалять элементы из списка во время перебора. Если размер списка `foods` постоянно меняется, Python может потерять информацию о том, каким должно быть следующее значение переменной `food`. Представьте, как трудно было бы подсчитать количество конфет в банке, если бы в это время кто-то их оттуда забирал или подкидывал новые.

Но если вы перебираете копию списка (а копия никогда не изменяется), добавление или удаление элементов из исходного списка не будет проблемой.

Отображение блоков «еды»

Код в строках 105 и 106 аналогичен тому, который мы использовали для отображения черного блока игрока.

```
104.     # Отображение "еды".
105.     for i in range(len(foods)):
106.         pygame.draw.rect(windowSurface, GREEN, foods[i])
```

Код в строке 105 перебирает каждый блок «еды» в списке `foods`, а код в строке 106 отображает блок «еды» на `windowSurface`.

Теперь, когда блок игрока и блоки «еды» отображены на экране, окно готово к обновлению, поэтому мы вызываем метод `update()` в строке 109 и завершаем программу, вызывая метод `tick()` в объекте `Clock`, который мы создали ранее.

```
108.     # Вывод окна на экран.
109.     pygame.display.update()
110.     mainClock.tick(40 )
```

Программа продолжит выполняться через игровой цикл и будет обновляться до тех пор, пока игрок не решит закончить игру.

Заключение

В этой главе была решена проблема обнаружения столкновений. В графических играх обнаружение столкновений между двумя прямоугольниками настолько распространено, что модуль `pygame` предоставляет свой собственный метод обнаружения столкновений для объектов `pygame.Rect` под названием `colliderect()`.

Первые несколько игр в этой книге были текстовыми. Вывод данных программы на экран имел текстовую форму, ввод данных осуществлялся игроком путем набора текста с клавиатуры. Графические программы позволяют вам принимать ввод с мыши.

Также эти программы могут реагировать на одиночные нажатия клавиш, когда игрок нажимает или отпускает ту или иную клавишу. Игроку не нужно вводить полный ответ и нажимать клавишу **Enter**. Это позволяет получать мгновенную обратную связь и делает игры гораздо более интерактивными.

Наша первая интерактивная графическая программа оказалась довольно занятной, но давайте пойдем дальше. В главе 20 вы узнаете, как с помощью `pygame` загружать изображения и воспроизводить звуковые эффекты.

20

ИСПОЛЬЗОВАНИЕ ЗВУКОВ И ИЗОБРАЖЕНИЙ

Из глав 18 и 19 вы узнали, как создавать графический интерфейс программ с поддержкой ввода данных с клавиатуры и мыши. Вы также научились отображать различные фигуры. В этой главе вы узнаете, как добавлять в свои игры звуки, музыку, а также изображения.

В ЭТОЙ ГЛАВЕ РАССМАТРИВАЮТСЯ СЛЕДУЮЩИЕ ТЕМЫ:

- Файлы звуков и изображений
- Отображение и масштабирование спрайтов
- Добавление музыки и звуков
- Включение и отключение звука

Добавление изображений с помощью спрайтов

Спрайт — это отдельное двумерное изображение, которое используется на экране как часть графики. На рис. 20.1 показаны некоторые примеры спрайтов.

Спрайты отображаются поверх фона. Вы можете отразить спрайт по горизонтали, чтобы он выглядел иначе, можете также отобразить один и тот же спрайт в одном окне несколько раз, и можете увеличивать или уменьшать размер спрайтов. Фоновое изображение тоже можно рассматривать как один большой спрайт. На рис. 20.2 показаны спрайты, используемые вместе.

Следующая программа продемонстрирует, как воспроизводить звуки и отображать спрайты с помощью `pygame`.

Рис. 20.1. Некоторые примеры спрайтов

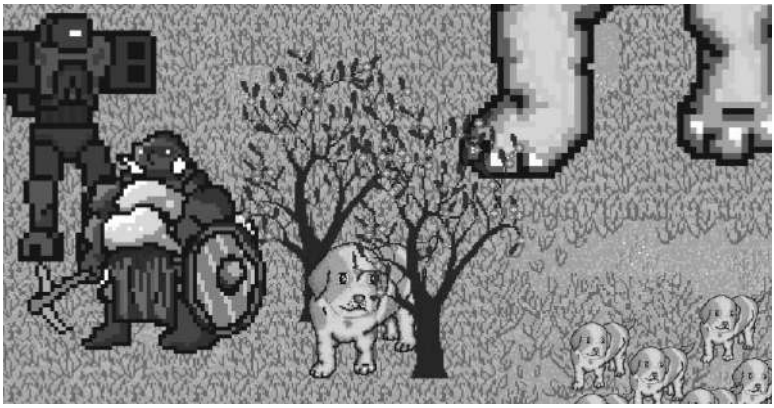


Рис. 20.2. Целая сцена со спрайтами, отображенными поверх фона

Графические и звуковые файлы

Спрайты хранятся в файлах изображений на вашем компьютере. Существует несколько форматов изображений, которые поддерживает `pygame`. Чтобы узнать формат файла изображения, посмотрите на конец его имени (после последней точки). То, что вы увидите, называется *расширением файла*. Например, файл *player.png* имеет формат PNG. Модуль `pygame` поддерживает форматы изображений BMP, PNG, JPG и GIF.

Вы можете загружать изображения из веб-браузера. В большинстве случаев это делается путем щелчка правой кнопкой мыши на изображении на веб-странице и последующего выбора пункта **Сохранить изображение** (Save Image) в появившемся меню. Запомните, куда именно на жестком диске сохранился файл изображения, потому что вам нужно будет скопировать его в ту же папку, где находится файл *.py* вашей программы Python. Вы также можете создавать свои собственные изображения с помощью графического редактора наподобие Microsoft Paint или Tux Paint.

К форматам звуковых файлов, поддерживаемых модулем `pygame`, относятся MIDI, WAV и MP3. Звуковые эффекты можно загружать из Интернета так

же, как файлы изображений, но звуковые файлы должны быть в одном из трех перечисленных форматов. Если на вашем компьютере есть микрофон, вы также можете записывать звуки и создавать собственные файлы WAV для использования в своих играх.

Пример запуска игры

Программа этой главы точно такая же, как и приложение с обнаружением столкновений из главы 19. Однако в этой программе вместо простых квадратов мы будем использовать спрайты. Для представления игрока вместо черного блока мы будем использовать спрайт человека, а спрайт вишен — вместо зеленых квадратов «еды». Мы также воспроизведем фоновую музыку и добавим звуковой эффект при «съедании» каждой из вишен спрайтом игрока.

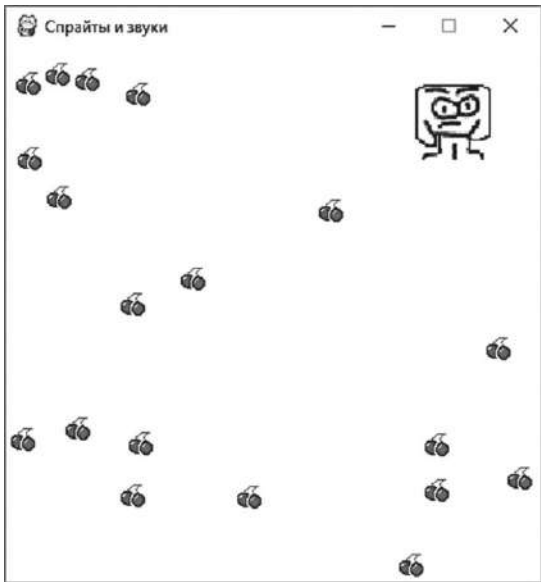


Рис. 20.3. Снимок игры «Спрайты и звуки»

В этой программе спрайт игрока будет «есть» спрайты вишен, и по мере того, как он их «ест», он будет увеличиваться. Когда вы запустите программу, игра будет выглядеть так, как показано на рис. 20.3.

Исходный код программы

В редакторе файлов создайте новый файл, выбрав команду меню **File** ⇒ **New File** (Файл ⇒ Новый файл). В открывшемся окне введите при-

веденный ниже исходный код и сохраните файл под именем *spritesAndSounds.py*. Затем нажмите клавишу **F5** и запустите программу. Если при выполнении программы возникают ошибки, сравните код, который вы набрали, с оригинальным кодом с помощью онлайн-инструмента на сайте **inventwithpython.com/diff/**.

spritesAndSounds.py

```
import pygame, sys, time, random
from pygame.locals import *

# Установка pygame.
pygame.init()
mainClock = pygame.time.Clock()

# Настройка окна.
WINDOWWIDTH = 400
WINDOWHEIGHT = 400
windowSurface = pygame.display.set_mode((WINDOWWIDTH, WINDOWHEIGHT), 0, 32)
pygame.display.set_caption('Спрайты и звуки')

# Настройка цвета.
WHITE = (255, 255, 255)

# Создание структуры данных блока.
player = pygame.Rect(300, 100, 40, 40)
playerImage = pygame.image.load('player.png')
playerStretchedImage = pygame.transform.scale(playerImage, (40, 40))
foodImage = pygame.image.load('cherry.png')
foods = []
for i in range(20):
foods.append(pygame.Rect(random.randint(0,WINDOWWIDTH-20),random.randint(0,WINDOWHEIGHT-20),20,20))

foodCounter = 0
NEWFOOD = 40

# Создание переменных клавиатуры.
moveLeft = False
moveRight = False
```

```
32. moveUp = False
33. moveDown = False
34.
35. MOVESPEED = 6
36.
37. # Настройка музыки.
38. pickUpSound = pygame.mixer.Sound('pickup.wav')
39. pygame.mixer.music.load('background.mid')
40. pygame.mixer.music.play(-1, 0.0)
41. musicPlaying = True
42.
43. # Запуск игрового цикла.
44. while True:
45.     # Проверка наличия события QUIT.
46.     for event in pygame.event.get():
47.         if event.type == QUIT:
48.             pygame.quit()
49.             sys.exit()
50.         if event.type == KEYDOWN:
51.             # Изменение переменных клавиатуры.
52.             if event.key == K_LEFT or event.key == K_a:
53.                 moveRight = False
54.                 moveLeft = True
55.             if event.key == K_RIGHT or event.key == K_d:
56.                 moveLeft = False
57.                 moveRight = True
58.             if event.key == K_UP or event.key == K_w:
59.                 moveDown = False
60.                 moveUp = True
61.             if event.key == K_DOWN or event.key == K_s:
62.                 moveUp = False
63.                 moveDown = True
64.         if event.type == KEYUP:
65.             if event.key == K_ESCAPE:
66.                 pygame.quit()
67.                 sys.exit()
68.             if event.key == K_LEFT or event.key == K_a:
69.                 moveLeft = False
70.             if event.key == K_RIGHT or event.key == K_d:
71.                 moveRight = False
72.             if event.key == K_UP or event.key == K_w:
```

```
73.                 moveUp = False
74.             if event.key == K_DOWN or event.key == K_s:
75.                 moveDown = False
76.             if event.key == K_x:
77.                 player.top = random.randint(0, WINDOWHEIGHT -             player.height)
78.                 player.left = random.randint(0, WINDOWWIDTH - player.width)
79.             if event.key == K_m:
80.                 if musicPlaying:
81.                     pygame.mixer.music.stop()
82.                 else:
83.                     pygame.mixer.music.play(-1, 0.0)
84.                 musicPlaying = not musicPlaying
85.
86.         if event.type == MOUSEBUTTONUP:
87.             foods.append(pygame.Rect(event.pos[0] - 10, event.pos[1] - 10, 20, 20))
88.
89.     foodCounter += 1
90.     if foodCounter >= NEWFOOD:
91.         # Добавление новой "еды".
92.         foodCounter = 0
93.  foods.append(pygame.Rect(random.randint(0,WINDOWWIDTH-20),random.randint(0,WINDOWHEIGHT-20),20,20))
94.
95.     # Создание на поверхности белого фона.
96.     windowSurface.fill(WHITE)
97.
98.     # Перемещение игрока.
99.     if moveDown and player.bottom < WINDOWHEIGHT:
100.         player.top += MOVESPEED
101.     if moveUp and player.top > 0:
102.         player.top -= MOVESPEED
103.     if moveLeft and player.left > 0:
104.         player.left -= MOVESPEED
105.     if moveRight and player.right < WINDOWWIDTH:
106.         player.right += MOVESPEED
107.
108.
109.     # Отображение блока на поверхности.
110.     windowSurface.blit(playerStretchedImage, player)
111.
112.     # Проверка, не пересекся ли блок с какими-либо блоками "еды".
113.     for food in foods[:]:
```

```
114.         if player.colliderect(food):
115.             foods.remove(food)
116.            player = pygame.Rect(player.left, player.top, player.width + 2, player.height + 2)
117.      playerStretchedImage = pygame.transform.scale(playerImage, (player.width, player.height))
118.             if musicPlaying:
119.                 pickUpSound.play()
120.
121.     # Отображение "еды".
122.     for food in foods:
123.         windowSurface.blit(foodImage, food)
124.
125.     # Вывод окна на экран.
126.     pygame.display.update()
127.     mainClock.tick(40 )
```

Настройка окна и создание структуры данных

Бóльшая часть кода в этой программе совпадает с кодом программы из главы 19. Мы сосредоточимся только на тех частях, которые добавляют спрайты и звуки. Сначала в строке кода 12 выведем название игры в заголовок окна.

```
12. pygame.display.set_caption('Спрайты и звуки')
```

Чтобы присвоить это название, вам нужно передать строку `'Спрайты и звуки'` функции `pygame.display.set_caption()`.

Добавление спрайта

Теперь, когда у нас есть заголовок, нам, собственно, нужны спрайты. Мы будем использовать три переменные для представления игрока, в отличие от предыдущих программ, где мы использовали только одну.

```
17. # Создание структуры данных блока.
18. player = pygame.Rect(300, 100, 40, 40)
19. playerImage = pygame.image.load('player.png')
20. playerStretchedImage = pygame.transform.scale(playerImage, (40, 40))
21. foodImage = pygame.image.load('cherry.png')
```

Переменная `player` в строке 18 будет хранить объект `Rect`, который отслеживает положение и размер спрайта игрока. Переменная `player` не содержит изображение спрайта игрока. В начале программы верхний левый угол спрайта игрока расположен в (300, 100), его начальные высота и ширина составляют 40 пикселей.

Вторая переменная, представляющая спрайт игрока, — `playerImage` в строке кода 19. Функции `pygame.image.load()` передается строка с именем файла загружаемого изображения. Возвращаемое значение представляет собой объект `Surface`, который содержит графику в файле изображения, отображаемого на его поверхности. Мы сохраняем этот объект `Surface` внутри `playerImage`.

Изменение размера спрайта

В строке кода 20 мы будем использовать новую функцию в модуле `pygame.transform`. Функция `pygame.transform.scale()` позволяет сжимать/расширять спрайт.

Первый аргумент — это объект `Surface` с изображением, отображенным на нем.

Второй аргумент — кортеж для новой ширины и высоты изображения в первом аргументе. Функция `scale()` возвращает объект `Surface` с отображенным изображением нового размера.

В программе этой главы мы сделаем так, чтобы спрайт игрока растягивался больше по мере «поедания» вишен. Мы сохраним исходное изображение в переменной `playerImage`, а растянутое изображение — в переменной `playerStretchedImage`.

В строке 21 мы снова вызываем функцию `load()`, чтобы создать объект `Surface` с изображением вишни, отображенном на нем. Убедитесь, что файлы *player.png* и *cherry.png* находятся в той же папке, что и файл *spritesAndSounds.py*, иначе модуль `pygame` не сможет их найти и выдаст ошибку.

Установка музыки и звуков

Теперь вам нужно загрузить звуковые файлы. В `pygame` есть два модуля для звука. Модуль `pygame.mixer` может воспроизводить короткие звуковые эффекты во время игры. Модуль `pygame.mixer.music` воспроизводит фоновую музыку.

Добавление аудиофайлов

Вызовите функцию-конструктор `pygame.mixer.Sound()`, чтобы создать объект `pygame.mixer.Sound` (для краткости назовем объектом `Sound`). Этот объект

имеет метод play(), который будет воспроизводить звуковой эффект при вызове.

```
37. # Настройка музыки.
38. pickUpSound = pygame.mixer.Sound('pickup.wav')
39. pygame.mixer.music.load('background.mid')
40. pygame.mixer.music.play(-1, 0.0)
41. musicPlaying = True
```

Код в строке 39 вызывает функцию pygame.mixer.music.load() для загрузки фоновой музыки, а код в строке 40 — функцию pygame.mixer.music.play(), чтобы начать воспроизведение этой музыки. Первый параметр сообщает pygame, сколько раз проигрывать фоновую музыку после первого запуска. Таким образом, передача значения 5 заставит pygame проигрывать фоновую музыку шесть раз. Здесь мы передаем параметр -1 — это специальное значение, которое заставляет фоновую музыку повторяться бесконечно.

Второй параметр play() — это момент в звуковом файле, с которого нужно начать воспроизведение. Передача значения 0,0 начнет воспроизведение фоновой музыки с самого начала. Передача значения 2,5 начнет воспроизведение с момента в 2,5 секунды после начала.

Наконец, переменная musicPlaying имеет логическое значение, которое сообщает программе, нужно воспроизводить фоновую музыку и звуковые эффекты или нет. Было бы любезно предоставить игроку возможность запускать программу без звука.

Включение/отключение звука

Клавиша **M** включает или отключает фоновую музыку. Если переменной musicPlaying присвоено значение True, тогда фоновая музыка сейчас воспроизводится, и чтобы ее остановить, нужно вызвать функцию pygame.mixer.music.stop(). Если переменной musicPlaying присвоено значение False, тогда фоновая музыка сейчас не воспроизводится, и мы должны запустить ее, вызвав функцию play(). Для этого в строках 79–84 используются инструкции if.

```
79.             if event.key == K_m:
80.                 if musicPlaying:
81.                     pygame.mixer.music.stop()
82.                 else:
```

```
83.                     pygame.mixer.music.play(-1, 0.0)
84.                 musicPlaying = not musicPlaying
```

Независимо от того, играет ли музыка или нет, нам необходимо переключить значение musicPlaying.

Переключение логического значения означает присвоение ему значения, противоположного текущему.

Строка кода musicPlaying = not musicPlaying присваивает переменной значение False, если в настоящее время она имеет значение True — или присваивает True, если — False.

Представьте переключение как процесс включения или выключения света: переключение выключателя меняет его состояние на противоположное.

Отображение спрайта игрока в окне

Помните, что значение, хранящееся в переменной playerStretchedImage, является объектом Surface. Код в строке 110 отображает спрайт игрока на объекте Surface окна (хранящемся в переменной windowSurface), используя blit().

```
109.     # Отображение блока на поверхности.
110.     windowSurface.blit(playerStretchedImage, player)
```

Второй параметр метода blit() — это объект Rect, который указывает, где на объекте Surface должен быть отображен спрайт. Программа использует объект Rect, хранящийся в переменной player и отслеживающий положение спрайта игрока в окне.

Проверка на столкновения

Этот код похож на предыдущие программы, но здесь есть несколько новых строк.

```
114.         if player.colliderect(food):
115.             foods.remove(food)
116.             player = pygame.Rect(player.left, player.top, player.width + 2, player.height +
2)
```

```
117.     playerStretchedImage = pygame.transform.scale(playerImage, (player.width, player.height))
118.             if musicPlaying:
119.                 pickUpSound.play()
```

Когда спрайт игрока «съедает» одну из вишен, его размер увеличивается на два пикселя по высоте и ширине. В строке 116 новый объект `Rect` на два пикселя больше старого объекта `Rect`, будет назначен в качестве нового значения переменной `player`.

В то время как объект `Rect` представляет положение и размер игрока, изображение игрока сохраняется в `playerStretchedImage` как объект `Surface`. В строке 117 программа создает новое растянутое изображение, вызывая функцию `scale()`.

Растяжение изображения часто немного искажает его. Если вы продолжите растягивать уже растянутое изображение, то быстро получите искажения. Но, каждый раз растягивая исходное изображение — путем передачи функции `scale()` в качестве первого аргумента `playerImage`, не `playerStretchedImage`, — вы искажаете изображение только один раз.

Наконец, код в строке 119 вызывает метод `play()` в объекте `Sound`, сохраненном в переменной `pickUpSound`. Но вызов осуществляется только если переменной `musicPlaying` присвоено значение `True` (это означает, что звук включен).

Отображение спрайтов вишен в окне

В предыдущих программах вы вызывали функцию `pygame.draw.rect()`, чтобы отобразить зеленый квадрат для каждого объекта `Rect`, сохраненного в списке `foods`.

Однако в данной программе вам вместо этого нужно отобразить спрайты вишен.

Вызовите метод `blit()` и передайте объект `Surface`, хранящийся в переменной `foodImage`, на котором отображено изображение вишен.

```
121.     # Отображение "еды".
122.     for food in foods:
123.         windowSurface.blit(foodImage, food)
```

Переменная `food`, содержащая каждый из объектов `Rect` в списке `foods` при каждой итерации цикла `for`, сообщает методу `blit()`, где нужно отображать `foodImage`.

Заключение

Вы добавили в свою игру изображения и звук. Изображения, называемые спрайтами, выглядят намного лучше, чем простые рисунки, которые использовались в предыдущих программах. Спрайты можно масштабировать (то есть растягивать), увеличивая или уменьшая их размер, поэтому мы можем отображать спрайты любого желаемого размера. В игре, представленной в этой главе, также есть фон и воспроизводятся звуковые эффекты.

Теперь, когда мы знаем, как создавать окно, отображать спрайты и геометрические примитивы, получать ввод с клавиатуры и мыши, воспроизводить звуки и осуществлять обнаружение столкновений, мы готовы создать настоящую графическую игру в `pygame`. Глава 21 объединяет все эти элементы для создания нашей самой продвинутой игры.

21

ИГРА «ЛОВКАЧ» С ГРАФИКОЙ И ЗВУКОМ

Предыдущие четыре главы рассказали вам о модуле `pygame` и научили вас использовать многие его функции. В этой главе мы будем использовать эти знания для создания графической игры под названием «Ловкач».

В ЭТОЙ ГЛАВЕ РАССМАТРИВАЮТСЯ СЛЕДУЮЩИЕ ТЕМЫ:

- Флаг `pygame.FULLSCREEN`
- Метод `Rect.move_ip()`
- Добавление чит-кодов
- Изменение игры «Ловкач»

В игре «Ловкач» игрок управляет спрайтом (своим персонажем), который должен уклоняться от целой кучи злодеев, падающих с верхней части экрана. Чем дольше игрок сможет уклоняться от злодеев, тем больше очков он получит.

Шутки ради мы также добавим в эту игру некоторые чит-режимы. Если игрок удерживает клавишу **X**, скорость каждого злодея становится сверхнизкой. Если игрок удерживает клавишу **Z**, злодеи будут менять направление на противоположное, перемещаясь по экрану снизу вверх.

Обзор основных типов данных `pygame`

Прежде чем мы начнем работу над игрой «Ловкач», давайте рассмотрим некоторые основные типы данных, использующиеся в `pygame`:

- **pygame.Rect**
  Объекты Rect представляют местоположение и размер прямоугольной области. Местоположение определяется атрибутом topleft объекта Rect (или его атрибутами topright, bottomleft и bottomright). Эти угловые атрибуты — кортежи целых чисел для обозначения координат *x* и *y*. Размер определяется атрибутами width и height, которые являются целыми числами, указывающими количество пикселей, содержащихся в прямоугольнике по ширине и высоте. Объекты Rect поддерживают метод colliderect(), проверяющий, сталкиваются ли они с другими объектами Rect.
- **pygame.Surface**
  Объекты Surface представляют области цветных пикселей. Объект Surface представляет изображение прямоугольника, в то время как объект Rect представляет только прямоугольную область и ее расположение. Объекты Surface имеют метод blit(), который используется для вывода изображения одного объекта Surface на другой такой объект. Объект Surface, который возвращается функцией pygame.display.set_mode(), — особый, так как все отображенное на этом объекте, отображается на экране пользователя, когда вызывается функция pygame.display.update().
- **pygame.event.Event**
  Модуль pygame.event генерирует объекты Event каждый раз, когда пользователь осуществляет ввод с клавиатуры, мыши или ввод иного рода. Функция pygame.event.get() возвращает список этих объектов Event. Вы можете определить тип объекта Event, проверив его атрибут type. Примерами некоторых типов событий являются QUIT, KEYDOWN и MOUSEBUTTONUP. (См. раздел «Обработка событий» в главе 19, в котором указан полный список всех типов событий.)
- **pygame.font.Font**
  Модуль pygame.font использует тип данных Font, который представляет шрифт, используемый для оформления текста в pygame. Аргументы, передающиеся pygame.font.SysFont() — строка с названием шрифта (обычно в качестве названия шрифта передается None, чтобы использовать шрифт по умолчанию) и целое число размера шрифта.
- **pygame.time.Clock**
  Объект Clock в модуле pygame.time пригодится для того, чтобы игры работали на нормальной скорости, а не на очень высокой. Объект Clock имеет метод tick(), которому может быть передано значение количества кадров в секунду (FPS, *frames per second*), с которым требуется запустить игру. Чем выше значение FPS, тем быстрее работает игра.

Пример запуска игры «Ловкач»

Когда вы запустите эту программу, окно будет выглядеть как показано на рис. 21.1.

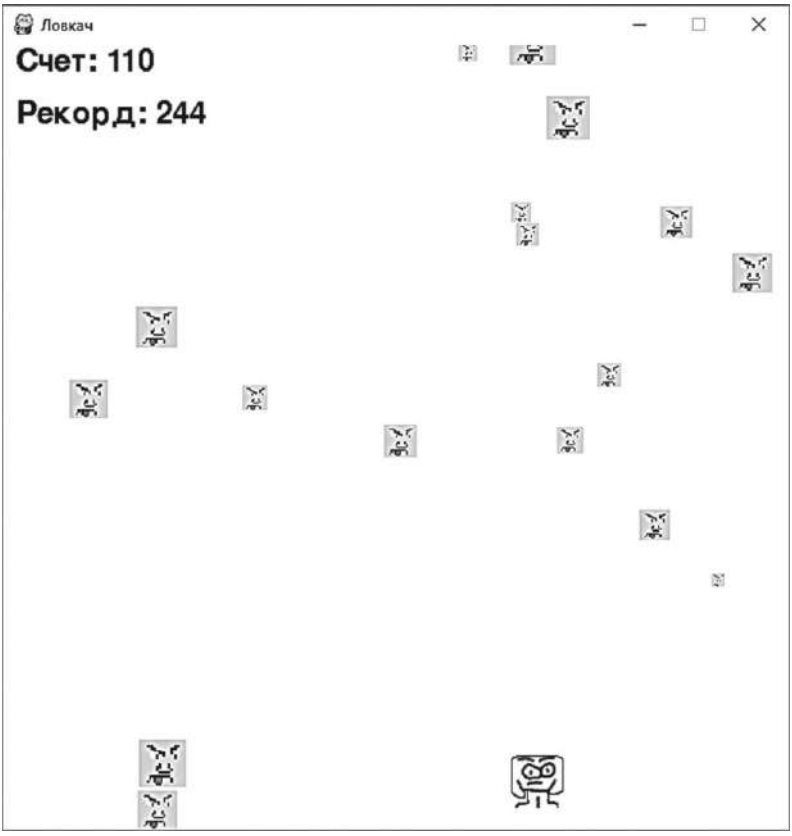


Рис. 21.1. Снимок игры «Ловкач»

Исходный код игры «Ловкач»

В редакторе файлов создайте новый файл, выбрав команду меню **File** ⇒ **New File** (Файл ⇒ Новый файл). В открывшемся окне введите приведенный ниже исходный код и сохраните файл под именем *dodger.py*. Поместите файлы изображений и звуков в папку с файлом *dodger.py*. Затем нажмите клавишу **F5** и запустите программу. Если при выполнении программы возникают ошибки, сравните код, который вы набрали, с оригинальным кодом с помощью онлайн-инструмента на сайте **inventwithpython.com/diff/**.

```
1. import pygame, random, sys
2. from pygame.locals import *
3.
4. WINDOWWIDTH = 600
5. WINDOWHEIGHT = 600
6. TEXTCOLOR = (0, 0, 0)
7. BACKGROUNDCOLOR = (255, 255, 255)
8. FPS = 60
9. BADDIEMINSIZE = 10
10. BADDIEMAXSIZE = 40
11. BADDIEMINSPEED = 1
12. BADDIEMAXSPEED = 8
13. ADDNEWBADDIERATE = 6
14. PLAYERMOVERATE = 5
15.
16. def terminate():
17.     pygame.quit()
18.     sys.exit()
19.
20. def waitForPlayerToPressKey():
21.     while True:
22.         for event in pygame.event.get():
23.             if event.type == QUIT:
24.                 terminate()
25.             if event.type == KEYDOWN:
26.                 if event.key == K_ESCAPE: # Нажатие ESC осуществляет выход.
27.                     terminate()
28.                 return
29.
30. def playerHasHitBaddie(playerRect, baddies):
31.     for b in baddies:
32.         if playerRect.colliderect(b['rect']):
33.             return True
34.     return False
35.
36. def drawText(text, font, surface, x, y):
37.     textobj = font.render(text, 1, TEXTCOLOR)
38.     textrect = textobj.get_rect()
39.     textrect.topleft = (x, y)
40.     surface.blit(textobj, textrect)
```

```
41.
42. # Настройка pygame, окна и указателя мыши.
43. pygame.init()
44. mainClock = pygame.time.Clock()
45. windowSurface = pygame.display.set_mode((WINDOWWIDTH, WINDOWHEIGHT))
46. pygame.display.set_caption('Ловкач')
47. pygame.mouse.set_visible(False)
48.
49. # Настройка шрифтов.
50. font = pygame.font.SysFont(None, 35)
51.
52. # Настройка звуков.
53. gameOverSound = pygame.mixer.Sound('gameover.wav')
54. pygame.mixer.music.load('background.mid')
55.
56. # Настройка изображений.
57. playerImage = pygame.image.load('player.png')
58. playerRect = playerImage.get_rect()
59. baddieImage = pygame.image.load('baddie.png')
60.
61. # Вывод начального экрана.
62. windowSurface.fill(BACKGROUNDCOLOR)
63. drawText('Ловкач', font, windowSurface, (WINDOWWIDTH / 3), (WINDOWHEIGHT / 3))
64. drawText('Нажмите клавишу для начала игры', font, windowSurface, (WINDOWWIDTH / 5) - 30, (WINDOWHEIGHT / 3) + 50)
65. pygame.display.update()
66. waitForPlayerToPressKey()
67.
68. topScore = 0
69. while True:
70.     # Настройка начала игры.
71.     baddies = []
72.     score = 0
73.     playerRect.topleft = (WINDOWWIDTH / 2, WINDOWHEIGHT - 50)
74.     moveLeft = moveRight = moveUp = moveDown = False
75.     reverseCheat = slowCheat = False
76.     baddieAddCounter = 0
77.     pygame.mixer.music.play(-1, 0.0)
78.
79.     while True: # Игровой цикл выполняется, пока игра работает.
80.         score += 1 # Увеличение количества очков.
81.
82.         for event in pygame.event.get():
```

```
83.             if event.type == QUIT:
84.                 terminate()
85.
86.             if event.type == KEYDOWN:
87.                 if event.key == K_z:
88.                     reverseCheat = True
89.                 if event.key == K_x:
90.                     slowCheat = True
91.                 if event.key == K_LEFT or event.key == K_a:
92.                     moveRight = False
93.                     moveLeft = True
94.                 if event.key == K_RIGHT or event.key == K_d:
95.                     moveLeft = False
96.                     moveRight = True
97.                 if event.key == K_UP or event.key == K_w:
98.                     moveDown = False
99.                     moveUp = True
100.                 if event.key == K_DOWN or event.key == K_s:
101.                     moveUp = False
102.                     moveDown = True
103.
104.             if event.type == KEYUP:
105.                 if event.key == K_z:
106.                     reverseCheat = False
107.                     score = 0
108.                 if event.key == K_x:
109.                     slowCheat = False
110.                     score = 0
111.                 if event.key == K_ESCAPE:
112.                     terminate()
113.
114.                 if event.key == K_LEFT or event.key == K_a:
115.                     moveLeft = False
116.                 if event.key == K_RIGHT or event.key == K_d:
117.                     moveRight = False
118.                 if event.key == K_UP or event.key == K_w:
119.                     moveUp = False
120.                 if event.key == K_DOWN or event.key == K_s:
121.                     moveDown = False
122.
123.             if event.type == MOUSEMOTION:
124.                 # Если мышь движется, переместить игрока к указателю мыши.
```

```
125.                playerRect.centerx = event.pos[0]
126.                playerRect.centery = event.pos[1]
127.        # Если необходимо, добавить новых злодеев в верхнюю часть экрана.
128.        if not reverseCheat and not slowCheat:
129.            baddieAddCounter += 1
130.        if baddieAddCounter == ADDNEWBADDIERATE:
131.            baddieAddCounter = 0
132.            baddieSize = random.randint(BADDIEMINSIZE, BADDIEMAXSIZE)
133.  newBaddie={'rect':pygame.Rect(random.randint(0,WINDOWWIDTH-baddieSize),0-baddieSize,baddieSize,baddieSize),
134.                        'speed': random.randint(BADDIEMINSPEED, BADDIEMAXSPEED),
135.                    'surface':pygame.transform.scale(baddieImage, (baddieSize, baddieSize)),
136.                        }
137.
138.            baddies.append(newBaddie)
139.
140.        # Перемещение игрока по экрану.
141.        if moveLeft and playerRect.left > 0:
142.            playerRect.move_ip(-1 * PLAYERMOVERATE, 0)
143.        if moveRight and playerRect.right < WINDOWWIDTH:
144.            playerRect.move_ip(PLAYERMOVERATE, 0)
145.        if moveUp and playerRect.top > 0:
146.            playerRect.move_ip(0, -1 * PLAYERMOVERATE)
147.        if moveDown and playerRect.bottom < WINDOWHEIGHT:
148.            playerRect.move_ip(0, PLAYERMOVERATE)
149.
150.        # Перемещение злодеев вниз.
151.        for b in baddies:
152.            if not reverseCheat and not slowCheat:
153.                b['rect'].move_ip(0, b['speed'])
154.            elif reverseCheat:
155.                b['rect'].move_ip(0, -5)
156.            elif slowCheat:
157.                b['rect'].move_ip(0, 1)
158.
159.        # Удаление злодеев, упавших за нижнюю границу экрана.
160.        for b in baddies[:]:
161.            if b['rect'].top > WINDOWHEIGHT:
162.                baddies.remove(b)
163.
164.        # Отображение в окне игрового мира.
165.        windowSurface.fill(BACKGROUNDCOLOR)
166.
```

```
167.         # Отображение количества очков и лучшего результата.
168.         drawText('Счет: %s' % (score), font, windowSurface, 10, 0)
169.         drawText('Рекорд: %s' % (topScore), font, windowSurface, 10, 40)
170.
171.         # Отображение прямоугольника игрока.
172.         windowSurface.blit(playerImage, playerRect)
173.
174.         # Отображение каждого злодея.
175.         for b in baddies:
176.             windowSurface.blit(b['surface'], b['rect'])
177.
178.         pygame.display.update()
179.
180.         # Проверка, попал ли в игрока какой-либо из злодеев.
181.         if playerHasHitBaddie(playerRect, baddies):
182.             if score > topScore:
183.                 topScore = score # установка нового рекорда
184.             break
185.
186.         mainClock.tick(FPS)
187.
188.     # Отображение игры и вывод надписи 'Игра окончена'.
189.     pygame.mixer.music.stop()
190.     gameOverSound.play()
191.
192.     drawText('ИГРА ОКОНЧЕНА!', font, windowSurface, (WINDOWWIDTH / 3), (WINDOWHEIGHT / 3))
193.     drawText('Нажмите клавишу для начала новой игры', font, windowSurface, (WINDOWWIDTH /
3) - 120, (WINDOWHEIGHT / 3) + 50)
194.     pygame.display.update()
195.     waitForPlayerToPressKey()
196.
197.     gameOverSound.stop()
```

Импорт модулей

Игра «Ловкач» импортирует те же модули, что и предыдущие pygame-программы — pygame, random, sys и pygame.locals.

```
1. import pygame, random, sys
2. from pygame.locals import *
```

Модуль pygame.locals содержит несколько констант, используемых pygame, таких как типы событий (QUIT, KEYDOWN и так далее) и клавиши клавиатуры (K_ESCAPE, K_LEFT и так далее). Используя синтаксис from pygame.locals import *, вы можете использовать в исходном коде просто QUIT вместо pygame.locals.QUIT.

Создание констант

Код в строках 4–7 устанавливают константы для размеров окна, цвета текста и цвета фона.

```
4. WINDOWWIDTH = 600
5. WINDOWHEIGHT = 600
6. TEXTCOLOR = (0, 0, 0)
7. BACKGROUNDCOLOR = (255, 255, 255)
```

Мы используем константы, потому что это гораздо более наглядно, чем если бы мы набирали значения вручную. Например, строка кода windowSurface.fill(BACKGROUNDCOLOR) более понятна, чем windowSurface.fill((255, 255, 255)).

Также это позволит вам легко модифицировать игру. Изменив значения переменной WINDOWWIDTH в строке 4, вы автоматически меняете код везде, где используется переменная WINDOWWIDTH. Если бы вместо этого вы использовали значение 600, вам пришлось бы изменить каждое вхождение 600 в коде. Проще один раз изменить значение константы.

В строке 8 вы создаете константу для FPS, количества кадров в секунду, необходимого для запуска игры.

```
8. FPS = 60
```

Кадр — это экран, отображаемый при одном переборе игрового цикла. В строке 186 вы передаете значение переменной FPS методу mainClock.tick(), чтобы функция знала, на какое количество времени приостанавливать программу. Здесь переменной FPS присвоено значение 60, но вы можете изменить значение переменной FPS в большую сторону, чтобы игра работала быстрее, или в меньшую, чтобы замедлить ее.

Код в строках 9–13 создают еще несколько констант для падающих спрайтов злодеев.

```
9. BADDIEMINSIZE = 10
10. BADDIEMAXSIZE = 40
11. BADDIEMINSPEED = 1
```

```
12. BADDIEMAXSPEED = 8
13. ADDNEWBADDIERATE = 6
```

Значения ширины и высоты спрайтов злодеев будут находиться в диапазоне между BADDIEMINSIZE и BADDIEMAXSIZE. Скорость, с которой спрайты злодеев падают на экране, будет находиться в диапазоне между BADDIEMINSPEED и BADDIEMAXSPEED пикселей за одну итерацию игрового цикла. А новый спрайт злодея будет добавляться в верхней части окна каждые ADDNEWBADDIERATE итераций игрового цикла.

Наконец, константа PLAYERMOVERATE хранит количество пикселей, на которое спрайт игрока перемещается в окне при каждой итерации игрового цикла (если перемещается).

```
14. PLAYERMOVERATE = 5
```

Увеличивая это число, вы увеличиваете скорость перемещения персонажа.

Определение функций

Для этой игры вы создадите несколько функций. Функции terminate() и waitForPlayerToPressKey() будут, соответственно, завершать и приостанавливать игру, функция playerHasHitBaddie() будет отслеживать столкновения спрайта игрока со спрайтами злодеев, а функция drawText() будет отображать на экране счет и другой текст.

Завершение игры и добавление паузы

Модуль pygame требует, чтобы для завершения игры вы вызывали как функцию pygame.quit(), так и sys.exit(). Строки 16–18 помещают их обеих в функцию terminate().

```
16. def terminate():
17.     pygame.quit()
18.     sys.exit()
```

Теперь, вместо того, чтобы вызывать и pygame.quit(), и sys.exit(), вам нужно вызвать только функцию terminate().

В некоторых случаях вам понадобится приостановить выполнение программы до тех пор, пока игрок не нажмет клавишу, — например, в самом начале игры, когда появится заголовок «Ловкач», или в конце, когда отобра-

жается надпись «Игра окончена». В строках 20–24 создается новая функция waitForPlayerToPressKey().

```
20. def waitForPlayerToPressKey():
21.     while True:
22.         for event in pygame.event.get():
23.             if event.type == QUIT:
24.                 terminate()
```

Внутри этой функции используется бесконечный цикл, который прерывается только при получении события KEYDOWN или QUIT. В начале цикла pygame.event.get() возвращает список объектов Event для проверки.

Если игрок закрыл окно во время ожидания программой, когда он нажмет клавишу, pygame сгенерирует событие QUIT, наличие которого вы проверяете в строке 23 с помощью event.type. Если игрок завершил игру, Python вызывает функцию terminate() в строке 24.

Если игра получает событие KEYDOWN, сначала нужно проверить, была ли нажата клавиша **Esc**.

```
25.             if event.type == KEYDOWN:
26.                 if event.key == K_ESCAPE: # Нажатие ESC осуществляет выход.
27.                     terminate()
28.                 return
```

Если игрок нажал клавишу **Esc**, программа должна завершиться. Если дело было не в этом, то интерпретатор пропустит блок if в строке 27 и перейдет прямо к инструкции return, что приведет к выходу из функции waitForPlayerToPressKey().

Если события QUIT или KEYDOWN не генерируются, код продолжает цикл. Поскольку цикл ничего не выполняет, все будет выглядеть так, будто игра замерла, пока игрок не нажмет клавишу.

Отслеживание столкновений со злодеями

Функция playerHasHitBaddie() вернет True, если спрайт игрока столкнулся со спрайтом одного из злодеев.

```
30. def playerHasHitBaddie(playerRect, baddies):
31.     for b in baddies:
32.         if playerRect.colliderect(b['rect']):
```

```
33.             return True
34.     return False
```

Параметр baddies представляет собой список структур данных словаря злодея. Каждый из этих словарей имеет ключ 'rect', а значение для этого ключа — объект Rect, который представляет размер и местоположение злодея.

playerRect также является объектом Rect. Объекты Rect имеют метод с именем colliderect(), который возвращает True, если объект Rect столкнулся с тем объектом Rect, который был передан этому методу. В противном случае colliderect() возвращает False.

Цикл for в строке 31 выполняет перебор каждого словаря злодея в списке baddies. Если любой из этих злодеев сталкивается с персонажем игрока, то playerHasHitBaddie() возвращает True. Если код перебрал всех злодеев в списке baddies, не обнаружив столкновения, playerHasHitBaddie() возвращает False.

Отображение текста в окне

Отображение текста в окне включает в себя несколько шагов, которые мы выполним с помощью функции drawText(). Таким образом, есть только одна функция, которую мы вызываем, когда хотим отобразить на экране количество очков игрока или надпись 'Игра окончена'.

```
36. def drawText(text, font, surface, x, y):
37.     textobj = font.render(text, 1, TEXTCOLOR)
38.     textrect = textobj.get_rect()
39.     textrect.topleft = (x, y)
40.     surface.blit(textobj, textrect)
```

Сначала вызов метода render() в строке 37 создает объект Surface, который форматирует текст определенным шрифтом.

Затем вам нужно узнать размер и местоположение объекта Surface. Можно получить объект Rect с этой информацией, используя метод Surface get_rect().

Объект Rect, возвращаемый из функции get_rect() в строке 38, содержит копию информации о ширине и высоте объекта Surface. Код в строке 39 изменяет местоположение объекта Rect, устанавливая новый кортеж для его атрибута topleft.

Наконец, код в строке 40 отображает объект Surface с отформатированным текстом на объекте Surface, который был передан функции drawText(). Отображение текста в pygame занимает на несколько шагов больше, чем про-

сто при вызове функции print(). Но если вы поместите этот код в единую функцию с именем drawText(), тогда для отображения текста на экране вам нужно будет вызвать лишь эту функцию.

Инициализация pygame и настройка окна

Теперь, когда с константами и функциями покончено, начнем вызывать функции pygame, которые настраивают окно и скорость работы.

```
42. # Настройка pygame, окна и указателя мыши.
43. pygame.init()
44. mainClock = pygame.time.Clock()
```

Код в строке 43 устанавливает модуль pygame, вызывая функцию pygame.init(). Код в строке 44 создает объект pygame.time.Clock() и сохраняет его в переменной mainClock. Этот объект поможет нам обеспечить выполнение программы с надлежащей скоростью.

Код в строке 45 создает новый объект Surface, который используется для отображения окна.

```
45. windowSurface = pygame.display.set_mode((WINDOWWIDTH, WINDOWHEIGHT))
```

Обратите внимание, что функции pygame.display.set_mode() передается только один аргумент — кортеж. Аргументы pygame.display.set_mode() — не два целых числа, а один кортеж из двух целых чисел. Вы можете указать ширину и высоту этого объекта Surface (и окна), передав кортеж с константами WINDOWWIDTH и WINDOWHEIGHT.

Функция pygame.display.set_mode() имеет второй необязательный параметр. Вы можете передать константу pygame.FULLSCREEN, чтобы окно заполнило весь экран. Взгляните на строку 45 с данной модификацией.

```
45. windowSurface = pygame.display.set_mode((WINDOWWIDTH, WINDOWHEIGHT), pygame.FULLSCREEN)
```

Для обозначения ширины и высоты окна по-прежнему передаются параметры WINDOWWIDTH и WINDOWHEIGHT, но изображение будет растянуто больше, чтобы соответствовать размеру экрана. Попробуйте запустить программу в полноэкранном режиме и в оконном.

Код в строке 46 создает заголовок окна со строкой 'Ловкач':

```
46. pygame.display.set_caption('Ловкач')
```

Эта надпись отображается в строке заголовка в верхней части окна.

В игре «Ловкач» указатель мыши должен быть невиден. Вам нужно, чтобы мышь перемещала персонажа игрока по экрану, но при этом указатель мыши помешал бы изображению персонажа. Мы можем сделать мышь невидимой с помощью всего одной строки кода.

```
47. pygame.mouse.set_visible(False)
```

Вызов pygame.mouse.set_visible(False) сообщает pygame, что указатель мыши нужно сделать невидимым.

Установка шрифтов, изображений и звуков

Поскольку в этой программе мы отображаем на экране текст, то нужно предоставить модулю pygame объект Font, который будет использоваться для текста. Код в строке 50 создает объект Font, вызывая функцию pygame.font.SysFont().

```
49. # Настройка шрифтов.
50. font = pygame.font.SysFont(None, 35)
```

При передаче значения None выбирается шрифт по умолчанию, при передаче значения 48 — размер шрифта составит 48 пунктов.

Затем мы создадим объекты Sound и настроим фоновую музыку.

```
52. # Настройка звуков.
53. gameOverSound = pygame.mixer.Sound('gameover.wav')
54. pygame.mixer.music.load('background.mid')
```

Функция-конструктор pygame.mixer.Sound() создает новый объект Sound и сохраняет ссылку на этот объект в переменной gameOverSound. В своих собственных играх вы можете создавать столько объектов звуков, сколько захотите.

Функция pygame.mixer.music.load() загружает звуковой файл для воспроизведения в качестве фоновой музыки. Эта функция не возвращает никаких

объектов. Единовременно может быть загружен только один фоновый звуковой файл. Фоновая музыка будет воспроизводиться во время игры непрерывно, но объекты `Sound` будут воспроизводиться только тогда, когда игрок столкнется со злодеем и проиграет.

Для этой игры можно использовать любой WAV или MIDI-файл. Некоторые звуковые файлы доступны в файлах примеров на веб-сайте https://eksmo.ru/files/python_games_sweigart.zip. Вы также можете использовать свои собственные звуковые файлы, если только присвоите файлам имена *gameover.wav* и *background.mid* или измените строки 53 и 54 в соответствии с именами файлов.

Затем нужно загрузить файлы изображений, которые будут использоваться для спрайтов игрока и злодеев.

```
56. # Настройка изображений.
57. playerImage = pygame.image.load('player.png')
58. playerRect = playerImage.get_rect()
59. baddieImage = pygame.image.load('baddie.png')
```

Изображение для персонажа сохраняется в *player.png*, а изображение для каждого злодея — в файле *baddie.png*.

Все злодеи выглядят одинаково, поэтому для них нужен только один файл изображения. Вы можете загрузить эти изображения с веб-сайта https://eksmo.ru/files/python_games_sweigart.zip.

Отображение начального экрана

Когда начинается игра, Python должен отобразить на экране ее название. Также нужно сообщить игроку, что он может начать игру, если нажмет любую клавишу. Этот экран отображается, чтобы после запуска программы у игрока было время приготовиться к игре. В строках 63 и 64 вводится код для вызова функции `drawText()`.

```
61. # Вывод начального экрана.
62. windowSurface.fill(BACKGROUNDCOLOR)
63. drawText('Ловкач', font, windowSurface, (WINDOWWIDTH / 3), (WINDOWHEIGHT / 3))
64. drawText('Нажмите клавишу для начала игры', font, windowSurface, (WINDOWWIDTH / 5) - 30, (WINDOWHEIGHT / 3) + 50)
65. pygame.display.update()
66. waitForPlayerToPressKey()
```

Мы передаем этой функции пять аргументов:

1. Строка текста, который вы хотите отобразить.
2. Шрифт этого текста.
3. Объект `Surface`, на котором будет отображаться текст.
4. Координата *x* на объекте `Surface`, где следует отобразить текст.
5. Координата *y* на объекте `Surface`, где следует отобразить текст.

Может показаться, что мы передаем много аргументов, но имейте в виду, что данный вызов каждый раз заменяет собой пять строк кода. Это сокращает код программы и упрощает поиск ошибок, поскольку проверять нужно меньше кода.

Функция `waitForPlayerToPressKey()` приостанавливает игру, зацикливаясь до тех пор, пока не будет сгенерировано событие `KEYDOWN`. Затем интерпретатор прерывает цикл, и программа продолжает работу.

Начало игры

Определив теперь все функции, можем начать писать основной код игры. Строки 68 и далее будут вызывать функции, которые мы определили до этого. Значение переменной `topScore` начинается с `0` при первом запуске программы. Каждый раз, когда игрок проигрывает, побив предыдущий рекорд, он заменяется последним набранным количеством очков.

```
68. topScore = 0
69. while True:
```

Бесконечный цикл, запущенный в строке 69, технически не является игровым циклом. Игровой цикл управляет событиями и отображает окно, пока игра работает. А этот цикл `while` повторяется каждый раз, когда игрок начинает новую игру. Когда игрок проигрывает, и игра сбрасывается, интерпретатор возвращается к строке 69.

Сначала нужно присвоить `baddies` значение пустого списка.

```
70.     # Настройка начала игры.
71.     baddies = []
72.     score = 0
```

Переменная `baddies` — это список словарей со следующими ключами:

- **`'rect'`**. Объект `Rect`, который описывает, где находится спрайт злодея и какого он размера.

- **'speed'**. Указывает, с какой скоростью спрайт злодея падает на экране. Целое число обозначает количество пикселей на одну итерацию игрового цикла.
- **'surface'**. Объект Surface, на котором отображено масштабированное изображение злодея. Это тот Surface, который отображен на объекте Surface, возвращаемом pygame.display.set_mode().

Код в строке 72 сбрасывает очки игрока до 0.

Начальное расположение пользователя — по центру экрана по ширине и в 50 пикселях выше от нижней границы, что указано в строке 73.

```
73.     playerRect.topleft = (WINDOWWIDTH / 2, WINDOWHEIGHT - 50)
```

Первым элементом кортежа в строке 73 является координата *x* левой стороны, вторым — координата *y* верхней стороны персонажа игрока.

Затем мы устанавливаем переменные для перемещений игрока и чит-кодов.

```
74.     moveLeft = moveRight = moveUp = moveDown = False
75.     reverseCheat = slowCheat = False
76.     baddieAddCounter = 0
```

Переменным перемещения moveLeft, moveRight, moveUp и moveDown присвоено значение False. Переменные reverseCheat и slowCheat также установлены равными False. Они изменят значение на True только тогда, когда игрок включит эти чит-коды, удерживая клавишу **Z** или **X**, соответственно.

Переменная baddieAddCounter выступает счетчиком, сообщающим программе, когда в верхнюю часть экрана нужно добавить спрайт нового злодея. Значение baddieAddCounter увеличивается на 1 при каждой итерации игрового цикла. (Это напоминает код в разделе «Добавление новых блоков “еды”» главы 19.)

Когда переменная baddieAddCounter принимает значение ADDNEWBADDIERATE, значение переменной baddieAddCounter сбрасывается до 0, и в верхнюю часть экрана добавляется новый спрайт злодея (эта проверка выполняется позже в строке 130).

Фоновая музыка начинает воспроизведение в строке 77 с вызовом функции pygame.mixer.music.play().

```
77.     pygame.mixer.music.play(-1, 0.0)
```

Поскольку первый аргумент равен -1, pygame повторяет музыку бесконечно. Второй аргумент — число с плавающей запятой, сообщающее, в скольких секундах после начала композиции находится момент, с которого ее следует проигрывать. Если передать 0,0, музыка начнет воспроизводиться с самого начала.

Игровой цикл

Код игрового цикла постоянно обновляет состояние игрового мира, изменяя положение спрайтов игрока и злодеев, обрабатывая события, созданные pygame, и отображая новый игровой мир на экране. Все это происходит несколько десятков раз в секунду, что заставляет игру работать в режиме реального времени.

Строка 79 — начало основного игрового цикла.

```
79.     while True: # Игровой цикл выполняется, пока игра работает.
80.         score += 1 # Увеличение количества очков.
```

Код в строке 80 увеличивает количество очков игрока при каждой итерации игрового цикла.

Чем дольше игрок продолжает играть, не проигрывая, тем больше у него становится очков. Цикл прервется только тогда, когда игрок либо проиграет, либо выйдет из программы.

Обработка событий клавиатуры

Существуют четыре типа событий, которые программа будет обрабатывать: QUIT, KEYDOWN, KEYUP и MOUSEMOTION.

В строке 82 начинается код, управляющий событиями.

```
82.         for event in pygame.event.get():
83.             if event.type == QUIT:
84.                 terminate()
```

Он вызывает функцию pygame.event.get(), которая возвращает список объектов Event. Каждый объект Event представляет собой событие, которое произошло с момента последнего вызова pygame.event.get(). Код проверяет атрибут type объекта Event, чтобы узнать, какой это тип события, а затем обрабатывает его соответствующим образом.

Если атрибут type объекта Event равен QUIT, значит, пользователь закрыл программу. Константа QUIT была импортирована из модуля pygame.locals.

Если тип события — KEYDOWN, значит, игрок нажал клавишу.

```
86.            if event.type == KEYDOWN:
87.                if event.key == K_z:
88.                    reverseCheat = True
89.                if event.key == K_x:
90.                    slowCheat = True
```

С помощью event.key == K_z код в строке 87 проверяет, указывает ли событие на нажатие клавиши **Z**.

Если это условие истинно, интерпретатор присваивает переменной reverseCheat значение True, чтобы активировать чит-код обратного направления. Аналогично код в строке 89 проверяет, нажата ли клавиша **X** для активации чит-кода замедления.

Код в строках 91–102 определяет, было ли создано событие, отвечающее за нажатие игроком одной из клавиш со стрелками или клавиш **W**, **A**, **S** или **D**. Этот код аналогичен обработке нажатий клавиш клавиатуры в программах из предыдущих глав.

Если тип события — KEYUP, значит, игрок отпустил клавишу.

```
104.            if event.type == KEYUP:
105.                if event.key == K_z:
106.                    reverseCheat = False
107.                    score = 0
108.                if event.key == K_x:
109.                    slowCheat = False
110.                    score = 0
```

Код в строке 105 проверяет, отпустил ли игрок клавишу **Z**, что деактивирует чит-код обратного направления. В таком случае код в строке 106 присваивает переменной reverseCheat значение False, а код в строке 107 сбрасывает счет до 0. Сброс очков предназначен для того, чтобы отбивать у игрока желание использовать чит-коды.

Код в строках 108 — 110 выполняет то же самое для клавиши **X** и чит-кода замедления. Когда клавиша **X** отпускается, переменной slowCheat присваивается значение False, а счет игрока сбрасываются до 0.

В любой момент игры игрок может нажать клавишу **Esc**, чтобы завершить игру.

```
111.                if event.key == K_ESCAPE:
112.                    terminate()
```

Код в строке 111 определяет, была ли отпущена клавиша **Esc**, проверяя `event.key == K_ESCAPE`. Если так и есть, код в строке 112 вызывает функцию `terminate()` для выхода из программы.

Код в строках 114–121 проверяет, была ли отпущена одна из клавиш со стрелками или **W**, **A**, **S** или **D**.

В этом случае интерпретатор присваивает соответствующей переменной перемещения значение `False`. Это напоминает код перемещения в программах из глав 19 и 20.

Обработка событий мыши

Теперь, когда вы разобрались с событиями клавиатуры, давайте обработаем все возможные события мыши. Игра «Ловкач» не реагирует на нажатие игроком кнопки мыши, но откликается на движение ее курсора. Благодаря этому у игрока есть два способа управления персонажем в игре: клавиатурой или мышью.

Событие `MOUSEMOTION` генерируется каждый раз, когда мышь перемещается.

```
123.            if event.type == MOUSEMOTION:
124.                # Если мышь движется, переместить игрока к указателю мыши.
125.                playerRect.centerx = event.pos[0]
126.                playerRect.centery = event.pos[1]
```

Объекты `Event` с атрибутом `type`, имеющим значение `MOUSEMOTION`, также содержат атрибут `pos` для обработки позиции события мыши. Атрибут `pos` хранит кортеж координат *x* и *y* места, куда в окне переместился указатель мыши.

Если тип события — `MOUSEMOTION`, персонаж игрока перемещается в положение указателя мыши.

Код в строках 125 и 126 устанавливает координаты *x* и *y* центра спрайта игрока равными координатам указателя мыши.

Добавление новых злодеев

При каждой итерации игрового цикла код увеличивает значение переменной baddieAddCounter на единицу.

```
127.        # Если необходимо, добавить новых злодеев в верхнюю часть экрана.
128.        if not reverseCheat and not slowCheat:
129.            baddieAddCounter += 1
```

Это происходит, только если чит-коды не включены. Помните, что значение переменных reverseCheat и slowCheat равно True, пока удерживаются клавиши **Z** и **X**, соответственно. Пока они удерживаются, значение переменной baddieAddCounter не увеличивается, так что в верхней части экрана не появятся новые злодеи.

Когда значение переменной baddieAddCounter достигает значения ADDNEWBADDIERATE, приходит время добавить нового злодея. Сначала значение переменной baddieAddCounter сбрасывается до 0.

```
130.        if baddieAddCounter == ADDNEWBADDIERATE:
131.            baddieAddCounter = 0
132.            baddieSize = random.randint(BADDIEMINSIZE, BADDIEMAXSIZE)
133.  newBaddie={'rect':pygame.Rect(random.randint(0,WINDOWWIDTH-baddieSize),0-baddieSize,baddieSize,baddieSize),
134.                        'speed': random.randint(BADDIEMINSPEED, BADDIEMAXSPEED),
135.                    'surface':pygame.transform.scale(baddieImage, (baddieSize, baddieSize)),
136.                        }
```

Код в строке 132 определяет размер злодея в пикселях. Размер будет случайным целым числом в диапазоне от BADDIEMINSIZE до BADDIEMAXSIZE, то есть между константами со значениями 10 и 40, присвоенными им в строках кода 9 и 10, соответственно.

В строке 133 создается новая структура данных злодея. Запомните, структура данных для baddies — это просто словарь с ключами 'rect', 'speed' и 'surface'. Ключ 'rect' содержит ссылку на объект Rect, который хранит местоположение и размер злодея.

Функция-конструктор pygame.Rect() имеет четыре параметра: координату *x* верхней стороны области, координату *y* левой стороны области, а также ширину и высоту в пикселях.

Злодей должен появиться в случайной позиции в верхней части окна, поэтому передайте random.randint(0, WINDOWWIDTH - baddieSize) для координаты

x злодея. Причина, по которой вы передаете `WINDOWWIDTH - baddieSize` вместо переменной `WINDOWWIDTH`, заключается в том, что если левая сторона злодея находится слишком далеко справа, тогда часть злодея выйдет за пределы окна, и не будет видна на экране.

Нижняя сторона злодея должна располагаться прямо над верхней границей окна, координата *y* которой равна `0`. Чтобы поместить туда нижнюю сторону злодея, присвойте его верхней стороне значение `0 - baddieSize`.

Ширина и высота спрайта злодея должны быть одинаковыми (его изображение имеет квадратную форму), поэтому передайте значение `baddieSize` в качестве третьего и четвертого аргументов.

Скорость, с которой злодей перемещается вниз по экрану, задается в ключе `'speed'`. Установите его равным случайному целому числу между `BADDIEMINSPEED` и `BADDIEMAXSPEED`.

Затем код в строке 138 добавит недавно созданную структуру данных злодея в список структур данных злодея.

```
138.            baddies.append(newBaddie)
```

Программа использует этот список, чтобы проверить, столкнулся ли спрайт игрока с каким-либо спрайтом злодея, и определить, где в окне отображать злодеев.

Перемещение спрайтов игрока и злодеев

Четыре переменных перемещения `moveLeft`, `moveRight`, `moveUp` и `moveDown` принимают значения `True` и `False`, когда `pygame` генерирует события `KEYDOWN` и `KEYUP`, соответственно. Если спрайт игрока перемещается влево, а его левая сторона больше `0` (то есть больше левой границы окна), то объект `playerRect` должен перемещаться влево.

```
140.        # Перемещение игрока по экрану.
141.        if moveLeft and playerRect.left > 0:
142.            playerRect.move_ip(-1 * PLAYERMOVERATE, 0)
```

Метод `move_ip()` будет перемещать объект `Rect` горизонтально или вертикально на определенное количество пикселей. Первым аргументом `move_ip()` является количество пикселей, на которое объект `Rect` нужно переместить вправо (чтобы переместить его влево, передайте отрицательное целое число). Второй аргумент — на сколько пикселей переместить объект `Rect` вниз

(для перемещения вверх передайте отрицательное целое число). Например, playerRect.move_ip(10, 20) переместит объект Rect на 10 пикселей вправо и на 20 пикселей вниз, а playerRect.move_ip(-5, -15) — на 5 пикселей влево и на 15 пикселей вверх. Буквы ip в конце имени метода move_ip() означают «in place — на месте». Это связано с тем, что метод изменяет сам объект Rect вместо того, чтобы возвращать новый объект Rect с внесенными изменениями. Существует также метод move(), который не изменяет объект Rect, но вместо этого создает и возвращает новый объект Rect в новом расположении.

Объект playerRect всегда будет перемещаться на количество пикселей PLAYERMOVERATE. Чтобы сделать отрицательное целое число из положительного, умножьте его на -1. В строке 142 выражение -1 * PLAYERMOVERATE равно -5, поскольку в PLAYERMOVERATE хранится 5. Поэтому вызов playerRect.move_ip(-1 * PLAYERMOVERATE, 0) переместит playerRect на 5 пикселей влево от его текущего местоположения.

Код в строках 143 по 148 делает то же самое для трех остальных направлений: вправо, вверх и вниз.

```
143.         if moveRight and playerRect.right < WINDOWWIDTH:
144.             playerRect.move_ip(PLAYERMOVERATE, 0)
145.         if moveUp and playerRect.top > 0:
146.             playerRect.move_ip(0, -1 * PLAYERMOVERATE)
147.         if moveDown and playerRect.bottom < WINDOWHEIGHT:
148.             playerRect.move_ip(0, PLAYERMOVERATE)
```

Каждая из трех инструкций if в строках с 143 по 148 проверяет, присвоено ли значение True соответствующей переменной перемещения и находится ли сторона объекта Rect игрока внутри окна. Затем она вызывает метод move_ip() для перемещения объекта Rect.

Теперь код перебирает каждую структуру данных злодея в списке baddies, чтобы немного переместить злодеев вниз.

```
150.         # Перемещение злодеев вниз.
151.         for b in baddies:
152.             if not reverseCheat and not slowCheat:
153.                 b['rect'].move_ip(0, b['speed'])
```

Если ни один из чит-кодов не был активирован, то местоположение злодея изменяется на количество пикселей, равное его скорости (хранящейся в ключе 'speed').

Реализация чит-кодов

Если активирован чит-код обратного направления, тогда злодей должен переместиться вверх на 5 пикселей.

```
154.            elif reverseCheat:
155.                b['rect'].move_ip(0, -5)
```

Передача `-5` в качестве второго аргумента методу `move_ip()` приведет к перемещению объекта `Rect` вверх на 5 пикселей.

Если был активирован чит-код замедления, тогда злодей все равно будет двигаться вниз, но со скоростью 1 пиксель за итерацию игрового цикла.

```
156.            elif slowCheat:
157.                b['rect'].move_ip(0, 1)
```

Значение нормальной скорости спрайта злодея (хранящейся в ключе `'speed'` структуры данных злодея) игнорируется при активации чит-кода замедления.

Удаление спрайтов злодеев

Все спрайты злодеев, которые выпали за пределы нижней границы окна, должны быть удалены из списка `baddies`. Помните, что нельзя добавлять или удалять элементы списка во время перебора этого списка. Вместо перебора списка `baddies` с помощью цикла `for`, перебирайте *копию* списка `baddies`. Чтобы сделать эту копию, используйте пустой оператор среза `[:]`.

```
159.        # Удаление злодеев, упавших за нижнюю границу экрана.
160.        for b in baddies[:]:
```

Цикл `for` в строке 160 использует переменную `b` для текущего элемента при переборе `baddies[:]`. Если злодей находится ниже нижней границы окна, его необходимо удалить, что мы и делаем в строке 162.

```
161.            if b['rect'].top > WINDOWHEIGHT:
162.                baddies.remove(b)
```

Словарь b — это текущая структура данных злодея из списка baddies[:]. Каждая структура данных злодея в списке представляет собой словарь с ключом 'rect', в котором хранится объект Rect. Таким образом, b['rect'] является объектом Rect для спрайта злодея. Наконец, атрибутом top выступает координата *y* верхней стороны прямоугольной области. Помните, что координаты *y* увеличиваются по направлению вниз. Итак, выражение b['rect'].top > WINDOWHEIGHT проверит, находится ли верхняя сторона злодея ниже нижней границы окна. Если это условие истинно, тогда код в строке 162 удаляет структуру данных злодея из списка baddies.

Отображение окна

После того, как все структуры данных были обновлены, с помощью функций модуля pygame нужно отобразить игровой мир. Благодаря тому, что игровой цикл выполняется несколько раз в секунду, возникает эффект плавного перемещения спрайтов игрока и злодеев.

До того, как будет отображено что-либо еще, код в строке 165 заливает цветом все окно, стирая все, что в нем было отображено ранее.

```
164.        # Отображение в окне игрового мира.
165.        windowSurface.fill(BACKGROUNDCOLOR)
```

Помните, что объект Surface в windowSurface — особый, потому что он возвращается pygame.display.set_mode(). Поэтому все, что отображено на этом объекте Surface, появится на экране после вызова pygame.display.update().

Отображение очков игрока

Код в строках 168 и 169 выводит текст о текущем счете и лучшем результате в верхнем левом углу окна.

```
167.        # Отображение количества очков и лучшего результата.
168.        drawText('Счет: %s' % (score), font, windowSurface, 10, 0)
169.        drawText('Рекорд: %s' % (topScore), font, windowSurface, 10, 40)
```

В выражении 'Счет:% s'% (score) используется интерполяция строк для вставки значения переменной score в строку. Эта строка, объект Font, хранящийся в переменной font, объект Surface для отображения на нем текста, а также координаты *x* и *y* позиции, куда должен быть помещен текст, переда-

ются методу drawText(), который будет обрабатывать вызов методов render() и blit().

Сделайте то же самое для отображения рекорда. Передайте значение 40 в качестве координаты *y* вместо 0, чтобы значение рекорда отображалось под строкой текущего количества очков.

Отображение спрайтов игрока и злодеев

Информация об игроке хранится в двух различных переменных. playerImage — это объект Surface, который содержит все цветные пиксели, из которых состоит изображение спрайта игрока. playerRect — это объект Rect, который хранит размер и местоположение спрайта игрока.

Метод blit() отображает спрайт игрока (находится в playerImage) на windowSurface в позиции playerRect.

```
171.        # Отображение прямоугольника игрока.
172.        windowSurface.blit(playerImage, playerRect)
```

Цикл for в строке 175 отображает спрайт каждого злодея на объекте windowSurface.

```
174.        # Отображение каждого злодея.
175.        for b in baddies:
176.            windowSurface.blit(b['surface'], b['rect'])
```

Каждый элемент в списке baddies — это словарь. Ключи словарей 'surface' и 'rect' содержат объект Surface с изображением злодея и объект Rect с информацией о его расположении и размере, соответственно.

Теперь, когда на windowSurface все отображено, нам нужно обновить окно, чтобы игрок мог видеть его содержимое.

```
178.        pygame.display.update()
```

Отобразим на экране этот объект Surface, вызвав метод update().

Проверка на столкновения

Код в строке 181 проверяет, столкнулся ли спрайт игрока со спрайтом какого-либо злодея, путем вызова функции playerHasHitBaddie(). Эта функ-

ция вернет значение True, если персонаж игрока столкнулся с любым из злодеев из списка baddies. В противном случае функция возвращает значение False.

```
180.        # Проверка, попал ли в игрока какой-либо из злодеев.
181.        if playerHasHitBaddie(playerRect, baddies):
182.            if score > topScore:
183.                topScore = score # установка нового рекорда
184.            break
```

Если спрайт игрока столкнулся со спрайтом злодея, и если текущее количество очков больше, чем предыдущий рекорд, тогда код в строках 182 и 183 обновляет его. Интерпретатор выходит из игрового цикла в строке 184 и переходит к строке 189, завершая игру.

Во избежание выполнения компьютером игрового цикла настолько быстро, насколько он может (это было бы слишком быстро для игрока, и он бы сразу же проиграл), мы вызываем метод mainClock.tick(), приостанавливая игру на очень короткое время:

```
186.        mainClock.tick(FPS)
```

Эта пауза будет достаточно длинной, чтобы обеспечить около 40 (значение, хранящееся в переменной FPS) итераций игрового цикла каждую секунду.

Экран окончания игры

Когда игрок проигрывает, воспроизведение фоновой музыки прекращается и исполняется звуковой эффект, свидетельствующий об окончании игры.

```
188.    # Отображение игры и вывод надписи 'Игра окончена'.
189.    pygame.mixer.music.stop()
190.    gameOverSound.play()
```

Код в строке 189 вызывает функцию stop() в модуле pygame.mixer.music для остановки фоновой музыки. Код в строке 190 вызывает метод play() в объекте Sound, хранящемся в gameOverSound.

Затем код в строках 192 и 193 вызывает функцию drawText(), чтобы вывести текст «ИГРА ОКОНЧЕНА!» на объект windowSurface.

```
192.     drawText('ИГРА ОКОНЧЕНА!', font, windowSurface, (WINDOWWIDTH / 3), (WINDOWHEIGHT / 3))
193. drawText('Нажмите клавишу для начала новой игры', font, windowSurface, (WINDOWWIDTH / 3) - 120, (WINDOWHEIGHT / 3) + 50)
194.     pygame.display.update()
195.     waitForPlayerToPressKey()
```

Код в строке 194 вызывает метод update(), чтобы вывести этот объект Surface на экран. После отображения текста игра останавливается, пока игрок не нажмет клавишу, вызвав функцию waitForPlayerToPressKey().

После того как игрок нажимает клавишу, интерпретатор выходит из вызова waitForPlayerToPressKey() в строке 195. В зависимости от того, сколько времени у игрока уходит на то, чтобы нажать клавишу, звуковой эффект либо продолжает проигрываться, либо нет. Чтобы остановить этот звуковой эффект перед началом новой игры, код в строке 197 вызывает функцию gameOverSound.stop().

```
197.     gameOverSound.stop()
```

Вот наша графическая игра и закончена!

Изменение игры «Ловкач»

Вы можете найти игру слишком простой или слишком сложной. К счастью, ее легко изменить, ведь мы потратили время на использование констант вместо того, чтобы вводить значения напрямую. Теперь все, что нам нужно сделать для модификации игры — это изменить значения констант.

Например, если вы хотите, чтобы программа в целом выполнялась медленнее, присвойте переменной FPS в строке 8 меньшее значение, например, 20. Это приведет к тому, что спрайты злодеев и игрока будут перемещаться медленнее, поскольку игровой цикл будет выполняться лишь 20 раз в секунду вместо 40.

Если хотите замедлить только спрайты злодеев, уменьшите значение переменной BADDIEMAXSPEED, например до 4. Это заставит спрайты всех злодеев перемещаться на количество пикселей в диапазоне от 1 (значение BADDIEMINSPEED) до 4, а не 1–8, за одну итерацию игрового цикла.

Если хотите вместо большого количества маленьких злодеев получить несколько злодеев побольше, увеличьте значение переменной `ADDNEWBADDIERATE` до `12`, `BADDIEMINSIZE` до `40`, а значение переменной `BADDIEMAXSIZE` до `80`. Теперь спрайты злодеев добавляются каждые 12 итераций игрового цикла вместо 6 итераций, так что их будет вдвое меньше, чем раньше. Но, чтобы игра оставалась интересной, размер злодеев значительно увеличится.

Не меняя базовую часть игры, можно изменить любую из констант и радикально повлиять на работу игры. Продолжайте экспериментировать с новыми значениями констант, пока результат не удовлетворит вас.

Заключение

В отличие от ранних текстовых игр, «Ловкач» действительно похож на современную компьютерную игру. Он содержит графику и музыку, а также поддерживает управление мышью. Хотя модуль `pygame` предоставляет функции и типы данных в качестве строительных блоков, именно вы, программист, создаете из них интересные игры.

Вы можете сделать все это, потому что знаете, как поручить задачу компьютеру — шаг за шагом, строка за строкой. Выражаясь на языке компьютера, вы можете заставить его произвести сложные математические расчеты и вывести результат для вас. Это полезное умение, и я надеюсь, что вы продолжите больше узнавать о программировании на Python. (А ведь узнать предстоит еще так много!)

Желаю вам успехов в создании собственных игр!

ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ

A

append(), метод, 128

B

blit(), метод, 336, 406
BMP, формат файла, 370
bool(), функция, 283
break, инструкция, 62, 64, 146
button, атрибут для события MOUSEBUTTONDOWN, 360

C

centerx, атрибут для объекта Rect, 330
centery, атрибут для объекта Rect, 330
choice(), функция из модуля random, 152
circle(), функция модуля pygame.draw, 333
Clock, объект, 382
Clock(), функция, 357
colliderect(), функция, 366, 392
continue, инструкция, 206

D

def, инструкция, 76, 77, 154

E

elif, инструкции, 138
ellipse(), функция модуля pygame.draw, 334
else, инструкция, 89
end='', инструкция, 131
end, параметр для функции print(), 70
Event, объект, 337
exit(), функция модуля sys, 227

F

False (Ложь), логическое значение, 57
fill(), функция, 331
find(), метод, 256
float(), функция, 57
Font, объект, 382, 394
for, инструкция, 51, 53, 54

G

get_rect(), метод, 330
GIF, формат файла, 370

I

IDLE
  интерактивная оболочка, 24
  редактор файлов, 37
if, инструкция, 61, 64
import, инструкция, 48
input(), функция, 41
int(), функция, 55, 180
in, оператор, 127

J

join(), метод, 202

K

KEYDOWN, событие, 359
keys(), метод, 151

L

len(), функция, 86, 149
line(), функция модуля pygame.draw, 333
list(), функция, 131
load(), функция, 376
lower(), метод, 91, 136

M

math, модуль, 227
MOUSEBUTTONUP, событие, 360
MOUSEBUTTON, событие, 360
MOUSEMOTION, событие, 360
move_ip(), метод, 402
move(), метод, 403

N

None, значение, 183
not, оператор, 82

O

or, оператор, 81

P

PixelArray, объект, 335
polygon(), функция модуля pygame.draw, 332
print(), функция, 41
pygame.display, модуль
  set_caption(), функция, 326
  set_mode(), функция, 324
  update(), функция, 331
pygame.draw, модуль
  circle(), функция, 333
  ellipse(), функция, 334
  line(), функция, 333
  polygon(), функция, 332
  rect(), функция, 334
pygame.quit(), функция, 338
pygame.Rect, тип данных, 358
pygame.Rect(), функция, 358
pygame, модуль
  выход из программ, 338
  закрашивание цветом объекта поверхности, 331
  события и цикл игры, 336
  функции рисования
    круг, 333
    линия, 333
    многоугольник, 332
    прямоугольник, 334
    эллипс, 334

Q

QUIT, событие, 359

R

random, модуль
  импорт, 48
range(), функция, 131
remove(), метод, 238
return, инструкция, 84
round(), функция, 309

S

shuffle(), функция модуля random, 192
sort(), метод списка, 201
Sound, объект, 376
Sound(), функция модуля pygame.mixer, 376
split(), метод, 128
sqrt(), функция модуля math, 228
str(), функция, 57, 62
Surface, объект, 326
sys, модуль, 227

T

tick(), метод объекта Clock, 357
time, модуль
  sleep(), функция, 88
  импорт, 76
True (Истина), логическое значение, 57

U

upper(), метод, 91

V

values(), метод, 151

W

while, инструкция, 79, 90

А

Абсолютное значение числа, 215
Алгоритмы, 167
Аргументы, 41
Атрибуты, 330

Б

Бесконечный цикл, 94
>= (больше или равняется), оператор, 58
> (больше чем), оператор, 58

В

Вывод, 64, 65
Вызов
  методов, 127
  функций, 77
Вызов методов, 127
Выполнение программы, 40
Выражения, 26
  оценка выражений, 26
- (вычитание), 25
  расширенное присваивание, 199

Г

Глобальная область, 85
Глобальные переменные, 97

Д

: (двоеточие), в инструкциях for, 54
" (двойные кавычки), 69
Дедуктивная игра «Холодно-горячо», 192
Дедуктивня игра «Холодно-горячо», 193, 195, 196, 197, 208
Декартова сстема координат, 209, 263
/ (деление), 25
  расширенное присваивание, 199

З

= (знак равенства), оператор присваивания, 28
(знак решетки), для комментариев, 41
Значения, 25
  присваивание переменной нового значения, 31
  сохранение значений в переменных, 28

И

Игра «Виселица»
  доработка игры, 147
  проектирование игры с помощью блок-схем, 108
Игра «Крестики-нолики», 160
Игра «Ловкач», 381
Игра «Охота за подводными сокровищами», 218
Игра «Реверси», 263
Игра «Угадай число», 46
Игра «Царство драконов», 73
Игровой цикл, 337
  для игры «Ловкач», 398
  для игры «Реверси», 296
  для программы «Анимация», 346
Индексы
  доступ к элементам списка, 125
  после удаления элементов списка, 154
Инструкции
  управления потоком, 51
Искусственный интеллект, 159, 263
Исходный код, 37
Итерация, 54

К

[] (квадратные скобки), для индексов, 124
Клавиатура
  обработка событий, 398
  пользовательский ввод, 359
Клавишы WASD, 361
Ключи
  для словарей, 149
  для шифров, 249
Комментарии, 40
Конкатенация
  списков, 126
  строк, 35
Константы, 124
Координаты y, 210
Кординаты x, 210
Короткие логические выражения, 180
() (круглые скобки), порядок выполнения операций, 27
Курсор в файловом редакторе, 37

Л

Логические операторы, 80
Логический тип данных, 57
Локальная область, 85
Локальные переменные, 98

М

Математика
выражения, 26, 27
математические хитрости, 214
операторы, 25
синтаксические ошибки, 27
целые числа и числа с плавающей точкой, 25, 27
<= (меньше или равняется), оператор, 58
< (меньше чем), оператор, 58
Множественное присваивание, 156
Мышь
обработка событий, 359

Н

Написание программ, 36
!= (не равняется), оператор, 58
Новая строка (), 70
(обратный слеш), для экранируемых символов, 67

О

Объекты, 326
' (одинарные кавычки), 69
Окно графического интерфейса пользователя, ГИП, 324
Операторы
математические операторы, 25
Опечатки, 30
Ось x, 210
Ось y, 210
Отладчик, 95
Отсупы в коде, 52
Ошибки, 93
поиск ошибок, 100
типы ошибок, 93

П

Параметры функции, 86
Пиксели, 213
окрашивание пикселей, 335
Порядок операций, 26
>>>, приглашение, 29
Программа AISim1, 300
Программа AISim2, 304
Программа AISim3, 310
Программа «Hello World», 37
Программа «Анимация», 339
Программа «Обнаружение столкновений», 353
Программа «Спрайты и звуки», 371
Программа «Шифр Цезаря», 251
Программа «Шутки», 65

Р

== (равняется), оператор, 58
Редактор файлов, 37
Режим полного перебора для шифров, 260

С

Сглаживание, 329
+ (сложение), 25
коммутативное свойство, 214
расширенное присваивание, 199
События, 337
Создание комментариев из кода, 302
Сообщения об ошибке, 94
ImportError, 320
IndexError, 126
NameError, 30
ValueError, 56
синтаксические ошибки, 27
Спрайты, 369
добавление, 375
изменение размера, 376
Срезы
списков, 132
Ссылки на список, 172
Статистические данные, 307
Строки, 34
интерполяция, 204
многострочные, 78
Структуры данных, 227

Т

Таблица истинности
оператор and, 80
оператор not, 82
оператор or, 81

Типы данных, 36
  атрибуты Rect, 330
  логический, 57
  словарь, 149
  списки, 124
  целые числа, 25
Трассировка, 94

У

Удаление элементов списка, 154
* (умножение), 25
  расширенное присваивание, 199
Условия, 59
Установка
  Python, 21

Ф

Файлы изображений, 370
{} (фигурные скобки), в словарях, 149
Функции, 41

Ц

Цвет
  заливка объектов поверхности, 331
  значения RGB в pygame, 326
  пикселей, 335
  текста, 329
Целые числа, 25
Циклы, 51
  вложенные, 206
  выход при помощи инструкции break, 62
  с инструкциями while, 79

Ч

Числа с плавающей точкой, 25, 27

Ш

Шрифты, 327

Э

Экранируемые символы, 67

Все права защищены. Книга или любая ее часть не может быть скопирована, воспроизведена в электронной или механической форме, в виде фотокопии, записи в память ЭВМ, репродукции или каким-либо иным способом, а также использована в любой информационной системе без получения разрешения от издателя. Копирование, воспроизведение и иное использование книги или ее части без согласия издателя является незаконным и влечет уголовную, административную и гражданскую ответственность.

Пособие для развивающего обучения

МИРОВОЙ КОМПЬЮТЕРНЫЙ БЕСТСЕЛЛЕР

Эл Свейгарт

УЧИМ PYTHON, ДЕЛАЯ КРУТЫЕ ИГРЫ

Директор редакции *Е. Капьёв*
Ответственный редактор *Е. Истомина*
Младший редактор *Е. Минина*
Художественный редактор *А. Гусев*

ООО «Издательство «Эксмо»
123308, Москва, ул. Зорге, д. 1. Тел.: 8 (495) 411-68-86.
Home page: **www.eksmo.ru** E-mail: **info@eksmo.ru**

Өндіруші: «ЭКСМО» АҚБ Баспасы, 123308, Мәскеу, Ресей, Зорге көшесі, 1 үй.
Тел.: 8 (495) 411-68-86.
Home page: www.eksmo.ru E-mail: info@eksmo.ru.
Тауар белгісі: «Эксмо»
Қазақстан Республикасында дистрибьютор және өнім бойынша
арыз-талаптарды қабылдаушының
өкілі «РДЦ-Алматы» ЖШС, Алматы қ., Домбровский көш., 3«а», литер Б, офис 1.
Тел.: 8 (727) 251-59-90/91/92; E-mail: RDC-Almaty@eksmo.kz
Интернет-магазин: www.book24.kz
Өнімнің жарамдылық мерзімі шектелмеген.
Сертификация туралы ақпарат сайтта: www.eksmo.ru/certification

Оптовая торговля книгами «Эксмо»:
ООО «ТД «Эксмо». 142700, Московская обл., Ленинский р-н, г. Видное,
Белокаменное ш., д. 1, многоканальный тел.: 411-50-74.
E-mail: **reception@eksmo-sale.ru**

По вопросам приобретения книг «Эксмо» зарубежными оптовыми
покупателями *обращаться в отдел зарубежных продаж ТД «Эксмо»*
E-mail: **international@eksmo-sale.ru**

International Sales: International wholesale customers should contact
Foreign Sales Department of Trading House «Eksmo» for their orders.
international@eksmo-sale.ru

По вопросам заказа книг корпоративным клиентам, в том числе в специальном
оформлении, *обращаться по тел.: +7 (495) 411-68-59, доб. 2261.*
E-mail: **ivanova.ey@eksmo.ru**

Сведения о подтверждении соответствия издания согласно законодательству РФ о техническом регулировании можно получить по адресу: http://eksmo.ru/certification/

Өндірген мемлекет: Ресей. Сертификация қарастырылмаған

Подписано в печать 14.03.2018. Формат 70x100 1/16.
Печать офсетная. Усл. печ. л. 33,7.
Тираж экз. Заказ

Знак информационной продукции
12+
согласно Федеральному закону от 29.12.2010 № 436-ФЗ

ISBN 978-5-699-99572-1
9 785699 995721 >

BOOK24.RU
ИНТЕРНЕТ-МАГАЗИН

В электронном виде книги издательства вы можете купить на www.litres.ru
ЛитРес: один клик до книг

КОГДА ВЫ ДАРИТЕ КНИГУ, ВЫ ДАРИТЕ ЦЕЛЫЙ МИР

ХОТИТЕ ЗНАТЬ БОЛЬШЕ?

Заходите на сайт:
https://eksmo.ru/b2b/

Звоните по телефону:
+7 495 411-68-59, доб. 2261

ВАШ ЛОГОТИП НА ОБЛОЖКЕ

ВАШ ЛОГОТИП НА КОРЕШКЕ

ОБРАЩЕНИЕ К КЛИЕНТАМ НА ОБЛОЖКЕ

4-е издание

Не играй в игры – создавай их!

Эта книга научит вас разрабатывать компьютерные игры при помощи популярного языка программирования Python – даже если вы никогда раньше не занимались программированием! Начните с создания таких простых игр, как «Виселица» и «Крестики-нолики», а затем перейдите к продвинутым анимированным и озвученным играм.

В процессе вы изучите ключевые концепции программирования и узнаете, как:

- **Использовать циклы, переменные и инструкции**
- **Выбирать правильные структуры данных, такие как списки, словари и кортежи**
- **Делать игры с графикой и анимацией, используя модуль pygame**
- **Использовать криптографию для преобразования текстовых сообщений в секретный код**
- **Отлаживать программы и находить распространенные ошибки**

В книге доступно описаны первые шаги для тех, кто интересуется программированием, но не знает, с чего начать. На примере простых, минималистичных игр объясняются основные концепты Python 3 и программирования в целом. Читателя постепенно вводят в курс дела, выдавая за раз ровно столько информации, сколько необходимо для понимания рассматриваемой программы. Если вы хотите начать писать свою первую программу уже сегодня, эта книга – отличный выбор.

Константин Ильченко,
руководитель отдела разработки Рамблер/Игры

no starch press
www.nostarch.com

ISBN 978-5-699-99572-1
9 785699 995721 >

БОМБОРА