Создание видеоигр с помощью PyGame

Майк Голд

Создание видеоигр с помощью PyGame

С пошаговыми примерами

Майк Голд

2023

Оглавление

Настройка Python и Pygame

Добро пожаловать в мир программирования PyGame и Python! Эта книга познакомит вас с библиотекой PyGame и научит вас создавать собственные игры с использованием языка Python. Мы начнем с базового обзора Python и библиотеки PyGame, а затем перейдем к разработке, написанию и отладке нашей собственной игры. От добавления графики и звуков до создания анимации и бонусов — мы расскажем обо всем, что вам нужно знать, чтобы создать собственную насыщенную интерактивную игру. Наконец, мы пройдем процесс отладки и тестирования нашей игры, прежде чем опубликовать ее для всего мира. Итак, давайте начнем и научимся создавать собственные игры с помощью PyGame и Python!

Начало старта

Установка Python

Вы можете найти последнюю версию Python на сайте Python.org[1]. Доступны как 32-битные, так и 64-битные версии. После того, как вы нажали кнопку «Загрузить», запустите загруженный исполняемый файл, следуя инструкциям, чтобы установить последнюю версию Python на свой компьютер.

[1] https://www.python.org/downloads/

Установка VSCode

Visual Studio Code доступен для операционных систем Windows, MacOS, Linux. Вы можете загрузить код Visual Studio с https://code.visualstudio.com/download. Выберите соответствующую загрузку для вашей ОС и запустите установку. После того, как вы установили Visual Studio Code, вам нужно установить расширения Python и Pylance.

Расширение Python:

Расширение Python для Visual Studio Code предоставляет широкий спектр функций, упрощающих разработку Python в VS Code, включая анализ кода, отладку, завершение кода IntelliSense, форматирование кода, рефакторинг, модульное тестирование и многое другое. Расширение имеет открытый исходный код и доступно бесплатно, и его можно установить, выполнив поиск на рынке расширений VS Code. С расширением Python разработчики могут быстро и легко создавать свои проекты Python и управлять ими, а также использовать широкий спектр расширенных функций.

Расширение Pylance:

Pylance — это расширение Visual Studio Code, обеспечивающее расширенную поддержку языка Python, включая быстрый многофункциональный IntelliSense, линтинг, анализ всего проекта и отладку. Pylance использует протокол языкового сервера (LSP) для связи с языковым сервером и поддерживает широкий спектр функций, таких как автозаполнение, рефакторинг кода, навигация по коду и диагностика ошибок. Pylance также предоставляет функцию автоматического импорта, которая может автоматически добавлять импорт для символов, когда вы вводите их в свой код. Pylance — отличный инструмент для разработчиков Python, позволяющий быстро и эффективно писать код.

Чтобы установить расширения, перейдите к символу расширений на левой панели Visual Studio Code и выполните поиск Pylance на торговой площадке. Нажмите на него и установите расширение в код VisualStudio. Также найдите расширение под названием Python и установите его.

Установка Pygame

Pygame — это библиотека с открытым исходным кодом для создания игр на Python. Она имеет широкий спектр возможностей и функций, которые облегчают начало создания игр.

Вы можете найти документацию по Pygame на pygame.org[2].

Чтобы начать использовать Pygame, вам необходимо установить его. Самый простой способ установить pygame — с терминала внутри VSCode. Щелкните терминал в верхней части меню и введите следующую строку:

```
pip  install  pygame
```

если у вас еще не установлен pip, вам нужно будет перейти на https://bootstrap.pypa.io/get-pip.py и загрузить файл в каталог вашего приложения python. Чтобы выяснить, где установлен python, вы можете спросить python! Перейдите к терминалу в Visual Code и введите

```
1 python
```

Вы увидите подсказку >>>. Вставьте следующий код

```
1
2 >>> import os
3 >>> import sys
>>> os.path.dirname(sys.executable)
```

Это выдаст путь, по которому вы разместите файл get-pip.py.

например C:\Python310 в Windows

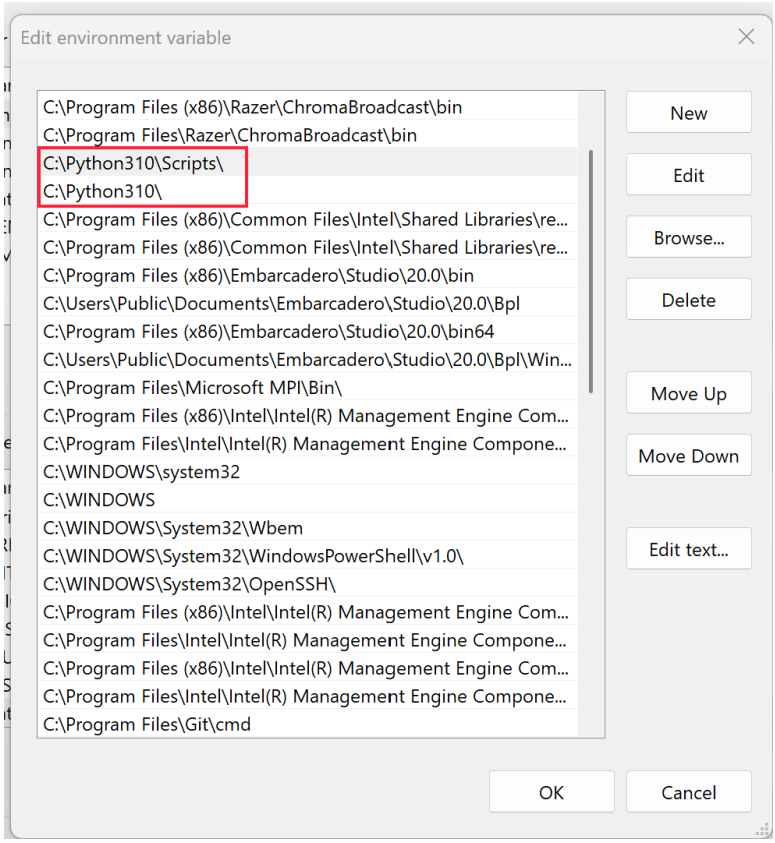
Поместите get-pip.py в указанный путь, а затем запустите

[2] http://www.pygame.org/docs/

```
1  py get-pip.py
```

⚠ Примечание. Возможно, вам потребуется добавить путь python к пути к переменным среды.

Два пути, которые у меня есть, показаны ниже



Введение в Python

В следующих главах мы будем программировать на Python, поэтому нам нужно дать вам основу для понимания языковых конструкций, которые мы будем использовать, а также того, как их запускать. В следующей главе вы узнаете, как использовать наиболее распространенные части языка и что мы будем использовать для создания нашей игры. Сначала давайте ответим на несколько распространенных вопросов о Python.

История Python

Python был создан Гвидо ван Россумом и впервые выпущен в 1991 году. Python — это высокоуровневый интерпретируемый язык программирования общего назначения. Python стал популярным благодаря понятному синтаксису и удобочитаемости. Python также поддерживает модули и пакеты, что позволяет повторно использовать код.

Python является интерпретируемым языком, что означает, что он компилируется во время выполнения. Это позволяет коду Python быть более терпимым к ошибкам и упрощает отладку. Python также поддерживает ряд систем и сред с открытым исходным кодом, таких как Django и Flask.

Python часто используется для научных вычислений, веб-разработки, машинного обучения и автоматизации. Python имеет большое и активное сообщество, что упрощает поиск помощи и поддержки в Интернете. Python используется такими организациями, как Google, Yahoo и NASA.

Что отличает Python от других языков?

Python — это интерпретируемый язык, поэтому начать работу с ним проще, чем с другими языками, такими как C или Java. Он также динамически типизирован, то есть вам не нужно объявлять тип при создании переменной. Это делает язык более выразительным и может уменьшить сложность некоторых приложений. Python также обладает высокой расширяемостью, что означает, что его можно расширять за счет существующих библиотек и новых модулей, написанных на C, C++ или других языках. Кроме того, синтаксис Python относительно прост и легок в освоении.

Какие типы приложений создаются с помощью Python?

Python используется в самых разных приложениях, включая настольные приложения с графическим интерфейсом пользователя, веб-приложения, разработку программного обеспечения, научные и числовые вычисления, а также искусственный интеллект и машинное обучение. Многие из самых популярных веб-сайтов и сервисов, таких как YouTube, Instagram, Quora и Dropbox, были созданы с использованием Python.

Почему я должен изучать Python?

Как уже говорилось, Python — это мощный и универсальный язык программирования с широким спектром приложений и вариантов использования. Он прост в освоении и имеет высокий уровень удобочитаемости, что делает его отличным выбором для начинающих, но он также популярен среди опытных разработчиков. Это универсальный язык, то есть его можно использовать для самых разных задач — от веб-разработки до науки о данных и машинного обучения. Python также имеет сильное сообщество разработчиков и пользователей, поэтому всегда есть поддержка и новые инструменты. Кроме того,

Python — это язык с открытым исходным кодом, что означает, что его можно использовать бесплатно и он доступен для всех, у кого есть доступ в Интернет.

Теперь, когда вы немного знаете об этом языке, давайте создадим нашу первую программу на Python, просто чтобы намочить ноги. Мы собираемся прыгнуть в python.

Давайте начнем с простой программы, которая печатает Hello World:

Создайте новую папку в VSCode под названием HelloWorld. Затем создайте новый файл с именем HelloWorld.py и добавьте следующую строку:

```
1 print("Hello World")
```

Сохраните файл. Перейдите к своему терминалу в VSCode (терминал bash) и запустите python с помощью следующей команды:

```
1 py -m HelloWorld
```

Вы должны увидеть следующий вывод в окне терминала:

```
1 Hello World
```

Неплохо! Если вы зашли так далеко, вы готовы к работе. Давайте немного поднимем циферблат. Давайте напишем программу, которая напишет Hello World 10 раз. Для этого мы будем использовать цикл for. Цикл for позволяет нам перебирать диапазон значений и каждый раз в цикле выводить «Hello World». Измените свой файл HelloWorld.py на приведенный ниже код и запустите.

```
for number in range(5):
1     print ('Hello World')
2
```

Эта программа производит следующий вывод

```
Hello World
Hello World
Hello World
Hello World
Hello World
```

Обратите внимание, что внутри нашего цикла for есть число. Каждый раз, когда число проходит через цикл, оно увеличивается до следующего числа. Мы можем показать это в нашем операторе печати, используя интерполяцию строк. Измените HelloWorld.py на этот код:

```
for number in range(5):
    print (f'Hello World #{number}')
```

Эта программа производит этот вывод после запуска:

```
Hello World #0
Hello World #1
Hello World #2
Hello World #3
Hello World #4
```

Обратите внимание, что функция диапазона начинается с 0 и заканчивается на 4, а не на 5. Если бы мы хотели, чтобы наш hello world считался до пяти, мы могли бы просто добавить единицу к числу.

```
for number in range(5):
    print (f'Hello World #{number+1}')
```

Он производит выходные данные, которые нумеруют Hello World 1-5:

```
Hello World #1
Hello World #2
Hello World #3
Hello World #4
Hello World #5
```

Оператор если

Что, если бы мы хотели распечатать только «Hello World's»? Теперь мы можем ввести оператор if, который позволяет нам принимать решения о том, какой из Hello Worlds будет напечатан.

```
for number in range(5):
    numberToPrint = number + 1
    if numberToPrint % 2 == 0:
       print (f'Even Hello World #{numberToPrint}')
```

Этот код вводит оператор if для принятия решений. В этом случае оператор if использует функцию модификатора (%), чтобы определить, есть ли какие-либо остатки при делении следующего числа на 2. Если остаток от деления numberToPrint на 2 равен нулю, печать будет выполнена. Так, например, 2 % 2 не имеет остатков, поэтому он проходит мод-тест numberToPrint % 2 == 0 и напечатает Even Hello World #2. С другой стороны, 5 % 2 равно 1, поэтому оно не проходит тест на равенство 0, поскольку 0 не равно 1. Печать будет пропущена для 5.

Таким образом, после запуска программы код напечатает «Even Hello World #2», «Even Hello World #4». Он не будет печатать «Even Hello World #1», «Even Hello World #3» и «Even Hello World #5», поскольку ни одно из этих чисел не является четным и не соответствует критериям функции mod.

```
Even Hello World #2
Even Hello World #4
```

Оператор else

если мы хотим получить более полный ответ на распечатку нашего четного числа, мы также можем напечатать, является ли число четным или нечетным, мы будем использовать оператор else, чтобы помочь нам здесь:

```
for number in range(5):
    numberToPrint = number + 1
    if (numberToPrint) % 2 == 0:
       print (f'Even Hello World #{numberToPrint}')
    else:
       print (f'Odd Hello World #{numberToPrint}')
```

Оператор else выполняется, когда условие в операторе if ложно. Он используется для выполнения другого кода, когда условие не выполняется. В приведенном выше примере оператор else выводит сообщение со словом Odd Hello World #{numberToPrint}, если число нечетное.

```
Odd Hello World #1
Even Hello World #2
Odd Hello World #3
Even Hello World #4
Odd Hello World #5
```

elif

В Python оператор elif (сокращение от «else if») — это условный оператор, который позволяет вам проверять несколько выражений на TRUE - ИСТИНА и выполнять блок кода, как только одно из условий оценивается как TRUE. Оператор elif следует тому же синтаксису, что и оператор if, но с одним дополнительным ключевым словом: elif.

Например, следующий код проверит, делится ли numberToPrint на 3, а если нет, то проверит, является ли numberToPrint четным. Если ни одно из них не верно, сработает else и напечатает, что оно не четное и не делится на 3:

```
for number in range(5):
    numberToPrint = number + 1
    if numberToPrint % 3 == 0:
       print (f'{numberToPrint} is divisible by 3')
    elif numberToPrint % 2 == 0:
       print (f'{numberToPrint} is even')
    else:
       print (f'{numberToPrint}
        Not even and not divisible by 3')
```

Вот вывод кода, иллюстрирующий, как работает if else:

```
1 Не четное и не делится на 3
2 четно
3 делится на 3
4 четно
5 Не четное и не делится на 3
```

Цикл while

Цикл while дает нам большую гибкость при переборе данных:

Иногда это дает нам слишком много гибкости! Следующий цикл while будет работать вечно:

```
while True:
  print('Hello World')
```

Здесь условие всегда истинно, поэтому оно никогда не завершит цикл. Циклы while заканчиваются, когда условие после while становится ложным. Другой способ выйти из цикла — использовать оператор break:

```
while True:
  print('Hello World')
  break
```

Выход для этого цикла:

```
Hello World
```

потому что программа по-прежнему войдет в цикл while и напечатает «Hello World», но сразу после того, как она нажмет оператор печати, она нажмет на разрыв, что приведет к выходу из цикла.

Мы можем показать мощь цикла while, переписав цикл for выше:

```
number = 1
while number <= 5:
    print (f'Hello World #{number}')
    number = number + 1
```

Цикл while начинается с числа, равного 1, поэтому условие защиты цикла истинно, поскольку 1 меньше или равно 5. Оператор печати выполняется с 1, а затем число увеличивается на 1, чтобы стать 2. Программа будет затем вернитесь назад, и, поскольку 2 все еще меньше или равно 5, он снова выполнит печать и увеличит число до 3. Он будет продолжать делать это до тех пор, пока он больше не будет соответствовать условию защиты. Когда число равно 6, оно больше не меньше или равно пяти, поэтому оно вырвется из цикла. Вот результаты работы этой программы:

```
Hello World #1
Hello World #2
Hello World #3
Hello World #4
Hello World #5
```

Списки Python

Списки Python — это структуры данных, в которых хранится набор элементов, которые могут относиться к разным типам данных. Доступ к элементам списка осуществляется с помощью индекса. Списки позволяют эффективно хранить данные, позволяя быстро извлекать и изменять хранящиеся в них данные. Списки можно использовать для различных задач, таких как сортировка, поиск и обработка данных.

Мы можем сделать то же самое, что и в наших предыдущих циклах, перебирая список:

```
numbers = [1, 2, 3, 4, 5]
for number in numbers:
    print (f'Hello World #{number}')
```

числа — это список элементов, пронумерованных от 1 до 5. Мы перебираем числа так же, как мы перебирали значения. возвращается из диапазона function.outputs:

```
Hello World #1
Hello World #2
Hello World #3
Hello World #4
Hello World #5
```

Добавление, вставка и удаление из списка

В Python списки можно изменять с помощью методов append (добавления), insert (вставки) и remove (удаления). Метод append добавляет элемент в конец списка, метод insert добавляет элемент по указанному индексу, а метод remove удаляет элемент по указанному индексу. Например, следующий код добавит элемент «c» в конец списка, добавит элемент «d» в индекс 2 и удалит элемент со значением «a»:

```
letters = ['a', 'b', 'e']
print(letters, 'starting letters')
letters.append('c')
print('append c: ',letters)
letters.insert(2, 'd')
print('insert d at position 2: ', letters)
letters.remove('a')
print('remove a: ',letters)
```

output:

```
starting letters:  ['a', 'b', 'e']
append c:  ['a', 'b', 'e', 'c']
insert d at position 2:  ['a', 'b', 'd', 'e', 'c']
remove a:  ['b', 'd', 'e', 'c']
```

Эти методы очень полезны, когда вы хотите изменить список, не создавая новый список с нуля.

Двумерный список

Двумерный список в Python — это тип структуры данных, в которой данные хранятся в нескольких измерениях.[1] По сути, это список списков, где каждый внутренний список представляет собой строку данных, а каждый внешний список — столбец данных. Двумерные списки удобны для данных, организованных в строки и столбцы, таких как таблицы, электронные таблицы и матрицы. Чтобы получить доступ к данным в двумерном списке, вам нужно сослаться на индекс строки и столбца. Например, чтобы получить доступ к элементу в третьей строке и четвертом столбце двумерного списка, вы можете использовать следующий код:

[1]Это хорошо описано при переполнении стека: https://stackoverflow.com/questions/2397141/how-to-initialize-a-two-dimensional-array-in-python

```
my_list = [[1,2,3,4], [5,6,7,8], [9,10,11,12]]

value = my_list[2][3]
print(value)
```

вывод для этого кода:

```
12
```

Двенадцать расположены в третьем массиве, в 4-м элементе этого массива (помните, что списки индексируются, начиная с 0, поэтому мы начинаем считать с 0)

```
value = my_list[0][0]
print(value)
```

Приведенный выше код выводит 1, поскольку 1 находится в индексе 0, который является первым списком в массиве списков и 0-м элементом в этом первом массиве.

Другой способ представить двумерные массивы — это строки и столбцы. Этот код можно рассматривать как доступ к числу 1, которое появляется в первой строке и первом столбце двумерного массива. Переменная my_list представляет собой двумерный массив, и два индекса используются для указания строки и столбца элемента, к которому мы хотим получить доступ. Первый индекс (0) указывает нулевую строку, состоящую из [1,2,3,4]. Второй индекс (0) указывает нулевой столбец. Код выводит значение элемента в строке 0, столбце 0, которое в данном случае будет равно 1.

Понимание списка

Понимание списков — это краткий способ создания списков в Python. Это синтаксическая конструкция, которая позволяет вам создать новый список, указав элементы, которые вы хотите включить, часто производные от существующих списков или других итерируемых объектов. Включение списков также может включать необязательные условия для фильтрации элементов. Они более читабельны и часто быстрее, чем эквивалентный код, использующий циклы.

Сопоставление списка

Предположим, у вас есть список чисел, и вы хотите создать новый список с квадратами этих чисел. Использование понимания списка:

```
numbers = [1, 2, 3, 4, 5]

squares = [x**2 for x in numbers]
print(squares)
```

output: [1, 4, 9, 16, 25]

Использование генератора списков для создания нового списка путем возведения в квадрат каждого числа из существующего списка эквивалентно перебору чисел в цикле, возведению их в квадрат и добавлению результата в новый список, но синтаксис спискового понимания обеспечивает более компактный и лаконичный синтаксис.

Фильтрация списка

Если вы хотите создать новый список, содержащий только четные числа из исходного списка, вы можете добавить условие фильтрации в понимание списка. Ниже приведен пример извлечения всех четных чисел из списка чисел.

```
even_numbers = [x for x in numbers if x % 2 == 0]
print(even_numbers)
```

output: [2, 4]

Понимание списка применяет фильтр к списку чисел с использованием условия if x % 2 == 0 для выборочного включения только четных чисел в новый список even_numbers. Любые числа, не удовлетворяющие условию, исключаются из результирующего списка.

Двумерные списки

Понимание списков также можно использовать для создания или изменения двумерных списков. Например, если у вас есть двумерный список (матрица) и вы хотите создать его транспонирование, вы можете использовать вложенные включения списка:

```
matrix = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]

transpose = [[row[i] for row in matrix] for i in range(le\
n(matrix[0]))]
print(transpose)
```

output: [[1, 4, 7], [2, 5, 8], [3, 6, 9]]

В этом примере обработка внешнего списка выполняет итерацию по индексам столбцов, а обработка внутреннего списка — по строкам. Результирующий список транспонирования представляет собой новый 2D-список, содержащий транспонированные элементы исходной матрицы.

Функции

В Python функции — это части повторно используемого кода, которые могут принимать одно или несколько входных значений, выполнять над ними некоторые операции и возвращать результат. Функции используются для того, чтобы сделать код более организованным, читабельным и пригодным для повторного использования. Функции можно определять с помощью ключевого слова def, и они могут принимать любое количество параметров. Например, следующая функция принимает два параметра, x и y, и возвращает их сумму:

```
def add(x, y):
    return x + y

sum = add(2, 3)
print(sum) # 5
```

Функции также могут принимать необязательные параметры, то есть параметры со значениями по умолчанию. Например, следующая функция принимает два параметра, x и y, и необязательный параметр с именем z, значение которого по умолчанию равно 0:

```
def add(x, y, z=0):
    return x + y + z

sum = add(2, 3)
print(sum) # 5
```

Python также поддерживает анонимные функции, которые не имеют имени и определяются с помощью ключевого слова lambda. Анонимные функции могут принимать любое количество параметров и всегда возвращают одно выражение. Ниже приведена анонимная функция для сложения двух чисел:

```
sum = lambda a, b: a + b
```

Мы можем вызвать лямбда-функцию, просто вызвав назначенную ей переменную:

```
  result = sum(4,5)
  print('4 + 5 = ', result)
```

Функции также могут иметь аргументы переменной длины, что позволяет им принимать любое количество аргументов. Эти аргументы хранятся в кортеже. Например, следующая функция принимает любое количество аргументов и выводит их:

```
def print_args(*args):
    for arg in args:
        print(arg)

print_args('a', 'b', 'c', 'd')
```

Вывод этой функции будет:

```
a b c d
```

Кроме того, функции также могут возвращать несколько значений. Вместо того, чтобы возвращать одно значение, вы можете вернуть кортеж, содержащий несколько значений. Это позволяет вам возвращать несколько фрагментов данных из одного вызова функции. Например, следующая функция возвращает два значения:

```
def get_info():
    name = "John"
    age = 25
    return (name, age)

name, age = get_info()
print(name) # John
print(age) # 25
```

Функции — мощный и универсальный инструмент в Python, и их можно использовать для написания более организованного, читаемого и пригодного для повторного использования кода. С помощью функций вы можете разбить свой код на более мелкие фрагменты и организовать их в осмысленные, автономные единицы. Это позволяет вам легко повторно использовать код в разных частях вашей программы, а также значительно упрощает отладку и устранение неполадок. Кроме того, функции также можно использовать для упрощения сложной логики и повышения удобочитаемости кода. Наконец, функции также можно использовать для повышения производительности, позволяя выполнять код параллельно, что может ускорить время выполнения.

Кортежи

В Python кортежи — это неизменяемые последовательности объектов. Обычно они используются для хранения связанных частей информации, таких как координаты или запись данных. Кортежи создаются с помощью круглых скобок и могут содержать объекты любого типа, включая другие кортежи. Кортежи также можно использовать для возврата нескольких значений из функции. Например, вы можете использовать следующий код для возврата двух значений из функции:

```
def get_info():
    name = "John"
    age = 25
    return (name, age)


name, age = get_info()
print(name) # John
print(age) # 25
```

Кортежи также полезны для группировки связанных данных. Например, если у вас есть список координат, вы можете использовать кортеж для хранения каждой координаты в одном месте.

Кортежи также являются отличным способом создания эффективного и безопасного словаря. Когда вы создаете словарь с кортежами, порядок элементов в каждом кортеже будет определять порядок ключей в словаре. Это может помочь вам избежать случайной перезаписи или удаления данных.

Классы

Классы в Python подобны шаблонам для создания объектов. Они являются основными строительными блоками для любого объектно-ориентированного языка программирования. Класс определяет свойства, поведение и атрибуты объекта. Классы также предоставляют методы, которые представляют собой функции, воздействующие на данные в классе.

Классы обычно используются для создания объектов, которые представляют объекты реального мира, такие как автомобиль или человек. Объекты, созданные из классов, могут иметь свои собственные значения и методы, которые можно использовать для выполнения задач. Вот простой пример класса Person в python:

```
class Person:
  def __init__(self, name, age, gender):
    self.name = name
    self.age = age
    self.gender = gender

  def introduce(self):
    print(f'Hello, my name is {self.name}')
```

Класс Person определяет атрибуты и функции, которые может выполнять человек. Ниже приведен пример создания нового объекта с именем tim из класса Person. После того, как мы создадим tim, мы можем вызвать метод внедрения для tim.

```
tim = Person("Tim", 28, "Male")
tim.introduce()
```

вывод для вызова Introduction показан ниже:

```
Hello, my name is Tim
```

Метод введения использует имя, которое мы использовали для создания Person tim. Если бы мы хотели сконструировать человека по имени Мэри, мы бы сделали следующее:

```
mary = Person("Mary", 30, "Female")
```

mary — это еще один экземпляр Person, как и tim, но Mary даст другой ответ, если представится:

```
mary.introduce()
```

output:

```
Hello, my name is Mary
```

Мы будем широко использовать классы при создании PyGame, потому что играми намного проще управлять, когда они организованы в классы.

Вот несколько встроенных классов, используемых в PyGame:

pygame.Surface

pygame.Surface — это базовый класс в библиотеке pygame, представляющий прямоугольную двумерную область в памяти, где вы можете рисовать, манипулировать и хранить пиксельные данные. Поверхности используются для различных графических операций в pygame, таких как рендеринг изображений, текста и фигур.

pygame.sprite.Sprite

Это базовый класс для видимых игровых объектов. Все видимые игровые объекты являются производными от этого класса.

pygame.rect.Rect

Этот класс используется для определения прямоугольной области экрана. Он используется для хранения и управления размером, положением и местоположением прямоугольной области. Он используется для определения столкновений между объектами.

pygame.time.Clock

Этот класс используется для управления временем и игровыми циклами. Это поможет вам следить за временем и управлять частотой кадров в игре.

pygame.font.Font

Этот класс используется для отображения текста на экране. Он помогает вам визуализировать текст с указанным шрифтом, размером, стилем и цветом.

Введение в PyGame

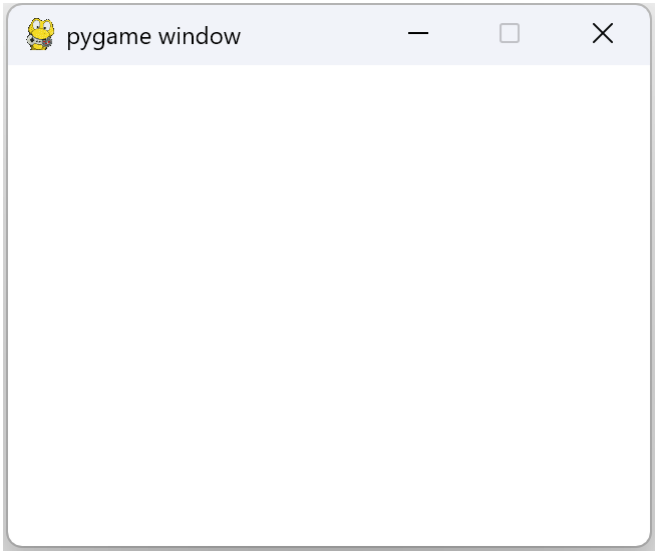
Pygame — это модуль Python, разработанный специально для разработки игр. Он предоставляет набор инструментов и библиотек для создания игр и мультимедийных приложений. Pygame был впервые создан Питом Шиннерсом в 2000 году как побочный проект для изучения мультимедийных возможностей Python. Он выпустил первую версию Pygame в марте 2000 года, которая включала базовые функции, такие как загрузка изображений, воспроизведение звука и обработка событий. С годами Pygame росла и развивалась вместе с языком Python, добавляя новые функции и возможности. В 2007 году сообщество Pygame создало подмножество Pygame для Android, которое позволило запускать приложения Pygame на устройствах Android. Сегодня Pygame широко используется как разработчиками игр, так и энтузиастами, и его популярность продолжает расти, поскольку Python становится все более популярным в качестве языка программирования для разработки игр.

Чтобы начать работу с PyGame, вы можете использовать следующий код, чтобы заполнить окно размером 320x240 пикселей белым фоном:

```
import pygame

# Инициализировать игру

pygame.init()

# Create the screen
gamewindow = pygame.display.set_mode((320, 240))

WHITE = (255, 255, 255)

while True:
    for event in pygame.event.get():
```

```
14          if event.type == pygame.QUIT:
15              sys.exit()
16      gamewindow.fill(WHITE)
17      pygame.display.flip()
```

pygame.init() инициализирует pygame, а pygame.display дает нам доступ к окну игры. в то время как True: это наш игровой цикл, который зацикливается на нашей игре навсегда или до тех пор, пока кто-то не закроет экран pygame, что вызовет событие выхода. Функция pygame.display.flip обновляет содержимое всего экрана. Обычно он используется после рисования или обновления отображения, чтобы убедиться, что изменения видны. Эта функция также известна как «переворот экрана» или «переворот страницы», поскольку она меняет местами передний и задний буферы, содержащие содержимое дисплея.



Hello World

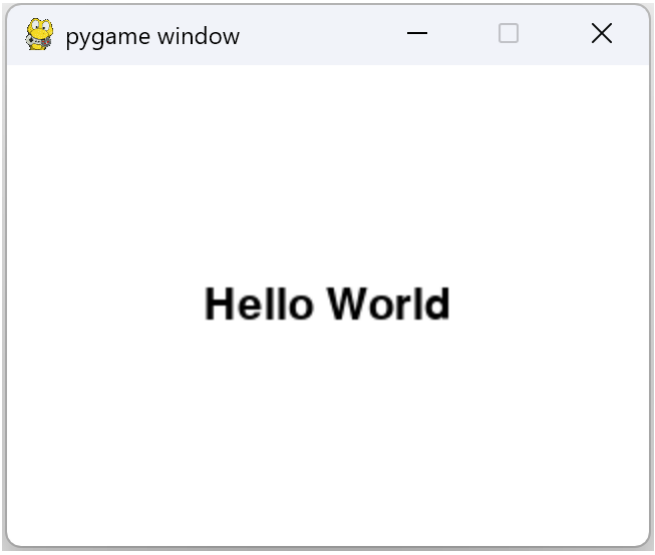
Чтобы добавить текст hello world на пустой экран, нам нужно создать объект шрифта. Строка font = pygame.font.Font(None, 32) создаетобъект шрифта размером 32. Затем этот объект шрифта можно использовать для отображения текста на поверхности дисплея. Аргумент None указывает, что следует использовать шрифт по умолчанию. Если передан файл шрифта, то вместо него будет использоваться этот шрифт. Размер шрифта указывается в пунктах, где 1 пункт равен 1/72 дюйма.

Функция font.render() используется для отображения текста на поверхности дисплея. Он принимает три аргумента: отображаемый текст, необходимость сглаживания текста и цвет текста. В этом случае функция font.render() используется для рендеринга текста «Hello World» с включенным сглаживанием и черным цветом. Визуализированный текст затем сохраняется в текстовой поверхности.

Чтобы поместить текст на экран, нам сначала нужно наложить текстовый объект на поверхность экрана.Мы будем использовать функцию наложения, чтобы нарисовать поверхность текста на экране в центре. Функция screen.blit() принимает два аргумента: поверхность, которую нужно нарисовать, и положение, в котором она должна быть нарисована. Первый аргумент в этом случае — это поверхность текста, которая была создана в предыдущем файле font.render. Второй аргумент — это кортеж, содержащий координаты центра экрана, который вычисляется путем вычитания половины ширины и высоты текстовой поверхности из ширины и высоты экрана. Это позволяет центрировать текст на экране.

```
import pygame

# Инициализировать игру

pygame.init()

# Создайте экран
screen = pygame.display.set_mode((320, 240))
WHITE = (255, 255, 255)
BLACK = (0, 0, 0)

# создать объект шрифта
font = pygame.font.Font(None, 32)

while True:
    screen.fill(WHITE)  # заполнить фон

    #  проверить событие выхода
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            sys.exit()

    # создать текстовую поверхность
    text = font.render("Hello World", True, BLACK)

    #  переместить текстовую поверхность в центр экрана
    screen.blit(text, ((screen.get_width() -
           text.get_width())/2,
           (screen.get_height() - text.get_height()) / 2))

    # обновить экран
    pygame.display.flip()
```

Этот код создаст следующее окно вывода:



Мигание Hello World

Допустим, мы хотим мигать Hello World черным и красным каждую секунду. Мы могли бы добавить временную задержку в игровой цикл и чередовать текстовые объекты разного цвета. Здесь мы используем список Python для чередования цветов при создании текстового объекта:

```
import pygame

# Инициализировать игру

pygame.init()

# Создайте экран
screen = pygame.display.set_mode((320, 240))
WHITE = (255, 255, 255)
BLACK = (0, 0, 0)
```

```
RED = (255, 0, 0)

HelloWorldColors = [BLACK, RED]

# создать объект шрифта
font = pygame.font.Font(None, 32)

count = 0
while True:
    count = count + 1
    screen.fill(WHITE)  # заполнить фон

    # проверить событие выхода
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            sys.exit()

    #создайте поверхность текста и чередуйте
    цвет,
    # используя RED - КРАСНЫЙ ИЛИ BLACK
    -ЧЕРНЫЙ
    text = font.render("Hello World", True,
              HelloWorldColors[count % 2])

    ### подождите 1 секунду (или 1000 миллисекунд)
    pygame.time.delay(1000)

    # перенесите текстовую поверхность на экран
    screen.blit(text, ((screen.get_width() -
                        text.get_width())/2,
                        (screen.get_height() -
                          text.get_height()) / 2))

    # обновить экран
    pygame.display.flip()
```

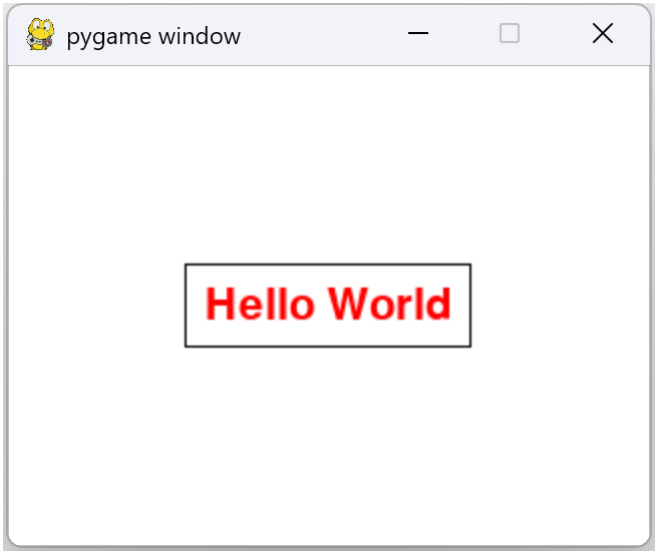
Окружение бордюром

Что, если мы захотим поместить черную рамку вокруг blinkin hello world? Код, показанный ниже, используется для рисования черной прямоугольной рамки вокруг текстовой поверхности. Функция pygame.draw.rect() принимает пять аргументов: поверхность дисплея для рисования, цвет прямоугольника, координаты верхнего левого угла прямоугольника, ширину и высоту прямоугольника и толщину линии прямоугольника. В этом случае верхний левый угол прямоугольника вычисляется путем вычитания 10 из координат центра экрана, а ширина и высота прямоугольника вычисляются путем прибавления 20 к ширине и высоте текстовой поверхности. Аргумент 1 указывает, что линия должна иметь толщину 1 пиксель.

```
while True:
    count = count + 1
    screen.fill(WHITE)  # заполнить фон

    # проверить событие выхода
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            sys.exit()

    # создать текстовую поверхность
    text = font.render("Hello World", True,
                HelloWorldColors[count % 2])
    pygame.time.delay(1000)

    #объемный текст с черной прямоугольной рамкой
    pygame.draw.rect(screen, BLACK,
                ((screen.get_width() -
                text.get_width())/2 - 10,
                (screen.get_height() -
                text.get_height()) / 2 - 10,
                text.get_width() + 20,
```

```
                text.get_height() + 20), 1)

    #перенесите текстовую поверхность на экран
    screen.blit(text, ((screen.get_width() -
                        text.get_width())/2,
                        (screen.get_height() -
                        text.get_height()) / 2))

    # обновить экран
    pygame.display.flip()
```

В результате появится следующий экран:

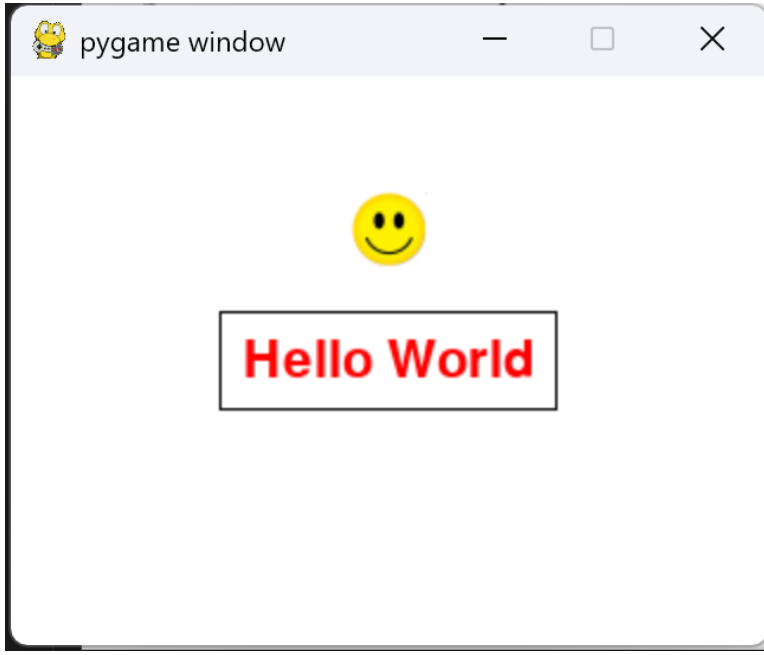


Добавление изображения

Добавление изображения в представление так же просто, как его загрузка и перенос на поверхность экрана. Вы, вероятно, будете использовать много изображений при создании своих игр, поэтому полезно знать, как рисовать изображения. В приведенном ниже коде мы загружаем изображение смайлика, а затем размещаем его по центру над текстом Hello World:

```
    # нарисуйте изображение смайлика над рамкой размером 32x32


    screen.blit(pygame.image.load("resources/smiley.png"),
                (screen.get_width()/2 - 16,
                (screen.get_height()
                - text.get_height()) / 2 - 60))
```

Это приводит к следующему отображению экрана:



Добавляем звук в нашу игру

В Pygame есть возможность рисовать фигуры, шрифты и изображения на экране. Он также поставляется с возможностью воспроизведения звуков и музыки. Следующий код показывает, как добавить звуковой сигнал на экран Hello World каждый раз, когда он мигает.

Сначала нам нужно инициализировать звуковой модуль pygame перед игровым циклом:

```
# Инициализируйте микшер для воспроизведения звука
 pygame.mixer.init()
pygame.mixer.music.load("resources/shortbeep.mp3")
```

Внутри нашего игрового цикла мы можем воспроизводить звук каждый раз в нашем игровом цикле после того, как все будет отрисовано. Мы уже добавили в игровой цикл задержку в 1 секунду, поэтому файл с коротким сигналом будет воспроизводиться каждую секунду.

```
count = 0
while True:
    count = count + 1
    screen.fill(WHITE)  # заполнить фон

    # проверить событие выхода
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            sys.exit()

    # создать текстовую поверхность
    text = font.render("Hello World", True, HelloWorldCol\
ors[count % 2])

    # 1 секундная задержка
```

```
16      pygame.time.delay(1000)
17
18      # объемный текст с черной прямоугольной рамкой
19      pygame.draw.rect(screen, BLACK,
20        ((screen.get_width() - text.get_width())/2 - 10,
21        (screen.get_height() -
22        text.get_height()) / 2 - 10,
23        text.get_width() + 20,
24        text.get_height() + 20), 1)
25
26      #нарисуйте изображение смайлика над рамкой размером
27      32x32
28
29
30      screen.blit(pygame.image.load("resources/smiley.png"),
31                  (screen.get_width()/2 - 16,
32                      (screen.get_height() -
33                      text.get_height()) / 2 - 60))
34
35      #  перенесите текстовую поверхность на экран
36      screen.blit(text, ((screen.get_width() -
37              text.get_width())/2,
38              (screen.get_height() - text.get_height()) / 2\
39  ))
40
41  # подавать звуковой сигнал
42      pygame.mixer.music.play()
43
44      # обновить экран
        pygame.display.flip()
```

Реакция на клавиатуру

Вы могли заметить, что мы добавили обработку событий в начале нашего игрового цикла, как показано в коде ниже. В настоящее время обработка событий только проверяет, вышли ли мы из игры или нет.

```
1
2     # проверить событие выхода
3       for event in pygame.event.get():
4           if event.type == pygame.QUIT:
                sys.exit()
```

Обработка событий также может позволить нам считывать события клавиатуры и мыши и использовать их в нашей игре. Давайте сначала посмотрим, можем ли мы использовать левую кнопку мыши, чтобы указать, где разместить смайлик. Когда левая мышь нажата, это вызовет событие pygame.MOUSEBUTTONDOWN внутри игрового цикла. Цикл событий будет получать положение мыши от мыши, и мы можем использовать это, чтобы поместить смайлик в положение мыши на экране.

```
1
2       for event in pygame.event.get():
3           if event.type == pygame.MOUSEBUTTONDOWN:
4               mouse_pos = pygame.mouse.get_pos()
5   ...
6
7   #  нарисовать изображение смайлика над рамкой размером 32x32
8
9       if mouse_pos != None:
10          screen.blit(pygame.image.load(
11              "resources/smiley.png"),
12              (mouse_pos[0] - 16, mouse_pos[1] - 16))
13      else:
14          screen.blit(pygame.image.load(
15              "resources/smiley.png"),
16              (screen.get_width()/2 - 16,
17              (screen.get_height() -
                text.get_height()) / 2 - 60))
```

Так что теперь, когда вы щелкнете где-нибудь в окне игры, вы увидите, что улыбка перемещается туда, где вы щелкнули! Вы можете заметить, что есть задержка между тем, когда вы нажимаете и когда смайлик фактически рисуется.

Это потому, что мы добавили 1-секундную задержку в коде для звука. На самом деле есть лучший способ справиться с миганием Hello World и звуковым сигналом каждую секунду, чтобы он не мешал извлечению событий. Способ сделать это - добавить часы вместо задержки и выполнять звуковой сигнал и мигание только тогда, когда часы достигают 1-секундной отметки на часах. Лучший способ проиллюстрировать использование часов — увидеть код:

```
    time = pygame.time

count = 0
oneSecondMarkReached = False
lastTime = 0

while True:
    # увеличивать счетчик каждую секунду
    if oneSecondMarkReached:
        count = count + 1

    screen.fill(WHITE)  # заполнить фон

    # проверить событие выхода
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            sys.exit()
        elif event.type == pygame.MOUSEBUTTONDOWN:
            mouse_pos = pygame.mouse.get_pos()

    # создать текстовую поверхность
    text = font.render("Hello World", True,
               HelloWorldColors[count % 2])

    # объемный текст с черной прямоугольной рамкой
    pygame.draw.rect(screen, BLACK,
```

```
27          ((screen.get_width() - text.get_width())/2 - 10,
28          (screen.get_height() -
29          text.get_height()) / 2 - 10,
30          text.get_width() + 20,
31          text.get_height() + 20), 1)
32
33  #нарисовать изображение смайлика над рамкой размером
32x32
34
35      if mouse_pos != None:
36          screen.blit(pygame.image.load(
37              "resources/smiley.png"),
38              (mouse_pos[0] - 16, mouse_pos[1] - 16))
39      else:
40          screen.blit(pygame.image.load(
41              "resources/smiley.png"),
42              (screen.get_width()/2 - 16,
43              (screen.get_height() -
44                text.get_height()) / 2 - 10 - 50))
45
46      # перенесите текстовую поверхность на экран
47      screen.blit(text,
48         ((screen.get_width() -
49             text.get_width())/2,
50             (screen.get_height()
51                 - text.get_height()) / 2))
52
53      if oneSecondMarkReached:
54          pygame.mixer.music.play()
55
56      #  обновить экран
57      pygame.display.flip()
58
59      # сбросить флаг oneSecondMarkReached
60      oneSecondMarkReached = False
61
```

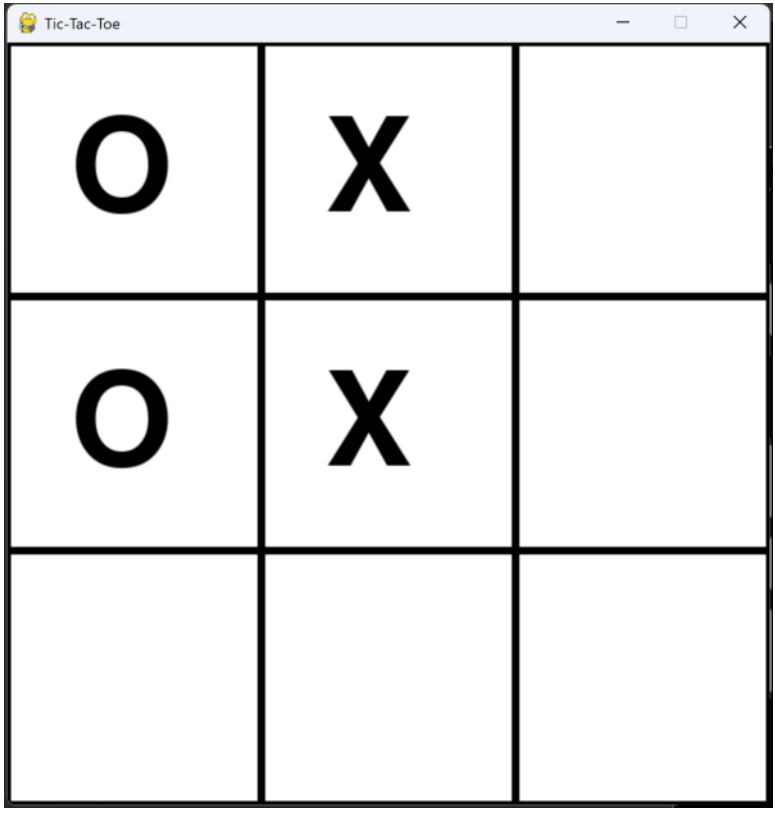
```
62      # информировать программу каждый раз,когда достигается
63      #  1-секундная отметка
64
65      currentTime = time.get_ticks()
66      if currentTime - lastTime > 1000:
67          lastTime = currentTime
68          oneSecondMarkReached = True
```

Нам нужно было немного изменить код, чтобы мигание происходило каждую секунду и звуковой сигнал — каждую секунду. Мы используем флаг oneSecond-MarkReached и устанавливаем его каждые 1000 тиков (1 секунда по времени). Затем он сбрасывается после того, как подаст звуковой сигнал и изменит цвет на текст «Hello World».

Выводы

Мы рассмотрели множество концепций, чтобы начать использовать множество связанных с игрой элементов, предоставляемых библиотекой pygame. Мы научились заливать фон на экран, рисовать текст, загружать и рисовать изображение, воспроизводить музыку и звук. В следующей главе мы погрузимся прямо в создание нашей первой игры, крестики-нолики.

Крестики-нолики в PyGame



Вступление

Добро пожаловать в главу о написании игры Tic Tac Toe с помощью PyGame. В этой главе мы рассмотрим основы библиотеки PyGame и способы ее использования для написания простой игры Tic Tac Toe для двух игроков. Мы расскажем, как нарисовать игровое поле, как определить ввод пользователя и как реализовать базовый ИИ для игры. К концу этой главы у вас должна быть работающая игра «Крестики-нолики», в которую вы сможете играть против компьютера. Итак, приступим!

Основной цикл

Следующий код представляет собой основной игровой цикл для игры в крестики-нолики, реализованной с помощью библиотеки PyGame. Он запускает цикл обработки событий для проверки пользовательского ввода, а затем рисует игровое поле. Если игра не окончена, он проверяет, поставил ли игрок X, затем ждет полсекунды, чтобы имитировать мышление ИИ, прежде чем поставить O. После того, как он поставит O, он проверяет, выиграл ли кто-нибудь игру, и если никто не выиграл, он проверяет, является ли это ничьей. Наконец, он обновляет дисплей.

```
1
2  ######################################################
3  # Основной игровой цикл
4  ######################################################
5  while True:
6      if game_over:
7          pygame.display.flip()
8          pygame.time.delay(1000)
9          draw_game_over_screen()
10         # Запустите обработку события, чтобы проверить выход
11         check_for_quit_event()
12     else:
13         game_window.fill(WHITE) # проверка белого фона
14         # для выхода и нажатия мыши
15         run_event_processing()
16         # Проверить на победу или ничью
17         game_over = check_for_win_or_draw()
18         draw_the_board()  # Нарисуйте игровое поле
19         pygame.display.flip()  # обновить экран
20
           # Проверьте, выиграл ли кто-нибудь после того,
           #  как X был размещен
```

```
        if game_over:
            continue

        # ИИ идет сюда, чтобы поставить O
        if X_placed:
            #  Подождите 1/2 секунды, чтобы это выглядело
            #  так, будто ИИ думает
            pygame.time.delay(500)
            O_placed = run_algorithm_to_place_O()
            game_over = check_if_anyone_won()
            # Снова нарисуйте доску, чтобы показать букву O,
            #  которую мы только что поставили.
            draw_the_board()
            X_placed = False

    # обновить экран
    pygame.display.flip()

    #  ограничить цикл до 60 кадров в секунду
    clock.tick(60)
```

Обработка событий

В основе содержимого игрового цикла лежит несколько функций, которые используют pygame для выполнения тяжелой работы. Давайте сначала посмотрим на функцию DoEventProcessing. Этот код представляет собой функцию в PyGame, которая запускает цикл обработки событий для проверки пользовательского ввода и щелчков мышью. Когда пользователь нажимает на доску, он обрабатывает событие нажатия мыши для X и устанавливает флаг X_placed в значение True, а также проверяет, решил ли пользователь выйти из игры. Событие выхода запускается, когда пользователь закрывает окно.

```
def run_event_processing():
    global X_placed
    global game_over

    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()  # выйти из игры
            quit()
        if event.type == pygame.MOUSEBUTTONDOWN:
            # Разместить X на доске
            handle_mouse_down_for_x()
            X_placed = True
```

Теперь давайте посмотрим на вызываемую функцию handle_mouse_down_for_x. Этот код используется для обработки события нажатия мыши для размещения X на доске для игры в крестики-нолики. Он использует функцию mouse.get_pos() библиотеки PyGame для получения положения мыши, затем делит строку и столбец на ширину и высоту сетки, чтобы получить строку и столбец клика. Наконец, он устанавливает соответствующую позицию в массиве доски в «X».

```
def handle_mouse_down_for_x():
    (row, col) = pygame.mouse.get_pos()
    row = int(row / grid_width)
    col = int(col / grid_height)
    board[row][col] = "X"
```

Создание доски

Функция draw_the_board используется для рисования доски Tic Tac Toe (крестики и нолики) в ее текущем состоянии. Он перебирает все строки и столбцы доски и вызывает функцию draw_game_board_square() для рисования каждого квадрата. Затем он проверяет, содержит ли доска в этой строке и столбце «X» или «O», и вызывает функцию draw_tic_tac_toe_letter() для рисования соответствующей буквы.

```
def draw_the_board():
    for row in range(grid_size):
        for col in range(grid_size):
            draw_game_board_square(row, col)
            # Сделать букву X
            if (board[row][col] == "X"):
                draw_tic_tac_toe_letter(row, col, 'X')
            # Сделать букву O
            if (board[row][col] == "O"):
                draw_tic_tac_toe_letter(row, col, 'O')
```

Создание игрового поля

Этот код используется для рисования квадрата игрового поля в указанной строке и столбце. Он использует функцию Rect() библиотеки PyGame для создания прямоугольного объекта с заданными строкой, столбцом, шириной и высотой. Затем он использует функцию draw.rect() для рисования прямоугольника в окне игры черным цветом и шириной линии 3.

```
def draw_game_board_square(row, col):
    rect = pygame.Rect(col * grid_width, row *
                       grid_height,
                       grid_width,
                       grid_height)
    pygame.draw.rect(game_window, BLACK, rect, 3)
```

Рисуем крестики-нолики

Этот код используется для рисования буквы «X» или «O» в указанной строке и столбце. Он использует функцию font.render() из библиотеки PyGame для рендеринга буквы как объекта Surface и задает черный цвет. Затем он использует метод game_window.blit() для рисования буквы в указанной строке и столбце, причем строка и столбец умножаются на ширину и высоту сетки, плюс четверть ширины и высоты сетки для ее центрирования.

```
def draw_tic_tac_toe_letter(row, col, letter):
    letter_piece = font.render(letter, True, BLACK)
    game_window.blit(
        letter_piece, (row * grid_width + grid_width/4,
                       col * grid_height + grid_height/4))
```

«AI» для размещения O

Для упрощения алгоритм размещения буквы O заключается в том, чтобы просто найти следующую доступную клетку. Мы улучшим это позже в этой главе, но эта стратегия должна, по крайней мере, позволить вам играть в игру против компьютерного оппонента.
AI - искусственный интелект (ИИ)

```
#######################################################
# Очень простой алгоритм размещения O на доске.
# Пройдитесь по всей доске и найдите первую доступную
#  клетку. Поместите O там.
#######################################################
def run_algorithm_to_place_O():
    for rowo in range(grid_size):
        for colo in range(grid_size):
            if (board[rowo][colo] == 0):
                board[rowo][colo] = "O"
                return True

    return False
```

Проверить на победу

Следующий код проверяет выигрыш на доске. Он смотрит, есть ли на доске три одинаковых символа подряд (по горизонтали, вертикали и диагонали).

```
def check_if_anyone_won():
    global winner
    #Проверить, выиграл ли кто-то по горизонтали
    for row in range(3):
        if board[row][0] == board[row][1]
                == board[row][2] != 0:
            winner = board[row][0]
            return True
    #Проверить, выиграл ли кто-то по вертикали
    for col in range(3):
        if board[0][col] == board[1][col]
                == board[2][col] != 0:
            winner = board[0][col]
            return True
    #Проверить, выиграл ли кто-то по диагонали
    if board[0][0] == board[1][1]
            == board[2][2] != 0:
        winner = board[0][0]
        return True
    if board[0][2] == board[1][1]
           == board[2][0] != 0:
        winner = board[0][2]
        return True

    #никто не выиграл, вернуть false
    return False
```

Проверка на ничью

Нам также нужно проверить, не осталось ли мест для размещения X или O, и ни один из игроков не выиграл игру. Для этого мы создадим новую функцию, которая проверяет, заполнена ли доска. Если никто не выиграл и доска заполнена, то ничья.

```
def check_if_board_is_full():
    for row in range(3):
        for col in range(3):
            if board[row][col] == 0:
                return False
    return True


#######################################################
#Проверьте, есть ли ничья, проверив, заполнена ли доска
#и никто не выиграл
#######################################################

def check_if_draw():
    return not (check_if_anyone_won()) and
       check_if_board_is_full()
```

Обработка состояния Game Over

Как только мы определили, есть ли выигрыш, проигрыш или ничья, мы устанавливаем для флага game_over значение True. Когда мы обнаруживаем, что игра окончена, мы должны отобразить экран завершения игры вместо доски для игры в крестики-нолики. В нашем основном цикле мы проверяем флаг game_over, и если он истинный, мы рисуем экран игры вместо доски для игры в крестики-нолики:

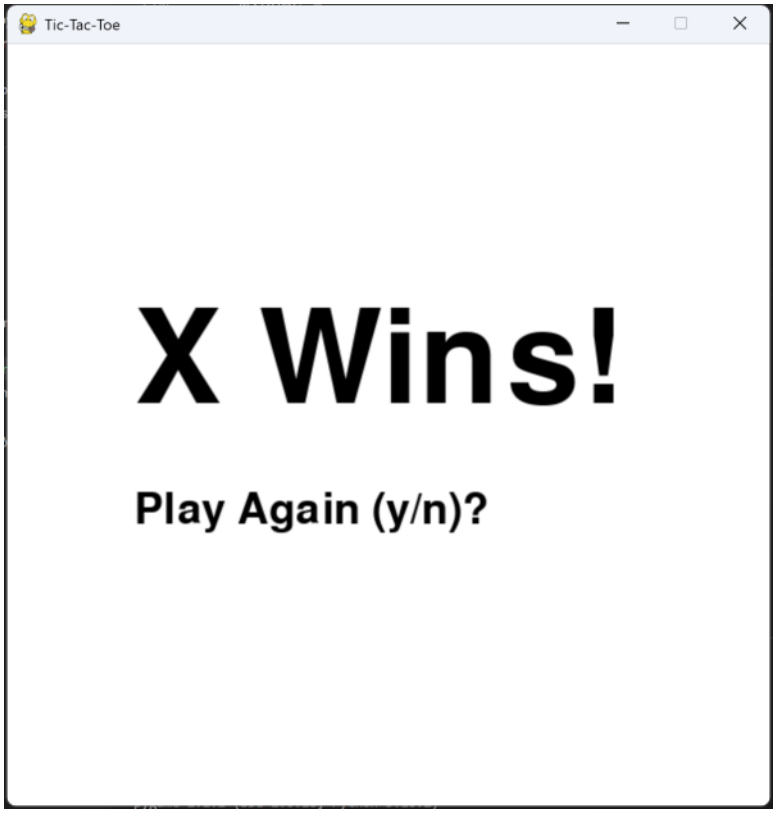
```
if game_over:
        #нарисовать игру на экране
        pygame.display.flip()
        pygame.time.delay(1000)
        draw_game_over_screen()
        check_for_quit_event()  # Запустите обработку события,
                                # чтобы проверить выход
    else:
        #нарисовать крестики / нолики на доске
```

Следующий код Python рисует игру на экране вместо доски для игры в крестики-нолики, как только мы определили, что игра завершена. Он проверяет строку победителя и отображает соответствующее сообщение о том, что произошло в игре, на основе этой строки. Экран Game Over также дает игроку возможность выбора: играть в новую игру или нет:

```
#######################################################
# Нарисуйте игру на экране, показывая, кто выиграл
#######################################################
def draw_game_over_screen():
    game_window.fill(WHITE)
    if winner == "X":
        text = font.render('X Wins!', True, BLACK)
    elif winner == "O":
        text = font.render('O Wins!', True, BLACK)
    else:
        text = font.render('Draw!', True, BLACK)

    playAgainText = smallfont.render(
          'Play Again (y/n)?', True, BLACK)

    game_window.blit(text,
        (window_width/2 - 200, window_height/2 - 100))

```

```
19      game_window.blit(playAgainText,
20          (window_width/2 - 200, window_height/2 + 50))
```

Получившееся изображение экрана игра закончена показано ниже:



Играю снова

Чтобы позволить игроку играть в новую игру, нам нужно очистить состояние текущей игры, когда пользователь нажимает клавишу y.

Мы сбрасываем глобальное состояние игры с помощью новой функции с именем initialize_game_values. Этот метод срабатывает, если пользователь нажимает клавишу «y» в состоянии окончания игры:

```
def check_for_quit_event():
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            quit()
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_y:
                initialize_game_values()
                game_window.fill(WHITE)
                return True
            elif event.key == pygame.K_n:
                pygame.quit()
                quit()
```

Как только в цикле событий обнаруживается y, мы инициализируем игровое поле и начинаем заново. Мы не останавливали игровой цикл, поэтому игровой цикл автоматически рисует доску на основе состояния переменной сброса после вызова initialize_game_values.

```
def initialize_game_values():
    global board
    global game_over
    global X_placed
    global O_placed
    global winner
    global clock

    game_over = False
    X_placed = False
    O_placed = False
    winner = ''

    board = [
```

```
15          [0, 0, 0],
16          [0, 0, 0],
17          [0, 0, 0],
18      ]
19
20      clock = pygame.time.Clock()
```

Лучший ИИ

Помните нашу дискуссию о внедрении более продвинутого ИИ (искусственного интеллекта) для игрока «O» в игре? Этот улучшенный алгоритм разработан, чтобы никогда не проигрывать! Вот шаги для этого уточненного алгоритма:

1. Подсчитайте количество сделанных ходов.
2. Если это второй ход (был сделан только один ход): a. Поместите «O» в центр доски, если она пуста. б. Если центр занят, поставьте «O» в первый доступный угол.
3. Для всех пустых позиций на доске: a. Проверьте, приведет ли размещение «O» в текущей позиции к победе для «O» игрока. Если это так, поставьте «O» и верните True. б. Проверьте, не помешает ли размещение «O» в текущей позиции выиграть игроку «X». Если это так, поставьте «O» и верните True.
4. Если «O» начинается в углу, поместите «O» в первый доступный угол и верните True.
5. Поместите «O» в первую доступную неугловую боковую позицию и верните True.ы
6. Если ни одно из вышеперечисленных условий не применимо, поместите «O» в первую доступную позицию и верните True.
7. Если свободных позиций нет, вернуть False.

Алгоритм использует ряд правил для определения оптимальной позиции для размещения «O» на доске «Крестики-нолики». Он учитывает текущее состояние доски и принимает решения на основе победы или блокировки соперника от победы, а также расставляет приоритеты угловых и неугловых сторон в зависимости от ситуации.

Вот код Python. Обратите внимание, что мы разбили его на 3 функции run_better_algorithm_to_place_O, is_winning_move и get_empty_positons:

```
# check if placing a piece in the row
# and column results in a winning move
def is_winning_move(player, row, col):
    n = len(board)
    # Проверить ряд
    if all(board[row][j] == player
        for j in range(n)):
        return True
    # Провер. столбец
    if all(board[i][col] == player
        for i in range(n)):
        return True
    #  Проверить главную диагональ
    if row == col and all(board[i][i]
       == player for i in range(n)):
        return True
    #Проверить второстепенную диагон.
    if row + col == n - 1 and
        all(board[i][n - i - 1]
          == player for i in range(n)):
        return True
    return False

#вернуть пустые позиции на доске в списке
def get_empty_positions():
    empty_positions = []
    for i, row in enumerate(board):
        for j, cell in enumerate(row):
            if cell == 0:
```

```
                empty_positions.append((i, j))
    return empty_positions


def run_better_algorithm_to_place_O():
    grid_size = len(board)
    empty_positions = get_empty_positions()
    num_moves = sum(1 for row in board for
            cell in row if cell != 0)

    #Второй ход: поместите букву «O» в центр или угол.
    if num_moves == 1:
        center = grid_size // 2
        if board[center][center] == 0:
            board[center][center] = "O"
            return True
        else:
            for row, col in
                [(0, 0), (0, grid_size - 1),
                (grid_size - 1, 0),
                (grid_size - 1, grid_size - 1)]:
                if board[row][col] == 0:
                    board[row][col] = "O"
                    return True

    # Попробуйте выиграть или заблокируйте X для победы
    for row, col in empty_positions:
        #Проверьте, выиграет ли игра от размещения "O"
        board[row][col] = "O"
        if is_winning_move("O", row, col):
            return True
        board[row][col] = 0

    # Проверьте, не заблокирует ли X  размещению «O»
    for row, col in empty_positions:
```

```
65          board[row][col] = "X"
66          if is_winning_move("X", row, col):
67              board[row][col] = "O"
68              return True
69          board[row][col] = 0
70
71      #Поместите «O» в угол в начале игры
72      if board[0][0] == "O"
73          or board[0][grid_size - 1] == "O"
74          or board[grid_size - 1][0] == "O"
75          or board[grid_size - 1][grid_size - 1]
76               == "O":
77          for row, col in
78              [(0, 0), (0, grid_size - 1),
79              (grid_size - 1, 0),
80              (grid_size - 1, grid_size - 1)]:
81              if board[row][col] == 0:
82                  board[row][col] = "O"
83                  return True
84
85      #Поместите «O» на неугловой стороне
86      for row, col in empty_positions:
87          if row not in [0, grid_size - 1]
88             and col not in [0, grid_size - 1]:
89              board[row][col] = "O"
90              return True
91
92      # Поместите «O» в любое доступное место
93      for row, col in empty_positions:
94          board[row][col] = "O"
95          return True
96
97      return False
```

Заключение

Изучив основы создания игры с помощью pygame, мы теперь можем вывести свои навыки на новый уровень, оптимизировав наш код с помощью объектно-ориентированного программирования. Организуя наши игровые объекты в классы, мы можем упростить и оптимизировать наш код, упростив управление и поддержку. В следующей главе мы рассмотрим эту технику более подробно и покажем вам, как реализовать ее в ваших собственных играх.

Использование классов в Pygame

Введение

Крестики и нолики и другие игры, которые мы пишем в Pygame, способствуют тому, чтобы их можно было разбить на более мелкие объекты, с которыми может работать программа. Классы в Python могут помочь разработчикам создавать повторно используемый и поддерживаемый код. Благодаря организации кода в классы его легче читать и понимать, а также можно более эффективно выполнять отладку и тестирование. Кроме того, это позволяет разработчикам легко изменять и добавлять в игру новые элементы без необходимости переписывать код, что особенно полезно при разработке сложных игр. Классы также помогают поддерживать порядок в коде и упрощают реализацию новых функций. Наконец, использование классов помогает разработчикам создавать более эффективный код, поскольку они могут легко использовать один и тот же код для похожих частей игры.

Давайте погрузимся в создание классов для нашей игры. Класс Python, представляющий букву (X или O), которую можно нарисовать на экране в pygame, можно определить следующим образом:

Вот пример класса Pygame, который наследуется от класса Sprite и рисует на экране «X» или «O»:

```
import pygame


class LetterSprite(pygame.sprite.Sprite):

    def __init__(self, letter, row, column,
                 grid_width, grid_height):
        #инициализировать базовый класс спрайта
        super().__init__()
        font = pygame.font.Font(None, 150)
        # визуализировать шрифт на поверхности изображения
        self.image = font.render(letter, True, (0, 0, 0))
        # определить границы изображения на доске
        self.rect = self.image.get_rect().move(
            row * grid_width + grid_width / 3,
            column * grid_height + grid_height / 3)

    def update(self):
        pass

    def draw(self, surface):
        letter_piece = self.image
        surface.blit(letter_piece, self.rect)
```

Класс Letter принимает 5 аргументов: сама буква, которая хранится как переменная экземпляра, строка и столбец, в которых буква размещается на доске, и размеры сетки. Конструктор инициализации выполняет большую часть тяжелой работы. Он создает изображение из шрифта по умолчанию и вычисляет положение прямоугольника из размеров строки, столбца и сетки. Так как движения от текущего положения нет, письмо не нуждается в методе обновления, поэтому с обновлением мы ничего не делаем. Метод draw() принимает только один аргумент: экран, то есть поверхность игры. Затем он использует pygame.Surface как средство для вывода письма на экран игры.

Мы можем создать наш LetterSprite, как только игрок нажмет событие mousedown, а затем мы можем использовать класс Group в pygame, чтобы собрать все x, которые мы добавляем на доску.

```
def handle_mouse_down_for_x():
    (row, col) = pygame.mouse.get_pos()
    row = int(row / grid_width)
    col = int(col / grid_height)
    board[row][col] = "X"
    letterX = LetterSprite('X', row, col,
          grid_width, grid_height)
    group.add(letterX)
```

Причина, по которой мы добавляем X в группу, заключается в том, что когда мы хотим отрисовать все игровые фигуры, мы просто вызываем group.draw(surface), и она нарисует для нас все игровые фигуры сразу. Как мы скоро увидим, то же самое можно сделать и с буквой «O»!

Теперь мы можем удалить 90% кода, рисующего X и O, и он сведется к одной строке кода: group.draw(game_window)

```
def draw_the_board():

    group.draw(game_window)

    for row in range(grid_size):
        for col in range(grid_size):
            draw_game_board_square(row, col)
```

Обратите внимание, что мы все еще зацикливаемся на создании игровых квадратов. Мы также можем создавать спрайты для игровых квадратов:

```
import pygame

# Создать спрайт
class GameBoardSquareSprite(pygame.sprite.Sprite):
    def __init__(self, color, row, column, width, height):
        super().__init__()
        self.width = width
        self.height = height
        # Create a surface for the sprite
        self.image = pygame.Surface([width, height])
        #сделать фоновую игровую плитку белой
        self.image.fill((255, 255, 255))
        self.rect = self.image.get_rect().move(row*width,
            column*height)
        #Нарисуйте прямоугольник на поверхности спрайта
        pygame.draw.rect(self.image, color, pygame.Rect(
            0, 0, width, height),  2)

    # Рисуем спрайт на экране

    def draw(self, surface):
        surface.blit(self.image, 0, 0)
```

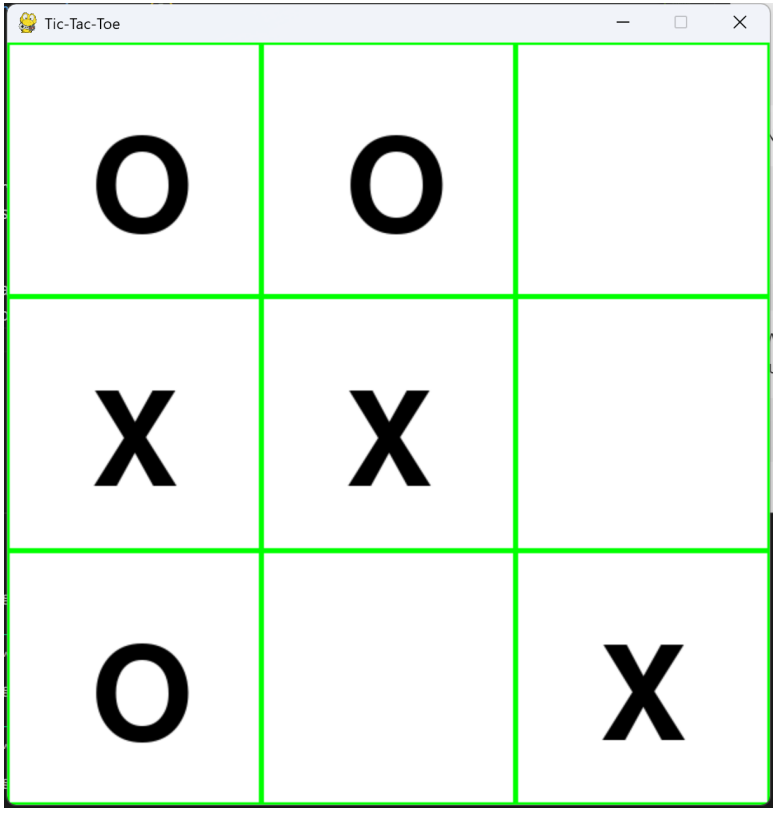
Теперь в нашей initialize_game_board мы добавим игровые плитки в группу:

```
def initialize_game_board():
    for row in range(3):
        for column in range(3):
            game_board_square = GameBoardSquareSprite(
                (0, 255, 0),
                 grid_width, grid_height)
            group.add(game_board_square)
```

Когда вызывается group.draw, он отрисовывает тайлы, а также сыгранные крестики и нолики. Наша функция draw_the_board теперь выглядит так:

```
def draw_the_board():
    group.draw(game_window)
```

Так как мы решили сделать квадраты нашего игрового поля зелеными, итоговое поле выглядит так, как показано на рисунке ниже:



Рефакторинг игровой логики

Чтобы еще больше разбить код на модули, мы можем вынести всю игровую логику из основного модуля Python в класс под названием GameBoard, который будет проверять выигрыши, проигрыши и ничьи, а также даст нам возможность заполнить доску нашими предположениями.

Он также может управлять алгоритмической логикой размещения букв «O».

```
class GameBoard:
    def __init__(self, grid_size):
        self.grid_size = grid_size
        self.winner = ''
        self.initialize_board()

    ##########################################################
    #Инициализировать доску нулями
    ##########################################################

    def initialize_board(self):
        self.board = [
            [0, 0, 0],
            [0, 0, 0],
            [0, 0, 0],
        ]

##################################################################
Проверьте, выиграл ли кто-то в любой строке, столбце или диагонали
##################################################################

```

```
22      def check_if_anybody_won(self):
23          #  Проверить, выиграл ли кто-то по горизонтали
24
25          for row in range(3):
26              if self.board[row][0] == self.board[row][1]
27                == self.board[row][2] != 0:
28                  self.winner = self.board[row][0]
29                  return True
30
31          #Проверить, выиграл ли кто-то по вертикали
32          for col in range(3):
33              if self.board[0][col] == self.board[1][col]
34                == self.board[2][col] != 0:
35                  self.winner = self.board[0][col]
36                  return True
37
38          # Проверить, выиграл ли кто-то по диагонали
39          if self.board[0][0] == self.board[1][1]
40          == self.board[2][2] != 0:
41              self.winner = self.board[0][0]
42              return True
43          if self.board[0][2] == self.board[1][1]
44          == self.board[2][0] != 0:
45              self.winner = self.board[0][2]
46              return True
47
48          return False
49
50      ########################################################
51      # Проверьте, заполнена ли доска
52      ########################################################
53
54      def check_if_board_is_full(self):
55          for row in range(3):
56              for col in range(3):
```

```
57                  if self.board[row][col] == 0:
58                      return False
59          return True
60
61      ########################################################
62      #  Проверьте, есть ли ничья, проверив, заполнена
63      #  ли доска и никто не выиграл.
64      ########################################################
65
66      def check_if_draw(self):
67          return not (self.check_if_anybody_won()) and
68             self.check_if_board_is_full()
69
70      ########################################################
71      # Поместите X
72      ########################################################
73      def place_X(self, row, col):
74          self.board[row][col] = "X"
75
76      ################################################
77      #  Используется run_better_algorithm_to_place_O, чтобы
78      #  определить, приводит ли размещение фигуры в строке
79      # или столбце на доске к выигрышному ходу. Это
80      #  используется для определения блокировки,а также
81      # для победы противника "O".
82      ################################################
83      def is_winning_move(self, player, row, col):
84          n = len(self.board)
85          # Check row
86          if all(self.board[row][j] == player
87             for j in range(n)):
88              return True
89          # Check column
90          if all(self.board[i][col] == player
91             for i in range(n)):
```

```
92              return True
93          # Check main diagonal
94          if row == col and all(self.board[i][i] ==
95              player for i in range(n)):
96                  return True
97          # Check secondary diagonal
98          if row + col == n - 1 and
99              all(self.board[i][n - i - 1]
100             == player for i in range(n)):
101             return True
102         return False
103
104     ################################################
105     #  Используется методом run_better_algorithm_to_place_O
106     #  для сбора всех доступных позиций на доске.
107     ################################################
108     def get_empty_positions(self):
109         empty_positions = []
110         for i, row in enumerate(self.board):
111             for j, cell in enumerate(row):
112                 if cell == 0:
113                     empty_positions.append((i, j))
114         return empty_positions
115
116 ######################################################
117 # Использует алгоритм, чтобы решить, где разместить O.
118 # Этот алгоритм никогда не проигрывает.
119 ######################################################
120     def run_better_algorithm_to_place_O(self):
121         grid_size = len(self.board)
122         empty_positions = self.get_empty_positions()
123         num_moves = sum(1 for row in self.board for
124                         cell in row if cell != 0)
125
126         # Второй ход: поместите  «O» в центр или угол.
```

```
        if num_moves == 1:
            center = grid_size // 2
            if self.board[center][center] == 0:
                self.board[center][center] = "O"
                return (True, center, center)
            else:
                for row, col in [(0, 0),
                    (0, grid_size - 1),
                    (grid_size - 1, 0),
                    (grid_size - 1, grid_size - 1)]:
                    if self.board[row][col] == 0:
                        self.board[row][col] = "O"
                        return (True, row, col)

        # Попробуйте выиграть или заблокируйте X от победы
        for row, col in empty_positions:
            # Проверьте, выиграет ли игра от размещения "O"
            self.board[row][col] = "O"
            if self.is_winning_move("O", row, col):
                return (True, row, col)
            self.board[row][col] = 0

        # Проверьте, не помешает ли размещение «O» X выиграть
        for row, col in empty_positions:
            self.board[row][col] = "X"
            if self.is_winning_move("X", row, col):
                self.board[row][col] = "O"
                return (True, row, col)
            self.board[row][col] = 0

        # Поместите «O» в угол в начале игры
        if self.board[0][0] == "O"
            or self.board[0][grid_size - 1] == "O"
            or self.board[grid_size - 1][0] == "O"
            or self.board[grid_size - 1][grid_size - 1]
```

```
162                 == "O":
163             for row, col in [(0, 0), (0, grid_size - 1),
164                              (grid_size - 1, 0),
165                              (grid_size - 1, grid_size - 1)]:
166                 if self.board[row][col] == 0:
167                     self.board[row][col] = "O"
168                     (True, row, col)
169                     return (True, row, col)
170
171         # Поместите «O» на неугловой стороне
172         for row, col in empty_positions:
173             if row not in [0, grid_size - 1]
174             and col not in [0, grid_size - 1]:
175                 self.board[row][col] = "O"
176                 return (True, row, col)
177
178         #Поместите «O» в любое доступное место
179         for row, col in empty_positions:
180             self.board[row][col] = "O"
181             return (True, row, col)
182
183         return (False, -1, -1)
```

Теперь мы можем вызывать все эти функции для проверки победителей и расстановки крестиков и ноликов на доске из основного файла игры, а основной файл игры стал намного чище.

В нашей Initialize_game_values мы строим доску следующим образом:

```
1     board = GameBoard(grid_size)
```

Затем везде, где мы используем доску, мы просто вызываем ее методы.

Ниже приведен вызов метода run_better_algorithm_to_place_O GameBoard, куда мы помещаем спрайт O из основной игровой программы. Алгоритмический метод на доске возвращает кортеж, указывающий, смогли ли мы найти место для фигуры на доске, и если да, то в какой строке и столбце она была размещена.

```
(O_placed, rowo, colo) =
    board.run_better_algorithm_to_place_O()
if O_placed:
                letterO = LetterSprite(
                    'O', colo, rowo,
                    grid_width,
                    grid_height)
                group.add(letterO)
```

Мы также можем получить доступ к любым внутренним атрибутам класса GameBoard, например к победителю игры:

```
 if board.winner == "X":
        text = font.render('X Wins!', True, BLACK)
    elif board.winner == "O":
        text = font.render('O Wins!', True, BLACK)
    else:
        text = font.render('Draw!', True, BLACK)
```

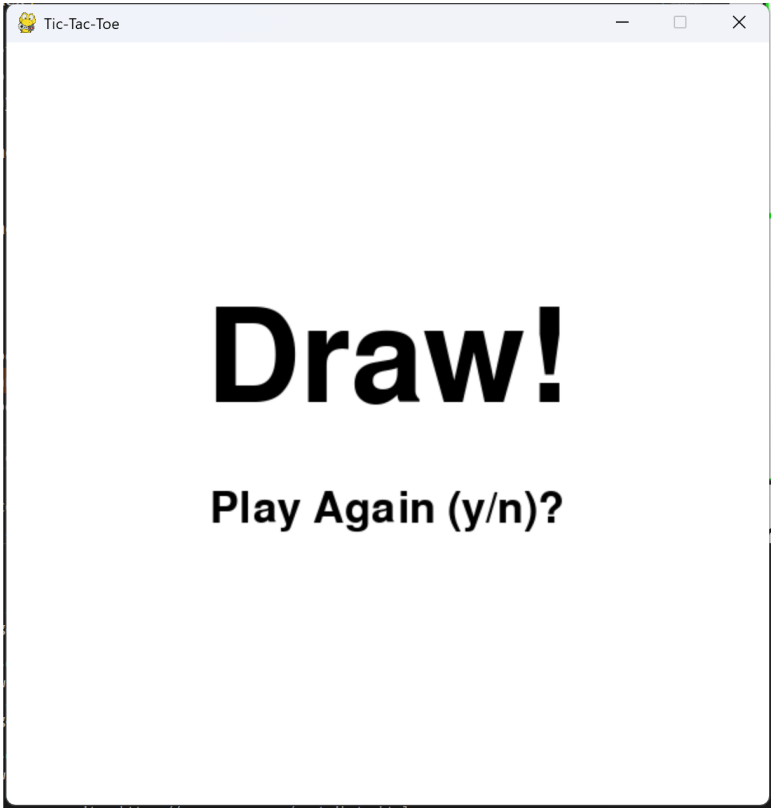
Глядя на этот код, на самом деле это возможность реорганизовать метод, который возвращает строку победителя для экрана Game Over. Итак, мы добавим в GameBoard новый метод get_winner_display_message:

```
 def get_winner_display_message(self):
        if self.winner == 'X':
            return 'X Wins!'
        elif self.winner == 'O':
            return 'O Wins!'
        else:
            return 'Draw!'
```

а затем вызовите его из функции draw_game_over_screen в нашей основной программе pygame.

```
def draw_game_over_screen():
    game_window.fill(WHITE)
    winnerMessage = board.get_winner_display_message()

    text = font.render(winnerMessage, True, BLACK)

    # получить ширину текста, чтобы мы могли
    # центрировать по горизонтали
    text_width = text.get_width()

    playAgainText = smallfont.render('Play Again (y/n)?',
      True, BLACK)                    'Играть снова'

   # получить подсказку о ширине размещения, чтобы
   # мы могли центрировать ее по горизонтали
    playAgainText_width = playAgainText.get_width()

    game_window.blit(
        text, (window_width/2 - text_width/2,
          window_height/2 - 100))

    game_window.blit(playAgainText,
        (window_width/2 - playAgainText_width/2,
        window_height/2 + 50))
```

Этот рефакторинг кода взял на себя ответственность за то, чтобы знать, как был определен победитель из основной программы, и поместил его в черный ящик, который мы могли вызывать из нашего объекта board.

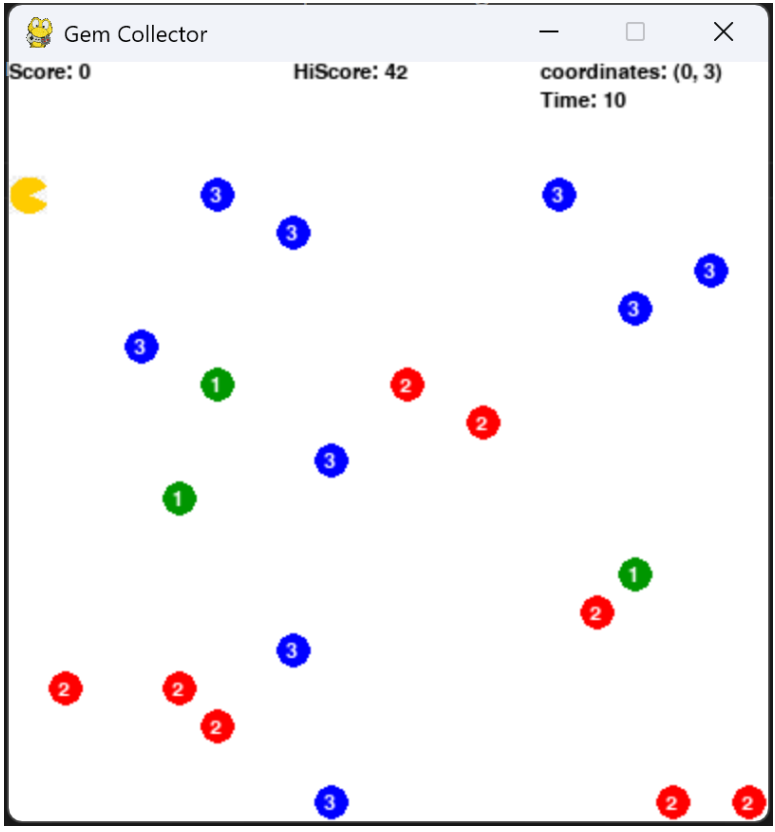


Заключение

В этой главе мы рассмотрели использование классов в pygame для улучшения организации и удобства сопровождения вашего кода. Мы достигли этого, используя класс GameBoard для обработки игровой логики и управления данными, возложив ответственность за отрисовку на классы спрайтов. Благодаря своим методам и свойствам эти классы были интегрированы в основной цикл.

Кроме того, мы перенесли большую часть низкоуровневого рисования в классы Sprite, упорядочив основную программу. В следующей главе мы представим игру, в которой игроки должны собирать драгоценные камни в течение определенного периода времени, используя клавиши со стрелками. Мы включим многие концепции, ранее обсуждавшиеся в предыдущих главах.

Глава 6 - Пожиратель камней



Введение

В предыдущей главе мы создали игру в крестики-нолики, чтобы играть против компьютера. Stone Eater — это игра, в которую вы играете на время! Цель игры состоит в том, чтобы съесть как можно больше драгоценных камней, до того, как время закончится.

Каждый камень стоит 1, 2 или 3 очка, поэтому вам нужно есть камни, которые имеют более высокую ценность. Stone Eater вводит в нашу игру несколько новых концепций. В этой главе мы узнаем, как обрабатывать события клавиатуры, анимировать персонажа игрока и воспроизводить звук, чтобы добавить еще одно измерение в нашу игру.

Проект игры

Классы

Мы воспользуемся классами для нашей игры. В игре мы будем использовать несколько классов спрайтов, каждый из которых представляет игровой объект. Будет спрайт, используемый для рисования пожирателя камней, а также спрайт, используемый для рисования всех драгоценных камней в игре. Также у нас будут спрайты для каждой статистики, которую мы используем в игре: счет, высокий счет, координаты и время. Мы также создадим спрайт общего сообщения для публикации текста, например «играть снова? » Кроме того, как мы это делали в крестиках-ноликах, мы создадим класс игровой доски, который будет контролировать всю игровую логику и состояние игры, когда игрок ест камень. Ниже приведен список классов, которые мы только что упомянули:

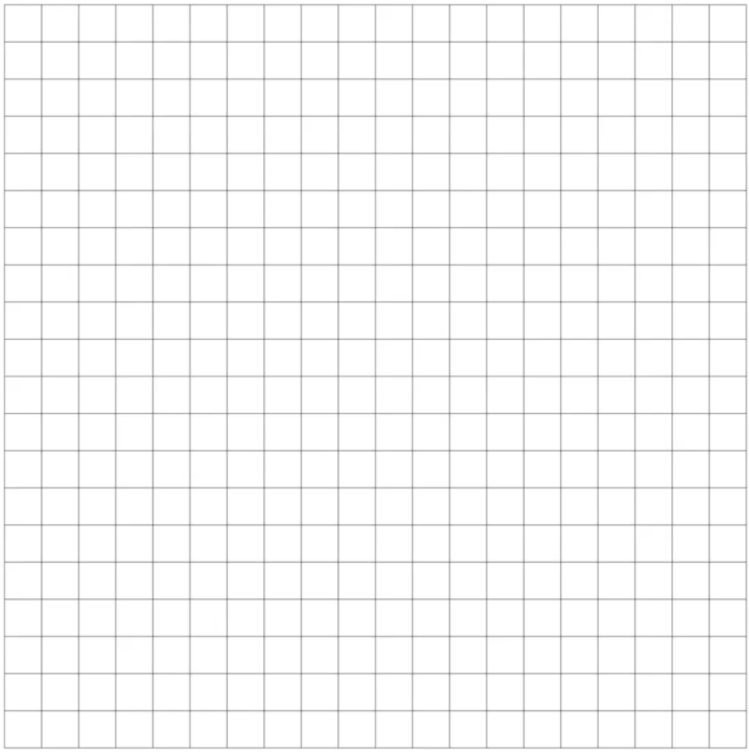
PlayerSprite StoneSprite GameBoard ScoreSprite HiScoreSprite TimeSprite CoordinateSprite MessageSprite

Макет игры

Как и в крестики-нолики, игра с пожирателем камней представляет собой сетку. Игровое поле в этой игре представляет собой сетку 20x20, каждая ячейка которой имеет ширину 20 пикселей. Когда игрок перемещается вверх, вниз, влево или вправо, пожиратель камней перемещается на следующую соседнюю ячейку в сетке. Драгоценные камни также случайным образом размещаются внутри сетки в разных ячейках.

Хотя игровое поле представляет собой сетку 20 x 20, мы используем только нижние 17 рядов, чтобы оставить место для спрайтов в верхней части игры.

Мы могли бы сделать это по-другому, поместив сетку ниже очков, но этот метод работает так же хорошо, пока мы ограничиваем игрока от перехода выше 3-й строки в сетке.



Инициализация игры

Прежде чем мы начнем игровой цикл, как и в любой другой игре, нам нужно инициализировать игровые элементы. Функция initialize_game_state устанавливает начальное состояние игры Stone Eater. Эта функция начинается с объявления нескольких глобальных переменных, которые будут использоваться на протяжении всей игры: gems_collected, gems_score, start_time, times_up и already_played.

Затем функция очищает группу gem_group, чтобы удалить все существующие камни на игровом поле. Она устанавливает start_time равным текущему времени и устанавливает times_up равным False. Начальная позиция игрока устанавливается в 3-й строке и первом столбце игрового поля. Время игры и счет обновляются до своих начальных значений time_limit и 0 соответственно. Для переменных gems_collected и gems_score установлено значение 0, а для уже_играемого установлено значение False. Наконец, функция вызывает метод initialize_board для сброса логики игрового поля в исходное состояние.

```
def initialize_game_state():
    global gems_collected, gems_score, start_time,
     times_up, already_played
    print("Initializing game state")
    gem_group.empty()
    start_time = pygame.time.get_ticks()
    times_up = False
    player.row = player_limit
    player.column = 0
    player.update()
    game_time.update_time(time_limit, 0)
    score.update_score(0)
    gems_collected = 0
    gems_score = 0
    already_played = False

    game_board.initialize_board()


initialize_game_state()
```

Цикл игры

Как и в крестиках-ноликах, игра состоит из приема событий и рисования спрайтов в зависимости от состояния доски и событий.

Ниже приведена грубая архитектура игрового цикла, который управляет всей игрой Stone Eater.

```
while running:
    # Получить событие клавиатуры для перемещения игрока
    # обновить позицию спрайта игрока, если была нажата клавиша
    #определить, столкнулся ли игрок с камнем
    # если игрок столкнулся,
    # удалить камень и воспроизвести звук
    #
    # обновить спрайты подсчета очков
    # нарисовать все драгоценные камни,
    # если время вышло
    # ничья воспроизвести снова (д/н) сообщение
    #   обновить высокий балл
    #    включи музыку в конце игры
    # еще
    # рисовать спрайты игрока и игрового времени
    # нарисовать все спрайты драгоценных камней
    # нарисовать все спрайты очков,
    # обратный цикл
```

Теперь давайте посмотрим на реальный игровой цикл. Ниже приведен полный код игрового цикла «Пожиратель камней», который, как вы заметите, похож на игровой цикл «крестики-нолики». Как мы описали в нашем псевдокоде, цикл обрабатывает события от пользователя, соответственно обновляет игровое поле и рисует все на игровом поле. Он использует группы спрайтов для выполнения как обновления, так и отрисовки спрайтов каждый раз в цикле. Игровой цикл также воспроизводит звуки, когда это необходимо: мы проигрываем звук каждый раз, когда съедаем камень, и мы проигрываем музыку, когда время истекает и игра завершена.

```
# Основной
игровой цикл
running = True
while running:
    running = process_events()  # обработать клавиатуру

    # Проверить, подобрал ли игрок драгоценный камень
    if game_board.check_for_gem(player) and
       (times_up == False):
        # драгоценный камень найден, обновите счет
        gems_collected += 1
        gems_score += game_board.get_cell_value(
          player.row, player.column)
        score.update_score(gems_score)
        # убрать драгоценный камень с доски и спрайт драгоценного камня
        game_board.remove_gem(player)
        which_sprite = detect_collisions(player,
           gem_group, piece_size)
        remove_sprite_from_group(which_sprite, gem_group)
        got_coin_sound.play()

    # Обновить координаты
    coordinates.update_coordinatees(player.row, player.co\
lumn)

    # Время обновления
    game_time.update_time(time_limit,
                pygame.time.get_ticks() - start_time)

    # Проверить, истекло ли время
    if (pygame.time.get_ticks() -
       start_time > time_limit * 1000)
       and (times_up == False):
        times_up = True

```

```
36      # очистить экран
37      window.fill(WHITE)
38
39      # Проверяем, установлен ли флаг таймаута
40      if times_up:
41          # установить текущий счет для сравнения
42          # с высоким баллом
43          # и один раз включи музыку в конце игры
44          if already_played == False:
45              hi_score.current_score = gems_score
46              victory_sound.play()
47              already_played = True
48
49          gems_collected_message.update_message(
50              f'You collected {str(gems_collected)} gems!')
51
52          gems_collected_message.update()
53          gems_collected_message.draw(window)
54          play_again_message.draw(window)
55      else:
56          # розыгрыш игрока и игрового времени
57          player.draw(window)
58          game_time.draw(window)
59
60      # нарисуй драгоценные камни
61      gem_group.draw(window)
62
63      # обновить статистику
64      # (счет, его счет, координаты, оставшееся время)
65      score_group.update()
66
67      # нарисуй статистику
68      score_group.draw(window)
69
70      # отображать всю графику
```

```
71      # мы только что нарисовали
72      pygame.display.flip()
```

Код в полной мере использует спрайты и группы спрайтов. Драгоценные камни и баллы находятся в группе, поэтому их можно рисовать сразу.

Обнаружение нажатия клавиш

Внутри нашего цикла событий мы проверяем, нажал ли пользователь какую-либо из клавиш со стрелками, чтобы мы могли перемещать пожирателя камней влево, вправо, вверх или вниз в зависимости от того, какую клавишу со стрелкой нажал игрок. Мы делаем это, перебирая все события в очереди событий pygame и проверяя, не нажаты ли какие-либо из них. Если да, то мы проверяем, какая клавиша клавиатуры была выбрана, и сравниваем ее с интересующими нас клавишами. Например, мы смотрим, была ли выбрана стрелка вверх (pygame.K_UP).

Как только мы определили, что он выбрал стрелку вверх, мы также проверяем, не пытается ли игрок уйти за пределы игрового поля, потому что мы не хотим, чтобы игрок выходил за пределы игрового поля. В случае клавиши со стрелкой вверх мы ограничиваем игрока третьей строкой сетки, чтобы игрок не начал переходить к игровой статистике. Как только мы определяем, что игрок находится внутри игрового поля, мы перемещаем игрока на одну ячейку в направлении нажатой клавиши. Для стрелки вверх мы вычитаем единицу из строки игрока, чтобы переместить его вверх по строке в сетке игрового поля.

```
for event in pygame.event.get():
        if event.type == pygame.QUIT or
        (event.type == pygame.KEYDOWN and
         event.key == pygame.K_n and times_up == True):
            running = False
        elif event.type == pygame.KEYDOWN:
            # Проверить, переместился ли игрок
            if event.key == pygame.K_UP
              and player.row > player_limit:
                player.row -= 1
                player.update()
            elif event.key == pygame.K_DOWN
              and player.row < GRID_LENGTH - 1:
                player.row += 1
                player.update()
            elif event.key == pygame.K_LEFT and
              player.column > 0:
                player.column -= 1
                player.update()
            elif event.key == pygame.K_RIGHT and
              player.column < GRID_LENGTH - 1:
                player.column += 1
                player.update()
            elif event.key == pygame.K_y and
               times_up == True:
                initialize_game_state()
```

Игровое поле

Как и в крестиках-ноликах, игровое поле в Stone Eater используется для размещения драгоценных камней, а также для отслеживания местоположения драгоценных камней и определения, были ли они съедены или нет. Ниже приведены методы класса Game Board и их назначение.

init (constructor) **initialize_board** -размещает начальные камни на доске

check-for-gem - проверяет, есть ли в строке и столбце драгоценный камень
remove_gem - удаляет драгоценный камень с доски
get_cell_value - получить значение ячейки в указанной строке и столбце

Когда мы инициализируем доску, мы размещаем драгоценные камни в случайных незанятых местах на доске.

```
1
2   def initialize_board(self):
3         # заполнить пустую сетку матрицей 20 x 20 из 0
4         self.grid = []
5         for i in range(self.grid_size):
6             self.grid.append([])
7             for j in range(self.grid_size):
8                 self.grid[i].append(0)
9         # Размещайте драгоценные камни случайным образом на self.grid
10        num_gems = 20
11        for i in range(num_gems):
12            gem_placed = False
13            while not gem_placed:
14                row = random.randint(self.player_limit,
15                   self.grid_size - 1)
16                column = random.randint(0,
17                  self.grid_size - 1)
18                if self.grid[row][column] == 0:
19                    self.grid[row][column]
20                       = random.randint(1, 3)
21                    gem_placed = True
22            #добавляем каменные спрайты в группу драгоценных камней
23                    # и размещаем их на доске
24                    self.gem_group.add(StoneSprite(
25                        self.colors[
26                         self.grid[row][column]-1],
27                         row, column, self.piece_size,
28                          self.grid[row][column]))
```

Below is the entire class that includes all the methods described above. The GameBoard class makes it easier to do all the game logic as it relates to the board and hides the internal grid mapping from the game loop so the game loop doesn't have to think about it.

```
class GameBoard:
    def __init__(self, size, piece_size,
      player_limit, gem_group):
        self.grid_size = size
        self.piece_size = piece_size
        self.player_limit = player_limit
        self.grid = []
        self.gem_group = gem_group
        # цвета драгоценных камней
        GREEN = (0, 150, 0)
        RED = (255, 0, 0)
        BLUE = (0, 0, 255)
        self.colors = [GREEN, RED, BLUE]

    def initialize_board(self):
        # заполняем пустую сетку матрицей 20 x 20 из нулей
        self.grid = []
        for i in range(self.grid_size):
            self.grid.append([])
            for j in range(self.grid_size):
                self.grid[i].append(0)

     # Размещайте драгоценные камни случайным образом на self.grid
        num_gems = 20
        for i in range(num_gems):
            gem_placed = False
            while not gem_placed:
                row = random.randint(self.player_limit,
                  self.grid_size - 1)
                column = random.randint(
                   0, self.grid_size - 1)
```

```
                if self.grid[row][column] == 0:
                    self.grid[row][column] =
                      random.randint(1, 3)
                    gem_placed = True
            # добавляем спрайты камней в группу драгоценных камней
                    # и расставляем их на доске
                    self.gem_group.add(StoneSprite(
                        self.colors[
                         self.grid[row][column]-1],
                         row, column,
                         self.piece_size,
                         self.grid[row][column]))

    def check_for_gem(self, player):
        if self.grid[player.row][player.column] > 0:
            return True
        else:
            return False

    def remove_gem(self, player):
        self.grid[player.row][player.column] = 0

    def get_cell_value(self, row, column):
        return self.grid[row][column]
```

Игровые спрайты

Каждый игровой спрайт рисует объект в игре. Все спрайты имеют в своей структуре следующий контракт:

```
class MySprite
    __init__
    def update(self)
    def draw(self, surface)
```

Нам не всегда нужно реализовывать обновление, потому что возможно, что игровой спрайт не двигается или изменяется каким-либо образом. Например, камни после их создания не меняются ни графически, ни в своем положении, поэтому нет необходимости их обновлять. С другой стороны, спрайт игрока движется с каждым нажатием клавиши, поэтому он должен постоянно обновляться в игре при обнаружении клавиши со стрелкой. Функция рисования используется для рисования спрайта, поэтому она используется всегда. Давайте посмотрим на спрайт для камня и спрайт для игрока:

Каменный спрайт ниже имеет большую часть кода в конструкторе. Это потому, что как только мы определяем его образ, он никогда не меняется. Даже отрисовка спрайта заранее предопределена в конструкторе. Все, что нужно сделать функции отрисовки, — это скопировать изображение, созданное в конструкции, и скопировать значение драгоценного камня (1,2 или 3). Функция обновления абсолютно ничего не делает, если она вызывается.

Давайте внимательнее посмотрим на конструктор (__init__), поскольку именно здесь находится основная часть класса. Конструктор создает объект шрифта и начинает с пустого изображения 20x20. Затем он заполняет изображение белым цветом и рисует на его поверхности закрашенный круг. Цвет круга будет зависеть от цвета, переданного в спрайт (красный, зеленый или синий). После того, как конструктор нарисует круг, он отображает белый шрифт поверх круга и вставляет его в центр круга. Наконец, он перемещает прямоугольник в строку и столбец, переданные в конструктор. Перемещение прямоугольника переместит весь камень в строку и столбец на доске.

```
1  class StoneSprite(pygame.sprite.Sprite):
2      def __init__(self, color, row, column,
3          piece_size, gem_value):
4           super().__init__()
5           WHITE = (255, 255, 255)
6           BLACK = (0, 0, 0)
7           small_font = pygame.font.Font(None, 16)
8
9           self.row = row
```

```
10          self.column = column
11
12          self.piece_size = piece_size
13          #  Создаём поверхность для спрайта
14          self.image = pygame.Surface(
15             [piece_size, piece_size])
16          self.image.fill(WHITE)
17
18          # Рисуем прямоугольник на поверхности спрайта
19          pygame.draw.circle(self.image, color,
20             (piece_size/2, piece_size/2),
21             int(piece_size/2.2))
22          self.gem_value = small_font.render(
23              str(gem_value), True, WHITE)
24          self.image.blit(self.gem_value, (piece_size/3,
25             piece_size/4))
26          self.rect = self.image.get_rect().move(
27              column*piece_size,
28              row*piece_size)
29
30
31      def update(self):
32          pass
33
34      # Рисуем спрайт на экране
35      def draw(self, surface):
36          surface.blit(self.image, self.rect)
37          surface.blit(self.gem_value, self.rect)
```

Теперь давайте посмотрим на спрайт пожирателя камней. В этом спрайте мы представляем новую библиотеку под названием pyganim (анимация pygame). Вам нужно будет установить библиотеку pyganim с помощью pip install:

```
1  pip install pyganim
```

Библиотека анимаций облегчает нам анимацию нашего пожирателя камней без необходимости обрабатывать его в игровом цикле. Способ, которым мы будем анимировать пожирателя камней, заключается в том, чтобы он открывал и закрывал рот, чтобы он выглядел так, как будто он ест камни (что-то вроде PacMan!). Для этого нам нужны только два изображения: едок с открытым ртом и едок с закрытым ртом.

Библиотека анимации pygame позволяет нам легко анимировать это с помощью метода PygAnimation, который берет изображения в паре со временем кадра в миллисекундах. Для нашего pacman мы чередуем два изображения каждые 250 миллисекунд или четверть секунды. Это даст нам желаемый эффект открывания и закрывания рта пожирателя. Кроме того, поскольку наши изображения довольно большие, нам нужно уменьшить их масштаб до размера ячейки сетки. Мы могли бы сделать это вручную, изменив размер изображения, или мы можем использовать функцию масштабирования, предоставленную нам библиотекой анимации. Мы решили уменьшить размер с помощью функции масштабирования.

Чтобы воспроизвести анимацию, мы просто вызываем play для объекта анимации, и он будет запускать анимацию открытия и закрытия рта едоков на протяжении всей игры.

```
import pygame
import pyganim

class PlayerSprite(pygame.sprite.Sprite):
    def __init__(self, row, column, piece_size):
        super().__init__()

        self.row = row
        self.column = column
        self.piece_size = piece_size
        self.anim = pyganim.PygAnimation(
            [("pacopen.png", 250), ("pacclose.png", 250)])
        self.anim.scale((piece_size, piece_size))
        self.anim.play()
        self.rect = pygame.Rect(
            column*piece_size, row*self.piece_size,
            self.piece_size, self.piece_size)

    def update(self):
        self.rect = self.anim.getRect().move(
            self.column*self.piece_size,
            self.row*self.piece_size)

    def draw(self, surface):
        self.anim.blit(surface, self.rect)
```

Обратите внимание, что в объекте пожирателя функция обновления не пуста. Причина этого в том, что каждый раз, когда в игре нажимается клавиша со стрелкой, мы обновляем позицию строки или столбца игроков. Мы должны вызвать update для объекта player, чтобы переместить прямоугольник его положения в новое положение ячейки сетки, определяемое нажатой клавишей. Чтобы нарисовать игрока, мы просто перенесли анимированный объект на игровую поверхность и поместили его в правильное положение на игровом поле.

Другой спрайт, который мы часто обновляем, — это спрайт времени. Этот спрайт покажет оставшееся время, оставшееся пользователю в игре. В основном цикле мы обновляем время, оставшееся после каждой итерации цикла:

```
 # Время обновления
    game_time.update_time(time_limit,
         pygame.time.get_ticks() - start_time)
```

TimeSprite обновляет свое внутреннее время и позже использует это время в функции обновления для отображения этого времени в объекте шрифта, содержащем оставшееся время:

```
import pygame


class TimeSprite(pygame.sprite.Sprite):
    def __init__(self):
        super().__init__()
        BLACK = (0, 0, 0)
        self.time = 0
        self.small_font = pygame.font.Font(None, 16)
        self.image = self.small_font.render(
            f'Time: {self.time}', True, BLACK)
        self.rect = self.image.get_rect().move(280, 15)


    def update(self):
        BLACK = (0, 0, 0)
        # обновить изображение времени
        self.image = self.small_font.render(
            f'Time: {self.time}',
            True, BLACK)
        self.rect = self.image.get_rect().move(280, 15)


    def draw(self, surface):
```

```
25            # Рисуем время на экране
26          surface.blit(self.image, self.rect)
27
28      def update_time(self, time_limit, time_in_millisecond\
29  s):
30
31          # рассчитать оставшееся время
32          calculated_time = int(time_limit -
33            (time_in_milliseconds / 1000))
34
35          # не нужно опускаться ниже 0
36          if calculated_time < 0:
37              calculated_time = 0
38          self.time = calculated_time
```

ScoreSprite похож на TimeSprite. Он содержит функцию для обновления счета, а затем в функции обновления он создает изображение для рисования счета с помощью self.score.

```
1   import pygame
2
3   class ScoreSprite(pygame.sprite.Sprite):
4       def __init__(self):
5           super().__init__()
6           BLACK = (0, 0, 0)
7           self.score = 0
8           self.small_font = pygame.font.Font(None, 16)
9          #нужно исходное изображение для определения прямоугольника
10          self.image = self.small_font.render(
11              f'Score: {self.score}', True, BLACK)
12         # получить прямоугольник, ограничивающий счет
13          self.rect = self.image.get_rect().move(0, 0)
14
15      def update(self):
16          BLACK = (0, 0, 0)
```

```
        self.image = self.small_font.render(
            f'Score: {self.score}', True, BLACK)
        # пересчитываем прямоугольник
        # так как изображение изменилось
        self.rect = self.image.get_rect().move(0, 0)

    def draw(self, surface):
        # Рисуем спрайт на экране
        surface.blit(self.image, self.rect)


    def update_score(self, score):
        self.score = score
```

Отслеживание рекорда

HiScoreSprite отвечает за отслеживание рекордов игры. Он отличается от Score Sprite тем, что запоминает наивысший балл и обновляется только при достижении более высокого балла. Счет хранится в файле, поэтому он запоминается, даже если пользователь выключит свой компьютер.

Класс обряда HiScoreSp начинается с инициализации переменной score, которая является наивысшим результатом, достигнутым игроком. Он открывает файл с именем hiscore.txt и считывает сохраненную в нем оценку, преобразует ее в целое число и сохраняет как переменную оценки.

Затем класс настраивает отображение счета, создавая мелкий шрифт и отображая текст «HiScore: [score]» в изображение. Атрибуту rect присваивается позиция на экране, в данном случае (150, 0). Метод update вызывает метод update_high_score для обновления максимальной оценки. Подобно спрайту Score, метод draw рисует на экране максимальный счет.

Метод update_high_score используется для обновления наивысшего результата в игре. Он сравнивает новую оценку с текущей наивысшей оценкой и, если новая оценка выше, обновляет переменную оценки и текст, отображаемый на экране, записывает новую оценку в файл hiscore.txt и сохраняет его. Если новый счет не выше, он ничего не делает.

```
import pygame


class HiScoreSprite(pygame.sprite.Sprite):
    def __init__(self):
        super().__init__()
        WHITE = (255, 255, 255)
        BLACK = (0, 0, 0)
        f = open('files/hiscore.txt', 'r')
        self.hi_score = int(f.read())
        print(f'read hi score of {self.hi_score}')
        f.close()
        self.current_score = -1
        self.small_font = pygame.font.Font(None, 16)
        self.image = self.small_font.render(
            f'HiScore: {self.hi_score}', True, BLACK)
        self.rect = self.image.get_rect().move(150, 0)
    #   Рисуем спрайт на экране

    def update(self):
        self.update_high_score(self.current_score)

    def draw(self, surface):
        surface.blit(self.image, self.rect)

    def update_high_score(self, score):
        BLACK = (0, 0, 0)
        if self.hi_score < score:
            self.hi_score = score
            self.image = self.small_font.render(
                f'HiScore: {self.hi_score}', True, BLACK)
            self.rect = self.image.get_rect().move(150, 0)
```

```
33              print(f'write hi score of {self.hi_score}')
34              f = open('files/hiscore.txt', 'w')
35              f.write(str(score))
36              f.close()
37          else:
38              pass
```

Обнаружение столкновений

Существует встроенная функция rect.colliderect для обнаружения столкновений между спрайтами. Мы можем перебрать все камни на доске и использовать функцию colliderect, чтобы определить, сталкивается ли один из камней с прямоугольником игрока на доске.

Функция, которую мы создадим, называется detect_collisions и принимает три аргумента:

playerSprite: объект PlayerSprite, представляющий персонажа игрока.

group:группа StoneSprites, представляющая камни на доске.

piece_size:целое число, представляющее длину и ширину каждого спрайта, измеренную в пикселях.

Цель функции — проверить, не столкнулся ли спрайт игрока с каким-либо из спрайтов в группе.

Функция начинается с перебора всех спрайтов в группе с использованием метода .sprites(). Для каждого спрайта он создает прямоугольное представление спрайта игрока (playerRect) с помощью конструктора pygame.Rect. Позиция спрайта игрока вычисляется путем умножения его свойств строки и столбца на piece_size, а размер прямоугольника устанавливается равным (piece_size, piece_size).

Затем вызывается метод colliderect объекта playerRect для свойства rect текущего объекта StoneSprite, который зацикливается. Этот метод возвращает True, если прямоугольник спрайта игрока и текущий прямоугольник спрайта пересекаются, что означает, что произошло столкновение.

Если обнаружено столкновение, функция возвращает текущий каменный спрайт. Если цикл завершается без обнаружения коллизии, возвращается None, чтобы указать, что коллизии не произошло.

```
def detect_collisions(playerSprite: PlayerSprite, group: \
pygame.sprite.Group, piece_size: int):

    for sprite in group.sprites():
        # обнаружить столкновение со спрайтом
        playerRect = pygame.Rect((playerSprite.column *
            piece_size,
            playerSprite.row * piece_size),
            (piece_size, piece_size))
        if playerRect.colliderect(sprite.rect):
            return sprite
    return None
```

Почему мы просто не вытащили playerRect справа от самого спрайта? Поскольку мы использовали библиотеку анимаций и масштабировали изображения пожирателя камней, прямоугольник почему-то тоже не масштабировался. Чтобы обойти эту проблему, мы можем просто воссоздать прямоугольник положения пожирателя камней на основе строки, столбца и piece_size.

Космическое вторжение в PyGame

Введение

Space Invaders — классическая видеоигра, разработанная корпорацией Taito в 1978 году. Игра была разработана Томохиро Нисикадо, вдохновленным популярной игрой Breakout и классическим научно-фантастическим фильмом «Звездные войны».

Первоначально игра была выпущена в Японии, но быстро завоевала популярность во всем мире, став культурным феноменом в 1980-х годах. Простой игровой процесс и культовая 8-битная графика сделали ее фаворитом среди игроков и сделали ее классикой эпохи аркад.

В игре игроки управляют космическим кораблем, который должен побеждать волны инопланетных захватчиков, спускающихся с верхней части экрана. Сложность игры увеличивается по мере продвижения игрока, с более быстрыми и агрессивными атаками пришельцев.

Space Invaders стала не только хитом игровых автоматов, но и помогла запустить индустрию видеоигр и вызвала волну игр на космическую тематику. За прошедшие годы она была перенесена на множество платформ, включая домашние консоли и персональные компьютеры, что обеспечило ей неизменную популярность и статус игровой иконы.

В этой главе мы опишем, как воссоздать классическую игру с помощью pygame.

Цель игры

Цель игры состоит в том, чтобы победить волны инопланетных захватчиков, которые спускаются с верхней части экрана, стреляя в них из лазерной пушки, управляемой игроком. Игрок должен перемещать пушку влево или вправо, чтобы уклоняться от атак инопланетян и выстраивать выстрелы. Пушка может стрелять только одной пулей за раз, поэтому игрокам нужно тщательно рассчитать время выстрела, чтобы не перегрузиться.

Пришельцы перемещаются по экрану вперед и назад и постепенно приближаются к пушке игрока. Если инопланетяне достигают нижней части экрана, игра окончена. Игрок должен попытаться уничтожить всех пришельцев, прежде чем они достигнут дна, чтобы перейти на следующий уровень.

По мере прохождения игроком уровней пришельцы становятся быстрее и агрессивнее, появляются новые типы пришельцев с разными способностями. Некоторые инопланетяне двигаются быстрее или непредсказуемее, в то время как для победы над другими требуется несколько выстрелов.

В игре ограниченное количество жизней, поэтому игроки должны стараться избегать ударов пришельцев. Если в пушку игрока попадает лазерный луч пришельца, он теряет жизнь, и игра заканчивается, когда все жизни потеряны.

В целом, Space Invaders — простая, но захватывающая игра, которая требует быстрой реакции и точного расчета времени, чтобы добиться успеха.


MIDWAY'S
a Bally Co.
SPACE INVADERS
PLAY
SPACE INVADERS
SCORE ADVANCE TABLE
MYSTERY
POINTS
POINTS
1 PLAYER
START GAME
2 PLAYER
FIRE
LASER

Основной цикл

Подобно тому, как мы настроили наш основной цикл для игры с пожирателем камней, нам нужно сделать то же самое для вторжения в космос. Цикл состоит из реагирования на события клавиатуры в игре, а также отрисовки различных спрайтов и обнаружения столкновений между ними. Чтобы облегчить чтение основного цикла, мы разбили код на несколько высокоуровневых функций.

Первая функция высокого уровня, process_events, обрабатывает события клавиатуры для перемещения и стрельбы игрока. Вот другие функции высокого уровня и их описания:

handle_scoring - обновляет текущий счет на экране на основе счета игроков. Эта функция также обновляет рекорд и индикатор жизней.

handle_alien_movement - эта функция перемещает инопланетян по экрану и помогает инопланетянам изменить направление, когда они достигают края экрана. Они также двигаются вниз каждый раз, когда сталкиваются с краем.

handle_player_movement - обрабатывает движение игрока в соответствии с нажатой клавишей со стрелкой. Стрелка влево перемещает игрока влево, пока он не отпустит клавишу, и то же самое со стрелкой вправо.

handle_bullet - эта функция обрабатывает активную пулю, исходящую от игрока, пытающегося поразить инопланетян. Она перемещает пулю вверх по экрану с определенной скоростью каждый раз в игровом цикле. Он также проверяет, не столкнулась ли пуля с инопланетянином, с помощью функции **handle_alien_hit**.

check_for_bomb_activation - эта функция проверяет на основе случайно сгенерированного значения, сбросил ли инопланетянин бомбу. Если инопланетянин сбросил бомбу, это записывается в массив, чтобы позже обработать его движение.

handle_active_bombs перебирает массив бомб и рисует активные бомбы. Также проверяет, попала ли бомба в игрока, вызывая handle_player_hit.

draw_aliens - Рисует всех пришельцев. Инопланетяне рисуются путем прохода по строкам пришельцев, хранящихся в Alien_groups, а затем по каждому пришельцу в строке. Строка — это группа pygame.

handle_alien_speedup - handle_alien_speedup — по мере того, как игрок убивает все больше и больше пришельцев, скорость пришельцев после определенного порога увеличивается. В настоящее время этот порог наступает, когда пришельцев всего 25, затем когда остается только 5 пришельцев, и, наконец, когда остается только 1 пришелец.

```
while running:
    (player_x, player_y) = player.position
    window.fill(BLACK)

    running = process_events()  # process the keyboard

    if (game_over):
        show_game_over_prompt()
        pygame.display.flip()
        continue

    # подсчет очков
    handle_scoring(window, score, player_score)

    # переместить пришельцев
    handle_alien_movement()

    # переместить игрока
    handle_player_movement(window_width, player_left,
      player_right, player, player_x)

    # переместить пулю
    if bullet_active:
        handle_bullet(bullet, bullet_active)


```

```
27      # проверять активацию бомбы каждые 2 секунды
28      check_for_bomb_activation()
29
30      # обновить активные бомбы
31      handle_active_bombs(active_bombs)
32
33      #  рисуем пришельцев и проверяем на удаление
34      draw_aliens(window, alien_groups)
35
36
37      # проверь, не пора ли ускорить инопланетян
38      #  в зависимости от количества оставшихся инопланетян
39      handle_alien_speedup(total_aliens)
40
41      #  показать дисплей
42      pygame.display.flip()
43
44      # обновить время игры
45      game_time = pygame.time.get_ticks() - start_time
```

Игровые спрайты

Как и в случае с пожирателем камней, проще всего создать космическое вторжение, создав спрайты. Эта игра содержит следующие спрайты.

ImageSprite - базовый класс для всех спрайтов, которые загружают изображение и отслеживают положение.
PlayerSprite - пушка игрока, которой вы управляете, для стрельбы по инопланетянам.
BombSprite - спрайт, представляющий изображение бомбы, сброшенной инопланетянином.
BulletSprite - спрайт, показывающий пулю, выпущенную по захватчикам.
InvaderSprite - рисует захватчика, используя два изображения для анимации его движения.
MessageSprite - используется для рисования игры поверх сообщения.
ScoreSprite - отображает счет игры в верхней части экрана.
HighScoreSprite - тот же спрайт, который мы использовали в пожирателе камней для отслеживания рекорда.
LivesSprite - Рисует количество оставшихся жизней у игрока

Спрайт игрока

Давайте сначала подробно рассмотрим спрайт игрока. Этот код определяет класс с именем PlayerSprite, который расширяет ImageSprite. Класс PlayerSprite представляет управляемый игроком спрайт, который может двигаться влево или вправо, быть убитым и взорваться. Он имеет следующие атрибуты:

dead: логический флаг, указывающий, мертв спрайт или нет.
speed: число с плавающей запятой, представляющее скорость, с которой спрайт движется влево или вправо.
death_time: целое число, представляющее время, когда спрайт был убит.
animate_explosion: объект PygAnim, представляющий анимацию взрыва спрайта.

Класс PlayerSprite имеет следующие методы:

init(self, name, x, y): конструктор класса PlayerSprite. Он инициализирует положение спрайта и устанавливает его атрибуты в их начальные значения.
update: метод, который обновляет позицию спрайта.
kill: метод, который убивает спрайт, запуская анимацию взрыва.
draw: метод, рисующий спрайт на заданной поверхности. Если спрайт не умер, он вызывает метод draw() суперкласса для рисования изображения спрайта. Если спрайт мертв, он воспроизводит анимацию взрыва и проверяет, прошло ли достаточно времени для завершения взрыва.
move_left: метод, который перемещает спрайт влево, обеспечивая его положение.
move_right: метод, который перемещает спрайт вправо, регулируя его положение.

```
import pygame
import pyganim
from ImageSprite import ImageSprite

class PlayerSprite(ImageSprite):
    def __init__(self, name, x, y):
        super().__init__(name, x, y)
        self.dead = False
        self.speed = .1
        self.death_time = 0
        self.animate_explosion = pyganim.PygAnimation(
            [("images/shipexplosion/frame1.gif", 250),
             ("images/shipexplosion/frame2.gif", 250),
             ("images/shipexplosion/frame3.gif", 250),
             ("images/shipexplosion/frame4.gif", 250),
             ("images/shipexplosion/frame5.gif", 250),
             ("images/shipexplosion/frame6.gif", 250),
             ("images/shipexplosion/frame7.gif", 250),
             ("images/shipexplosion/frame8.gif", 250),],
             loop=False)

    # просто вызовите суперкласс, чтобы настроить прямоугольник
    def update(self):
        super().update()

    # Рисуем спрайт на экране
    def kill(self):
        self.animate_explosion.play()
        self.dead = True
        self.death_time = pygame.time.get_ticks()


    def draw(self, surface):
        if not self.dead:
            super().draw(surface)
```

```
36          else:
37              self.animate_explosion.blit(surface,
38                 self.rect)
39              if (pygame.time.get_ticks() -
40                self.death_time) > 5000:
41                  self.dead = False
42
43      def move_left(self):
44          (x, y) = self.position
45          self.position = (x - self.speed, y)
46
47      def move_right(self):
48          (x, y) = self.position
49          self.position = (x + self.speed, y)
```

Взрыв игрока

Объект animate_explosion использует библиотеку pyganim для управления анимацией взрыва корабля, быстро отрисовывая каждый из 8 кадров взрыва один раз. Анимация инициализируется со всей информацией, необходимой для воспроизведения кадров взрыва, и о том, как долго будет отображаться каждый кадр:

```
1   self.animate_explosion = pyganim.PygAnimation(
2               [("images/shipexplosion/frame1.gif", 250),
3                ("images/shipexplosion/frame2.gif", 250),
4                ("images/shipexplosion/frame3.gif", 250),
5                ("images/shipexplosion/frame4.gif", 250),
6                ("images/shipexplosion/frame5.gif", 250),
7                ("images/shipexplosion/frame6.gif", 250),
8                ("images/shipexplosion/frame7.gif", 250),
9                ("images/shipexplosion/frame8.gif", 250),],
10               loop=False)
```

Чтобы воспроизвести анимацию, мы просто вызываем метод play для animate_-explosion внутри нашего метода kill:

```
1  def kill(self):
2          self.animate_explosion.play()
3          self.dead = True
4          self.death_time = pygame.time.get_ticks()
```

Invader Sprite - Захватчик Спрайт

Далее давайте взглянем на InvaderSprite, который рисует анимированного инопланетянина, движущегося по экрану. Класс InvaderSprite наследуется от pygame.sprite.Sprite, который является базовым классом для всех спрайтов в Pygame. Метод init() инициализирует различные переменные экземпляра, такие как два спрайта изображения (name1 и name2), которые используются для рисования и анимации пришельца, спрайт изображения взрыва, родительская строка, представленная группой спрайтов, скорость пришельца, его текущее направление (влево или вправо), его начальное положение и его точки. Метод update() обновляет положение двух спрайтов изображения, представляющих инопланетянина, в зависимости от его текущего положения. Метод draw() рисует текущий спрайт изображения на поверхности игры. Методы move_left(), move_right() и move_down() перемещают инопланетянина влево, вправо или вниз соответственно. Метод switch_image() переключается между двумя спрайтами изображения инопланетянина в зависимости от номера изображения, переданного в этот метод. Методы get_width() и get_height() возвращают ширину и высоту текущего спрайта изображения инопланетянина. Метод kill() переключает спрайт изображения на спрайт взрыва и помечает инопланетянина как мертвого.

```
import pygame
from ImageSprite import ImageSprite
from BombSprite import BombSprite


class InvaderSprite(pygame.sprite.Sprite):
    def __init__(self, name1, name2, x, y, parent, points\
):
        super().__init__()
        self.imageSprite1 = ImageSprite(name1, x, y)
        self.imageSprite2 = ImageSprite(name2, x, y)
        self.explosion =    ImageSprite('explosion', x, y)
        self.imageSprite = self.imageSprite1
        self.parent = parent
        self.speed = .01
        self.currentDirection = 'right'
        self.position = (x, y)
        self.rect = self.imageSprite.image.get_rect()
          .move(self.position)
        self.dead = False
        self.death_time = 0
        self.bomb_active = False
        self.points = points

    # обновить положение двух спрайтов,
    # представляющих инопланетянина
    def update(self):
        self.imageSprite.rect = self.imageSprite
            .image.get_rect()
            .move(self.position)
        self.imageSprite1.rect = self.imageSprite.rect
        self.imageSprite2.rect = self.imageSprite.rect

    # Рисуем спрайт на экране

```

```
36      def draw(self, surface):
37          self.imageSprite.draw(surface)
38
39      def move_left(self):
40          (x, y) = self.position
41          self.position = (x - self.speed, y)
42
43      def move_right(self):
44          (x, y) = self.position
45          self.position = (x + self.speed, y)
46
47      def move_down(self):
48          (x, y) = self.position
49          self.position = (x, y + 10)
50
51      #переключение между двумя изображениями, изображающими инопланетянина
52      def switch_image(self, imageNumber):
53          if self.dead == True: return
54          if (imageNumber == 1):
55              self.imageSprite = self.imageSprite1
56          else:
57              self.imageSprite = self.imageSprite2
58
59      def get_width(self):
60          return self.imageSprite.get_width()
61
62      def get_height(self):
63          return self.imageSprite.get_height()
64
65      def kill(self):
66          self.imageSprite = self.explosion
67          self.imageSprite.draw(self.imageSprite.image)
68          self.imageSprite.update()
69          self.dead = True
70          self.death_time = pygame.time.get_ticks()
```

Как движется инопланетянин?

Класс InvaderSprite использует здесь трюк, чтобы выполнить анимацию открытия и закрытия инопланетянина. В любой момент времени свойство self.imageSprite содержит ссылку либо на изображение инопланетянина с открытой клешней, либо на изображение пришельца с закрытой клешней. Программа либо передает 1, либо 0 в метод switch_image и соответственно присваивает спрайту изображения два или одно из двух изображений. Когда придет время рисовать инопланетянина, независимо от того, чему назначен imageSprite в это время, будет тот, который будет нарисован на поверхности.

Пуля Спрайт

Когда игрок нажимает стрелку вверх, активируется зеленая пуля, которая может выстрелить в инопланетянина, если столкнется с ним. Изображение пули создается простым заполнением прямоугольной поверхности пули зеленым цветом.

```
class BulletSprite(pygame.sprite.Sprite):
    def __init__(self, x, y, bullet_width,
           bullet_height, speed):
        super().__init__()
        WHITE = (255, 255, 255)
        GREEN = (0, 255, 0)
        BLACK = (0, 0, 0)
        small_font = pygame.font.Font(None, 16)

        self.position = (x, y)
        self.speed = speed

       # Создаём поверхность для спрайта
        self.image = pygame.Surface(
           [bullet_width, bullet_height])
        self.image.fill(GREEN)
```

```
        # Рисуем прямоугольник на поверхности спрайта
        self.rect = self.image.get_rect().move(x, y)

    # перемещаем спрайт в соответствии с положением пули
    def update(self):
        (x, y) = self.position
        self.rect = self.image.get_rect().move(x, y)

    # Рисуем спрайт на экране
    def draw(self, surface):
        surface.blit(self.image, self.rect)
```

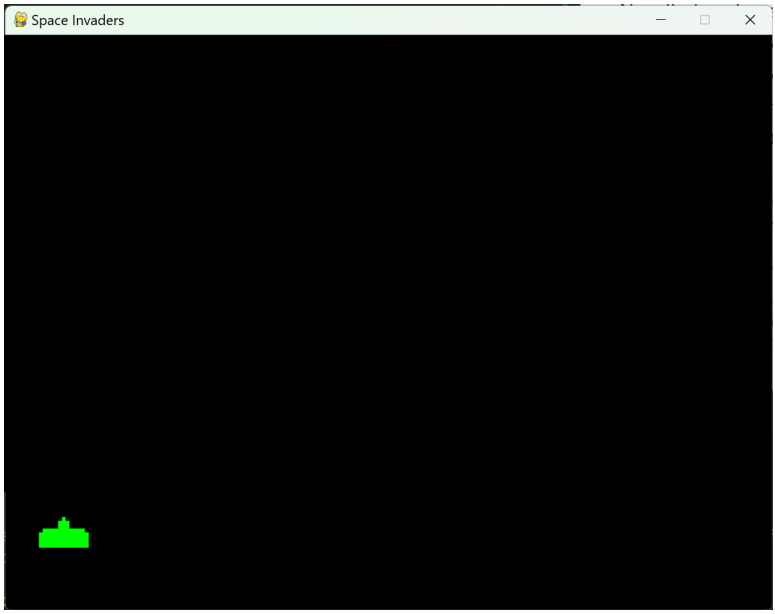
Спрайт Бомба

Спрайт-бомба — это спрайт, который падает с инопланетян. Он наносится на поверхность изображения в виде ряда диагностических белых линий, используемых для формирования формы молнии.

```
class BombSprite(pygame.sprite.Sprite):
    def __init__(self, x, y, bullet_width,
       bullet_height, speed, parent):
        super().__init__()
        WHITE = (255, 255, 255)
        GREEN = (0, 255, 0)
        BLACK = (0, 0, 0)
        small_font = pygame.font.Font(None, 16)

        self.position = (x, y)
        self.speed = speed
        self.parent = parent

        # Создаём поверхность для спрайта
        self.image = pygame.Surface(
          [bullet_width, bullet_height])
```

```
17          pygame.draw.lines(self.image, WHITE, True,
18              [(0, 0), (5, 5), (0, 10), (10, 15)], 1)
19
20          # Рисуем прямоугольник на поверхности спрайта
21          self.rect = self.image.get_rect().move(x, y)
22
23          # обновить бомбу в соответствии с текущей позицией
24      def update(self):
25          (x, y) = self.position
26          self.rect = self.image.get_rect().move(x, y)
27
28      # Рисуем спрайт на экране
29
30      def draw(self, surface):
31          surface.blit(self.image, self.rect)
```

Перемещение игрока



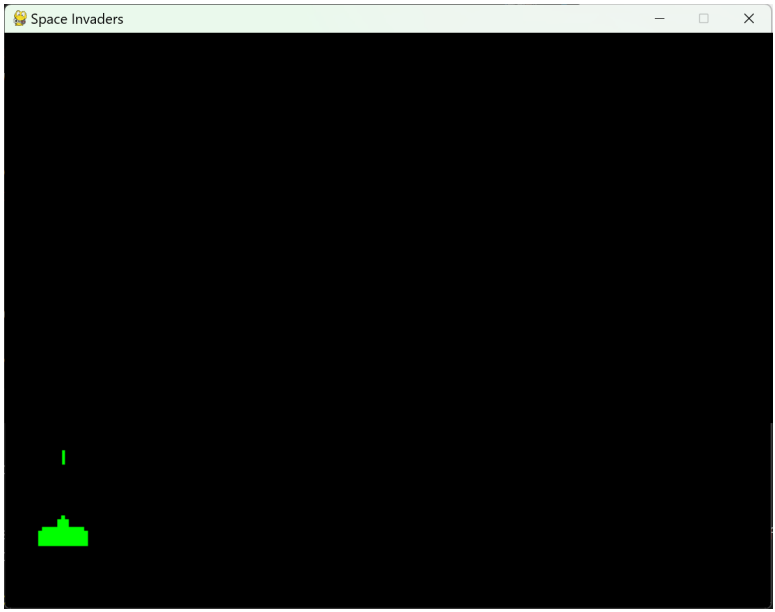
Теперь, когда у нас есть спрайт игрока, мы можем перемещать игрока в соответствии с нажатиями клавиш. В нашем методе process_events мы будем фиксировать нажатия клавиш. Если пользователь нажимает стрелку влево, мы отмечаем флаг, говорящий, что игрок движется влево. Когда отпустят стрелку влево, флаг будет очищен. То же самое происходит и с правой стрелкой. Если пользователь нажмет стрелку вправо, будет отмечен флажок, указывающий, что игрок движется вправо. Как только пользователь отпустит стрелку вправо, этот флаг будет очищен.

```
def process_events():
    global player_left, player_right
    (player_x, player_y) = player.position
    running = True
    for event in pygame.event.get():
        if event.type == pygame.KEYDOWN:
            # Проверяем, двигался ли игрок
            if event.key == pygame.K_LEFT
              and player_x > 0:
                player_left = True
                player_right = False
            elif event.key == pygame.K_RIGHT and
               player_x < window_width:
                player_right = True
                player_left = False
        elif event.type == pygame.KEYUP:
            player_left = False
            player_right = False
    return running
```

Внутри нашего игрового цикла мы используем флаги player_left и player_right, установленные методом process_events, для перемещения игрока. Мы консолидируем поведение игрока в методе handle_player_movement, который принимает параметры, необходимые для перемещения спрайта игрока. Обратите внимание, что мы проверяем границы и если игрок выйдет за границы экрана, мы не разрешаем движение. Также обратите внимание, что если игрок помечен как мертвый, нам не нужно его перемещать. Игрок перемещается со скоростью, соответствующей скорости игрока. Как только скорость добавляется или вычитается из положения игрока, спрайт игрока обновляется и перерисовывается.

```
def handle_player_movement(window_width, player_left,
  player_right, player, player_x):
    if (player.dead):
        pass
    elif player_left:
        if (player_x - player.speed) > 0:
            player.move_left()
    elif player_right:
        if (player_x + player.speed) <
            window_width - player.get_width():
            player.move_right()

    player.update()
    player.draw(window)
```

Выстрел пулей



Пуля выстреливается с помощью клавиши со стрелкой вверх. После того, как пуля выпущена, ее нельзя выстрелить повторно, пока пуля не попадет в пришельца или не пройдет за верхнюю часть экрана. Пуля активируется стрелкой вверх, поэтому мы ищем это в нашем методе process_event:

```
def process_events():
    global player_left, player_right, bullet_active
    (player_x, player_y) = player.position
    running = True
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
        elif event.type == pygame.KEYDOWN:
            #Проверяем, двигался ли игрок
            if event.key == pygame.K_UP:
```

```
                if bullet_active == False:
                    bullet_active = True
                    bullet.position = (player_x + 30,
                       player_y - 20)
                    bullet_fire_sound.play()
                    ...
    return running
```

После того, как мы активировали пулю, мы можем обрабатывать ее состояние и движение в методе основного цикла с именем handle_bullet. В основном цикле, если пуля активна, мы вызываем handle_bullet для рисования движущейся пули. Обработчик пули принимает спрайт пули и флаг bullet_active, которые мы установили в методе process_events. Y-позиция пули устанавливается путем вычитания скорости пули из текущей y-позиции пули и обновления позиции пули. Если позиция bullet_y находится за пределами верхней части экрана (при y=0), мы устанавливаем для флага bullet_active значение false.

```
def handle_bullet(bullet, bullet_active):
    (bullet_x, bullet_y) = bullet.position
    bullet_y = bullet_y - bullet.speed
    bullet.position = (bullet_x, bullet_y)
    bullet.update()
    bullet.draw(window)
    if (handle_alien_hit(bullet_x, bullet_y)):
        bullet_active = False
        bullet.position = (0, 0)

    if (bullet_y < 0):
        bullet_active = False

    return bullet_active
```

Проверка инопланетных попаданий

Метод handle_bullet также проверяет, попали ли мы в инопланетянина, вызывая метод handle_alien_hit с текущими координатами пули. handle_alien_hit не только проверяет всех инопланетян, чтобы увидеть, не попал ли в кого-то из них пуля, но и обрабатывает убийство пришельца. handle_alien_hit перебирает все строки инопланетян и каждого инопланетянина в каждой строке и проверяет, находится ли позиция пули внутри цели инопланетянина. Если это так, инопланетный спрайт погибает и воспроизводится звук взрыва. Также обновляется счет игрока

```
def handle_alien_hit(bullet_x, bullet_y):
    global gems_collected, player_score, bullet,
       alien_groups
    for alien_group in alien_groups:
        for alien in alien_group:
            (x, y) = alien.position
            if bullet_x > x and
              and bullet_x < x + alien.get_width()
              and bullet_y > y and
                  bullet_y < y + alien.get_height():
                alien.kill()
                alien.death_time = pygame.time.get_ticks()
                alien_dying.play()
                player_score += alien.points
                return True
    return False
```

Когда инопланетянин умирает, он заменяется изображением взрыва пришельца и помечается как мертвый, как показано в InvaderSprite ниже. Также отмечено время смерти, поэтому мы можем поддерживать взрыв в течение определенного периода времени.

```
class InvaderSprite(pygame.sprite.Sprite):
    ...
    def kill(self):
        self.imageSprite = self.explosion
        self.imageSprite.draw(self.imageSprite.image)
        self.imageSprite.update()
        self.dead = True
        self.death_time = pygame.time.get_ticks()
```

В нашем основном цикле мы вызываем check_for_removal каждый раз в цикле. Если инопланетянин мертв более 1/4 секунды (или 250 миллисекунд), то мы удаляем его из строки. Если из ряда выбиты все пришельцы, то удаляем сам ряд.

```
def check_for_removal(alien):
        if alien.death_time > 0
          and alien.death_time + 250 <
             pygame.time.get_ticks():
          alien.parent.remove(alien)
          if (len(alien.parent) == 0):
              alien_groups.remove(alien.parent)
```

Рисование инопланетян

Чтобы обнаружить попадание инопланетян, вам нужно, чтобы инопланетяне попали! В этом разделе мы опишем, как нарисованы инопланетяне. Мы уже рассмотрели спрайт захватчика, следующим шагом будет отрисовка всех видов захватчиков в разных рядах на экране с помощью класса InvaderSprite. Обратите внимание, что два нижних ряда захватчиков представляют собой один и тот же спрайт и приносят 10 очков, а 2-й и третий ряды — это разные типы захватчиков, приносящие 20 очков. Верхний ряд также является уникальным захватчиком стоимостью 30 очков.

Сначала мы создадим конфигурацию пришельцев, которую вы видите на рисунке выше, используя метод create_aliens. В нашем игровом цикле мы будем обновлять движение пришельцев с помощью функции handle_alien_movement и рисовать пришельцев с помощью метода draw_aliens.

create_aliens рисует все 5 рядов пришельцев. Код определяет список с именем Alien_names, который содержит имена разных типов пришельцев в игре. Каждый тип имени пришельца относится к образу пришельца, созданному с помощью InvaderSprite. Для каждого InvaderSprite есть два изображения файла: открытый инопланетный спрайт и закрытый инопланетный спрайт, оканчивающийся на c.

Функция create_aliens() используется для создания настоящих пришельцев в игре. Она начинается с создания пустого списка с именем Alien_groups.

Затем функция использует цикл для создания пяти рядов пришельцев. В каждой строке функция использует еще один цикл для создания 11 инопланетян в этой строке. При создании пришельца он конструирует InvaderSprite с открытой и закрытой версиями конкретного пришельца.

Конструктор InvaderSprite также назначает каждому инопланетянину позицию на экране для InvaderSprite, которая определяется его строкой и столбцом в сетке пришельцев. Также в конструкторе назначается родительская группа и очки, которые инопланетянин получает при попадании в него. Каждый инопланетянин добавляется в объект pygame.sprite.Group(), который является контейнером для нескольких спрайтов в Pygame.

Наконец, каждый объект pygame.sprite.Group() добавляется в список Alien_-groups, который используется для отслеживания всех строк пришельцев в игре.

В целом, этот код настраивает различные типы инопланетян в игре и создает их на экране в виде сетки, используя функциональные возможности спрайтов Pygame.

```
## словарь для подсчета очков
score_dict =  {
    'invader1': 30,
    'invader2': 20,
    'invader3': 10
}

alien_names = ['invader1', 'invader2', 'invader2', 'invad\
er3', 'invader3' ]
```

```
10  def create_aliens():
11      global alien_groups
12      alien_groups = []
13      for i in range(0, 5):
14          alien_group = pygame.sprite.Group()
15          for j in range(0, 11):
16              alien = InvaderSprite(alien_names[i],
17                  alien_names[i] + 'c',
18                  30 + (j * 60), 60 + i*60, alien_group,
19                  score_dict[alien_names[i]])
20              alien_group.add(alien)
21          alien_groups.append(alien_group)
22
```

Итак, мы нарисовали наши ряды инопланетян, теперь как их перемещать? Для этого мы вызываем handle_alien_movement в нашем основном цикле. Этот метод начинается с поиска самого левого пришельца и самого правого пришельца. Причина, по которой он их находит, заключается в том, что ему нужно знать, какой инопланетянин заставит пришельцев изменить направление и спуститься на ступеньку ниже. Нам также нужно знать самого нижнего инопланетянина, чтобы сказать, когда пришельцы приземляются. Функция move_aliens, вызываемая внутри handle_alien_-motion, выполняет реальное движение инопланетянина, которое мы обсудим чуть позже. Следующая часть кода перебирает всех пришельцев и выполняет анимацию открывания и закрывания когтей. Цикл вызывает switch_image для каждого спрайта захватчика и передает общее игровое время, деленное на частоту моргания, а затем на модуль 2, который будет генерировать 1 или 0. 1 или 0 представляют, открывает ли пришелец свои когти или закрывает их. Чем выше частота моргания, тем медленнее инопланетянин будет открывать и закрывать когти. Позже мы изменим скорость моргания по мере уменьшения популяции пришельцев, чтобы ускорить анимацию пришельцев.

Мы также можем использовать игровое время, чтобы определить, когда воспроизводить инопланетный звук, когда он перемещается по экрану. Мы проверяем модуль py game_time 400, чтобы воспроизводить звук примерно каждые 1/2 секунды, когда результат равен нулю. Последний бит кода устанавливает позицию всех пришельцев, определяемую флагами, вычисленными методом move_aliens. Независимо от того, на что установлены флаги, все инопланетяне будут следовать направлению этих флагов, поскольку все пришельцы движутся по экрану в тандеме.

Последний фрагмент кода py move_aliens_down = False устанавливает флаг, который указывает инопланетянам двигаться вниз, в значение false. Мы должны сбросить этот флаг после того, как мы уже переместили инопланетян вниз для того, чтобы пришельцы переместились только на одну строку вниз, а затем продолжили движение влево или вправо. В противном случае инопланетяне двинулись бы вниз довольно быстро!

```
def handle_alien_movement():
    global game_time, move_aliens_down, alien_groups,
           move_aliens_right
    alien_rightmost = find_rightmost_alien()
    alien_leftmost = find_leftmost_alien()
    alien_bottommost = find_bottommost_alien()
    (move_aliens_right, move_aliens_down) =
        move_aliens(
          alien_leftmost,
          alien_rightmost,
          alien_bottommost,
          move_aliens_right,
          move_aliens_down)

    # делаем анимацию
    for alien_group in alien_groups:
        for next_alien in alien_group:
            next_alien.switch_image(
                 int(game_time/blink_speed) % 2 )
            next_alien.update()

    #проигрывать инопланетный звук каждые полсекунды
    if game_time % 400 == 0 and aliens_exist():
        alien_movement.play()
```

```
25
26      for alien_group in alien_groups:
27          for alien in alien_group:
28              (x,y) = alien.position
29              if move_aliens_right:
30                  alien.move_right()
31              else:
32                  alien.move_left()
33              if move_aliens_down:
34                  alien.move_down()
35              alien.update()
36
37  #  сбросить перемещение инопланетянина вниз, мы только хотим,
38  #    чтобы они  переместились на одну строку вниз.
39      move_aliens_down = False
```

move_aliens, показанный ниже, показывает, как мы вычисляем определение движения множества всех пришельцев. Сначала мы получаем положение самого дальнего левого инопланетянина с именем first_alien и самого дальнего и самого правого инопланетянина с именем last_alien. Затем, если мы в данный момент двигаем пришельцев вправо, мы проверяем, попали ли пришельцы в правую часть экрана. Если они наткнулись на границу, двигаясь вправо, пришло время сменить направление и двигаться вниз. В результате, если инопланетянин врежется в правую стену, мы установим для флага move_right значение false, чтобы показать, что теперь мы движемся влево. Мы также установим флаг move_down в значение true, если, конечно, мы уже не достигли нижней части экрана.

Мы также делаем аналогичную проверку координат first_alien, если мы в данный момент движемся влево (или не движемся вправо). если первая позиция пришельца выходит за пределы левой части экрана, пришло время изменить направление пришельцев и переместить их вправо и вниз.

```
def move_aliens(leftmost, rightmost, bottommost, move_rig\
ht, move_down):
    global game_time

    last_alien = rightmost
    first_alien = leftmost

    # ничего не делать, если первый и последний инопланетянин пуст

    if (last_alien is None) or (first_alien is None):
        return (move_right, move_down)

    # получить координаты позиции для первого
    # и последнего инопланетянина
    (last_alien_x, last_alien_y) = last_alien.position
    (first_alien_x, first_alien_y) = first_alien.position

    # если мы уже движемся вправо, определить, должны ли мы
    # продолжать или двигаться вниз и в обратном направлении

    if move_right:
        if last_alien_x + last_alien.speed >=
            window_width - (last_alien.rect.width + 5):
            move_right = False
            if last_alien_y + last_alien.speed <

              window_height - last_alien.rect.height:
                        if (bottommost.position[1] <
                  window_height - 50):
                    move_down = True

    return move_right, move_down

    # если мы уже движемся влево, определить должны ли мы
    # продолжать или двигаться вниз и в обратном направлении\
```

```
ion
    if not move_right:
        if first_alien_x - first_alien.speed <= 0:
            move_right = True
            if first_alien_y + first_alien.speed <
             window_height - first_alien.rect.height:
              if (bottommost.position[1] <
                window_height - 50):
                    move_down = True



    return move_right, move_down
```

Бомбардировка игрока

Игра не была бы очень сложной, если бы инопланетяне просто перемещались по экрану, как сидящие утки, так что давайте добавим к картинке несколько инопланетных бомб. Вышеупомянутая бомбардировка инопланетянами выполняется двумя функциями в нашем основном цикле: check_for_bomb_activation() и handle_active_bombs. Первый метод проверяет, собирается ли инопланетянин сбросить бомбу. Чтобы определить, бомбит ли нас пришелец, мы перебираем всех пришельцев и генерируем случайное число для этого пришельца. Если случайное число падает ниже частоты бомбардировки, то инопланетянин бомбит, и мы помечаем этого пришельца как имеющего активную бомбу. Также отмечаем положение инопланетянина в момент определения.

```
def check_for_bomb_activation():
     global game_time, active_bombs,
       alien_groups, bomb_frequency
     if (game_time/1000 % 2 == 0):
        if len(alien_groups) <= 0: return
        # найти всех инопланетян, которые в данный момент
        #  имеют возможность бомбить под собой
        bombing_aliens = [alien
         for alien_group in alien_groups
           for alien in alien_group.sprites()]
        for alien in bombing_aliens:
            # не сбрасывать бомбу, если
            # инопланетянин уже сбрасывает одну
           # нужно дать нашему игроку шанс побороться!
            if (alien.bomb_active == False):
           # если случайно сгенерированное число
                # ниже частоты бомбы, которую
                # сбрасывает инопланетянин
                activate_bomb = random.randint(0, 100)
                   < bomb_frequency
                if activate_bomb and
                   alien.bomb_active == False:
                    alien.bomb_active = True
```

```
                    newestBomb =
                      BombSprite( 0, 0, 5, 15, .03, alien)
                    newestBomb.position
                      = (alien.position[0] +
                         alien.get_width()/2,
                      alien.position[1]
                         + alien.get_height())
                    active_bombs.append(newestBomb)
```

Следующая функция, handle_active_bombs, будет продолжать перемещать каждую активную бомбу вниз от горизонтального положения, с которого она началась. если положение бомбы превышает высоту окна, мы удаляем бомбу и помечаем бомбу как неактивную. Нам также нужно проверить, не попал ли в игрока бомба, и обработать смерть игрока. Если игрок убит, мы, по сути, делаем то же самое, что делали, когда бомба попала в нижнюю часть экрана. Отмечаем бомбу как неактивную, сбрасываем ее положение и удаляем бомбу из коллекции активных бомб. Обратите внимание, что мы удаляем бомбы, добавляя бомбы в отдельный список удаления, а затем удаляя их. Причина, по которой мы это делаем, заключается в том, что мы не хотим изменять список active_bomb, пока мы его просматриваем. Удаление элементов из списка во время циклического просмотра этих элементов может привести к странному поведению.

```
def handle_active_bombs(active_bombs):
    bombs_to_remove = []
    for bomb in active_bombs:
        (bomb_x, bomb_y) = bomb.position
        bomb_y = bomb_y + bomb.speed
        bomb.position = (bomb_x, bomb_y)
        bomb.update()
        bomb.draw(window)
        if (bomb_y > window_height):
            bomb.parent.bomb_active = False
            bomb.position = (0, 0)
            bombs_to_remove.append(bomb)
```

```
13          if (handle_player_hit(bomb_x, bomb_y)):
14              bomb.parent.bomb_active = False
15              bomb.position = (0, 0)
16              bombs_to_remove.append(bomb)
17
18      # удалить бомбы, отмеченные для удаления,
19      # из списка активных бомб
20      if (len(bombs_to_remove) > 0):
21          if (len(active_bombs) > 0):
22              for bomb in bombs_to_remove:
23                  active_bombs.remove(bomb)
```

Ускорение движения пришельцев

По мере прохождения игры мы хотим бросить вызов игроку, угрожая приземлиться инопланетянам быстрее, когда они перемещаются по экрану. Мы создали метод handle_alien_-speedup и вызвали его в основном цикле, чтобы увеличить скорость пришельцев в зависимости от того, сколько пришельцев осталось. Приведенный ниже код будет получать общее количество пришельцев, вызывая total_aliens, и на основе этого общего количества мы изменим скорость пришельцев. Если инопланетяне достигли в общей сложности 20, 5 или 1, скорость пришельцев будет увеличена, а также скорость анимации открытия и закрытия когтей пришельцами (это делает пришельцев более угрожающими!)

```
1   def handle_alien_speedup(total_aliens):
2       global blink_speed, bomb_frequency,
3         first_speed_up, second_speed_up, third_speed_up
4
5       if (total_aliens() == 20):
6         if first_speed_up == False:
7           blink_speed = 200
8           bomb_frequency = 10
9           speed_up_aliens()
10          first_speed_up = True
```

```
11
12      if (total_aliens() == 5):
13        if second_speed_up == False:
14          blink_speed = 100
15          bomb_frequency = 20
16          speed_up_aliens()
17          second_speed_up = True
18
19      if (total_aliens() == 1):
20        if third_speed_up == False:
21          bomb_frequency = 40
22          blink_speed = 50
23          speed_up_aliens(2.0)
24          third_speed_up = True
```

Предоставленный код определяет функцию под названием speed_up_aliens, которая принимает необязательный аргумент-фактор со значением по умолчанию 1,0. Эта функция перебирает всех пришельцев, которые существуют в разных группах пришельцев, и увеличивает их скорость на коэффициент, который рассчитывается с использованием заданного параметра фактора. Если параметр фактора не передан, скорость всех пришельцев во всех группах будет увеличена на фиксированный коэффициент 0,01. Поскольку начальная скорость всех пришельцев равна 0,01, первое ускорение удваивает скорость пришельцев.

Функция speed_up_aliens сначала проходит по каждой группе Alien_- в списке Alien_groups, который представляет каждую строку пришельцев. В пределах каждой Alien_group функция перебирает каждого пришельца в этой группе, а затем обновляет атрибут скорости пришельца, добавляя результат 0,01, умноженный на значение коэффициента, к текущей скорости. Результат этого расчета будет добавлен к текущему значению атрибута Alien.speed, эффективно увеличив скорость пришельца на рассчитанный коэффициент.

Обратите внимание, что параметр factor является необязательным, и функция все равно будет работать, даже если аргумент не передан. В этом случае будет использоваться значение коэффициента по умолчанию 1,0, и скорость всех пришельцев будет увеличена на 0,01 * 1,0 или 0,01. Если в функцию передано значение фактора, скорость всех инопланетян вместо этого будет увеличена на этот фактор. Последний инопланетянин ускорен в 3 раза по сравнению с исходной скоростью (0,01 + 0,01 * 2,0).

```
def speed_up_aliens(factor = 1.0):
    for alien_group in alien_groups:
        for alien in alien_group:
            alien.speed = alien.speed + .01 * factor
```

Добавление подсчета очков

Когда мы играем в игру, мы хотели бы отслеживать определенную статистику для пользователя и отображать ее. В игре о космическом вторжении мы будем отслеживать Score, High Score и Lives. Мы уже создали Score и HiScore Sprites в нашей игре с пожирателем камней, и мы можем использовать те же классы в нашей игре о космическом вторжении. Жизни — единственный новый класс, показанный ниже. Этот класс покажет, сколько жизней осталось, в виде 3 значков кораблей, отображаемых в правом верхнем углу. Когда корабль уничтожается, один значок корабля удаляется до тех пор, пока не останется ни одного, после чего игра заканчивается.

```
import pygame

class LivesSprite(pygame.sprite.Sprite):
    def __init__(self, window_width):
        super().__init__()
        WHITE = (255, 255, 255)
        self.window_width = window_width
        self.lives = 3
        self.livesImage = pygame.image.load(
          'images/man.gif')
        self.livesImage =
         pygame.transform.scale(self.livesImage,
           (40, 32))
```

```
14          self.rect = pygame.Rect(0, 0, 0, 0)
15          self.small_font = pygame.font.Font(None, 32)
16          self.image = self.small_font.render(
17              f'Lives: {self.lives}', True, WHITE)
18      # Рисуем спрайт на экране
19
20      def update(self):
21          WHITE = (255, 255, 255)
22          self.image = self.small_font.render(
23              f'Lives:', True, WHITE)
24          self.rect = self.image.get_rect()
25              .move(self.window_width - 250, 0)
26
27      def draw(self, surface):
28          surface.blit(self.image, self.rect)
29          for i in range(self.lives):
30              surface.blit(self.livesImage,
31               (self.window_width - 180 + i * 50, 0))
32
33      def update_lives(self, lives):
34          self.lives = lives
```

Мы можем использовать то же изображение, что и для нашего игрока, как и для нашего индикатора жизней, нам просто нужно уменьшить его с помощью метода pygame.transform.scale.

Вот подробное объяснение кода:

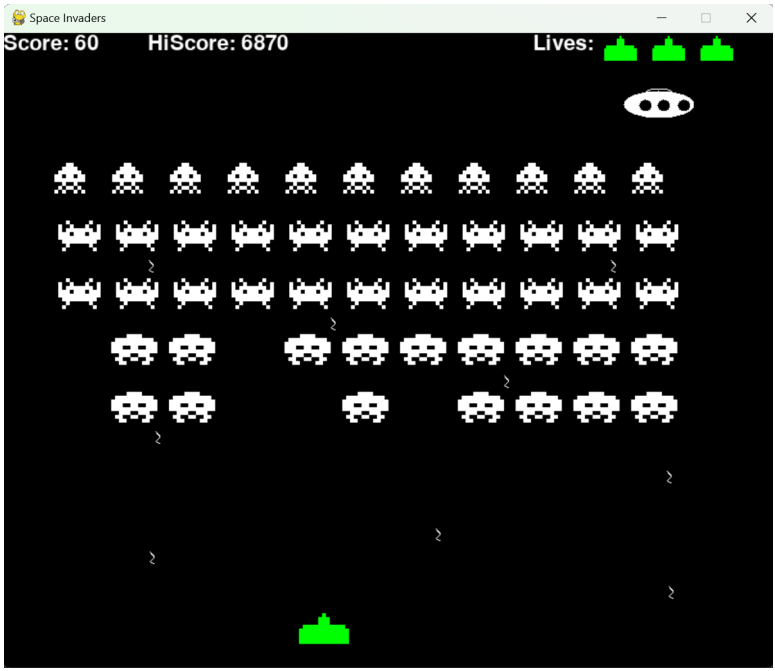
1. class LivesSprite(pygame.sprite.Sprite): определяет новый класс под названием LivesSprite, который наследуется от pygame.sprite.Sprite.
2. def init(self, window_width): определяет конструктор для класса LivesSprite. Он принимает ширину окна в качестве параметра, помогающего определить, где разместить индикатор в верхней части экрана.

3. super().init() инициализирует суперкласс (pygame.sprite.Sprite), чтобы убедиться, что класс LivesSprite наследует все необходимые атрибуты и методы.
4. WHITE = (255, 255, 255) определяет кортеж белого цвета, используемый для рендеринга текста.
5. self.window_width = window_width хранит ширину окна в переменной экземпляра.
6. self.lives = 3 инициализирует количество жизней равным 3.
7. self.livesImage = pygame.image.load('images/man.gif) загружает изображение для представления жизни игрока из файла 'images/man.gif'.
8. self.livesImage = pygame.transform.scale(self.livesImage, (40,32)) уменьшает изображение жизни до размера (40, 32).
9. self.rect = pygame.Rect(0, 0, 0, 0) инициализирует прямоугольную область для отображения текста жизней.
10. self.small_font = pygame.font.Font(None, 32) создает объект afont размером 32 для рендеринга текста Lives.
11. self.image = self.small_font.render(f' Lives: {self.lives}',True, WHITE) отображает исходный текст для отображения жизней.
12. def update(self): определяет метод обновления, отображаемого текста Lives.
13. def draw(self, surface): определяет метод рисования, который рисует текст отображения жизней и изображения жизней на экране. Параметр surface - это поверхность, на которой отображается отображение жизней.
14. def update_lives(self, lives): определяет метод update_lives, который обновляет количество жизней из основного цикла, когда игрок умирает. Параметр жизни - это новое количество жизней.
15. Таким образом, класс LivesSprite отвечает за отображение количества жизней игрока в игре Space Invaders. Он обрабатывает рендеринг как текста, так и живых изображений на игровом экране.

В методе handle_player_hit внутри нашей основной программы, если в игрока попала инопланетная бомба, мы уменьшаем количество жизней и обновляем LivesSprite, чтобы отразить потерянную жизнь:

```
handle_player_hit(bomb_x, bomb_y):
...
player_lives = player_lives - 1
lives_indicator.update_lives(player_lives)
```

Запуск НЛО



В классической аркадной игре Space Invaders тарелка, также известная как НЛО или корабль-загадка, появляется в верхней части экрана и перемещается по экрану горизонтально через равные промежутки времени.Тарелка обычно появляется каждые 25–30 секунд, но это может варьироваться в зависимости от конкретной версии игры или этапа, на котором находится игрок. Цель тарелки - дать игроку возможность заработать бонусные очки, сбивая ее, когда она перемещается с одной стороны экрана на другую. В нашей версии мы будем запускать тарелку каждые 20 секунд.

Чтобы запрограммировать это, мы начнем с создания SaucerSprite. Спрайт будет анимирован с использованием 3 изображений тарелок, которые создадут иллюзию вращения тарелки. Вот разбивка кода SaucerSprite:

Вот разбивка класса и его методов:

- **init**: метод-конструктор инициализирует спрайт тарелки тремя разными изображениями (name1, name2 и name3) для тарелки, начальной позицией (x, y) и необязательным параметром уровня. Он устанавливает начальный спрайт изображения, генерирует случайный счет для тарелки, вычисляет скорость на основе уровня и инициализирует другие соответствующие атрибуты.
- **reset**: этот метод сбрасывает тарелку на заданное положение и уровень, обновляя скорость, очки и другие атрибуты.
- **update**: этот метод сбрасывает тарелку на заданное положение и уровень, обновляя скорость, очки и другие атрибуты.
- **draw**: Этот метод рисует спрайт тарелки на заданной поверхности. Если тарелка мертва, он рисует счет тарелки вместо спрайта.
- **move_left**: этот метод перемещает спрайт тарелки влево наего значение скорости, при условии, что тарелка не мертва.
- switch_image: этот метод переключает спрайт изображения тарелки между тремя предоставленными изображениями на основе заданного номера изображения.
- **get_width** и **get_height**: эти методы возвращают ширину и высоту спрайта-блюдца соответственно.
- **kill**: этот метод устанавливает для атрибута dead блюдца значение True и записывает время, когда блюдце было убито, с помощью функции get_ticks() Pygame. Причина, по которой он записывает время смерти, заключается в том, чтобы дать спрайту время показать количество очков, набранных на экране после того, как тарелка умрет.

Вот полный код спрайта с тарелкой:

```
import random
import pygame
from ImageSprite import ImageSprite


class SaucerSprite(pygame.sprite.Sprite):
    def __init__(self, name1, name2, name3, x, y, level =\
 1):
        super().__init__()
        self.active = False
        self.imageSprite1 = ImageSprite(name1, x, y)
        self.imageSprite2 = ImageSprite(name2, x, y)
        self.imageSprite3 = ImageSprite(name3, x, y)
        self.imageSprite  = self.imageSprite1
        self.explosion = pygame.font.Font(None, 32)
        self.imageSprite = self.imageSprite1
        self.points =  random.randint(1, 6) * 50
        self.saucerScore = self.explosion.render(
            str(self.points), True, (255, 255, 255))
        self.speed = .05 * (.9 + level/10.0)
        self.position = (x, y)
        self.rect = self.imageSprite.image
            .get_rect().move(self.position)
        self.dead = False
        self.death_time = 0

    def reset(self, x, y, level = 1):
        self.imageSprite = self.imageSprite1
        self.points =  random.randint(1, 6) * 50
        self.saucerScore = self.explosion
            .render(str(self.points), True,
              (255, 255, 255))
        self.speed = .05 * (.9 + level/10.0)
        self.currentDirection = 'left'
        self.position = (x, y)
```

```
36          self.rect = self.imageSprite.image
37              .get_rect().move(self.position)
38          self.dead = False
39          self.death_time = 0
40
41      # обновить положение 3-х спрайтов,
42      # представляющих НЛО
43      def update(self):
44          self.rect = self.imageSprite.rect
45          if self.dead == True: return
46          self.imageSprite.rect = self.imageSprite.image
47              .get_rect().move(self.position)
48          self.imageSprite1.rect = self.imageSprite.rect
49          self.imageSprite2.rect = self.imageSprite.rect
50          self.imageSprite3.rect = self.imageSprite.rect
51          self.rect = self.imageSprite.rect
52
53      # Рисуем спрайт на экране
54
55      def draw(self, surface):
56          if self.dead == True:
57              surface.blit(self.saucerScore, self.rect)
58          else:
59              self.imageSprite.draw(surface)
60
61      def move_left(self):
62          if self.dead == True: return
63          (x, y) = self.position
64          self.position = (x - self.speed, y)
65
66      #переключение между 3 изображениями, представляющими тарелку
67      def switch_image(self, imageNumber):
68          if self.dead == True: return
69          if (imageNumber == 1):
70              self.imageSprite = self.imageSprite1
```

```
        elif (imageNumber == 2):
            self.imageSprite = self.imageSprite2
        else:
            self.imageSprite = self.imageSprite3


    def get_width(self):
        return self.imageSprite.get_width()

    def get_height(self):
        return self.imageSprite.get_height()

    def kill(self):
        self.dead = True
        self.death_time = pygame.time.get_ticks()
```

Как мы будем решать, когда запускать НЛО? В оригинальной игре тарелка перемещалась по экрану примерно каждые 25 секунд. Мы сделаем то же самое в нашей игре внутри игрового цикла:

```
#запускайте блюдце каждые 20 секунд, если игра еще не окончена.

    if game_over == False and game_time % 20000 == 0 and \
game_time > 0:
        start_saucer()
```

Функция start_saucer инициализирует нашу тарелку и подготавливает ее к перемещению по экрану. Она также воспроизводит особый звук блюдца:

```
def start_saucer():
    if saucer.active == False:
        saucer.reset(window_width - 50, 50, level)
        saucer.active = True
        saucer_sound.play()
        saucer.position = (window_width - 50, 50)
```

Когда тарелка инициализирована, нам нужно, чтобы она двигалась по экрану. Мы делаем это через функцию handle_saucer_movement в основном цикле.

```
def handle_saucer_movement():
    global game_time
    saucer_show_score_time = 1000

    if saucer.active:
        saucer.move_left()
        saucer.update()
        saucer.draw(window)
        saucer.switch_image(
         int(game_time/saucer_blink_speed) % 3)
        (saucer_x, saucer_y) = saucer.position
        if (saucer_x <= -100):
            saucer.dead = True
            saucer.position = (0, 0)

    if saucer.dead:
        current_saucer_death_time
         = pygame.time.get_ticks()
            - saucer.death_time
        if current_saucer_death_time >
           saucer_show_score_time:
            saucer.active = False
```

Эта функция отвечает за перемещение, обновление и рисование тарелки (или НЛО) из игры, подобной Space Invaders. Она также обрабатывает случай, когда тарелка «мертвая» (например, попала под снаряд).

Вот пошаговое объяснение кода:

1. **saucer_show_score_time = 1000**: эта строка устанавливает переменную, представляющую время (в миллисекундах), в течение которого на экране будет отображаться оценка тарелки после ее уничтожения.
2. **if saucer.active**: Это условие проверяет, является ли тарелка активной (видимой и движущейся на экране).
3. **saucer.move_left()**: если тарелка активна, этот метод перемещает тарелку влево.
4. **saucer.update()**:этот метод обновляет положение тарелки и ее спрайты изображения.
5. **saucer.draw(window)**: Тарелка рисуется на заданной поверхности (окно).
6. **saucer.switch_image(int(game_time/saucer_blink_speed)% 3)**: эта строка переключает спрайт изображения тарелки в зависимости от текущего игрового времени и переменной под названием saucer_blink_speed, создавая эффект анимации.
7. Следующие строки проверяют, не сдвинулась ли тарелка с экрана (влево). Если это так,тарелка считается мертвой, и её положение сбрасывается на (0, 0).
8. **if saucer.dead**: это условие проверяет, мертва ли тарелка.
9. **current_saucer_death_time = pygame.time.get_ticks()** - **saucer**.death_time: эта строка вычисляет время, прошедшее с момента убийства тарелки.

if current_saucer_death_time > saucer_show_score_time: это условие проверяет, превышает ли время, прошедшее с момента смерти блюдца, время, в течение которого должен отображаться счет блюдца. Если true, блюдце становится неактивным (saucer.active = False), то есть оно не будет отображаться или обновляться до тех пор, пока не будет сброшено и активировано снова.

Таким образом, функция handle_saucer_movement гарантирует, что блюдце перемещается по экрану, обновляет свое положение и переключает изображения, когда оно активно. Когда тарелка мертва, она показывает счет в течение установленного времени, прежде чем сделать ее неактивной.

Проверяем, попали ли мы в блюдце

Когда мы стреляем пулей из нашего игрока, пуля может попасть либо в инопланетянина, либо в блюдце, либо долететь до самого верха экрана. Нам нужно обработать случай, когда пуля попадает в блюдце, движущееся по экрану:

Внутри функции handle_bullet нам нужно проверить, не столкнулась ли пуля с тарелкой:

```
def handle_bullet(bullet, bullet_active):
    ...

    if (handle_saucer_hit(bullet_x, bullet_y)):
        bullet_active = False
        bullet.position = (0, 0)
```

Функция handle_saucer_hit проверяет столкновение тарелки с пулей. Если он сталкивается, функция вызывает функцию уничтожения SaucerSprite. Функция убийства переводит тарелку в мертвое состояние, что позволяет тарелке показывать счет в течение нескольких секунд после выстрела. Эта функция также воспроизводит звук, указывающий на удар по блюдцу, и добавляет баллы из бонусных очков блюдца.

```
def handle_saucer_hit(bullet_x, bullet_y):
    global player_score, bullet, saucer
    (x, y) = saucer.position
    # check if bullet collides with saucer
    if bullet_x > x
        and bullet_x < x + saucer.get_width()
        and bullet_y > y
        and bullet_y < y + saucer.get_height():
            saucer.kill()
            saucer_dying.play()
            player_score += saucer.points
        return True
    return False
```

Заключение

В этой главе мы углубились в разработку динамичного 2D-шутера от первого лица Space Invasion. Наше путешествие включало в себя основные игровые компоненты, такие как пользовательский ввод, звуковой дизайн, движение врагов, обнаружение столкновений и захватывающая анимация.

Хотя Space Invasion немного сложнее по сравнению с нашими предыдущими проектами, он служит отличным планом для создания собственных игровых проектов в реальном времени. Создавая уникальные спрайты, вы можете раскрыть свой творческий потенциал, комбинируя и адаптируя их к множеству игровых жанров и концепций. Это стремление не только расширяет ваш набор навыков, но и открывает целый мир возможностей для инноваций в игровой среде.

Приложение

Где найти изображения

В Open Game Art[2] можно выбрать бесплатные изображения из Itch.io[3], где есть как бесплатные, так и платные игровые активы.

Где найти звуки

На FreeSound.org[4] есть множество бесплатных звуков, которые вы можете загрузить для использования в своей игре.

[1] https://github.com/microgold/pygames/blob/master/ReadMe.md
[2] https://opengameart.org/
[3] https://itch.io/
[4] https://freesound.org/

Также на SoundBible.com[5] есть бесплатные звуки, которые вы можете просмотреть.

Другие источники

Python и Pygame могут быть сложными для изучения, но существует множество отличных ресурсов, которые помогут вам начать работу. Мы рекомендуем:

Для Python

Учебное пособие по Python — изучите программирование на Python (шаг за шагом)[6]

Учебное пособие по Python — полный курс для начинающих[7]

Python.org[8]: официальный веб-сайт языка программирования Python предоставляет множество информации для начинающих, включая учебные пособия, документацию и ресурсы сообщества.

Codecademy[9]: Codecademy предлагает бесплатный курс Python для начинающих.

Coursera[10]: Coursera предлагает бесплатные курсы Python от ведущих университетов и институтов.

Learn Python the Hard Way[11]: бесплатная онлайн-книга, предлагающая практический подход к изучению Python, основанный на упражнениях. []

Python для всех[12]: серия бесплатных онлайн-курсов, предлагаемых Мичиганским университетом, которые знакомят с программированием на Python.

---

[5] https://soundbible.com/
[6] https://www.youtube.com/watch?v=XGf2GcyHPhc
[7] https://www.youtube.com/watch?v=rfscVS0vtbw
[8] https://www.python.org/
[9] https://www.codecademy.com/learn/learn-python
[10] https://www.coursera.org/courses?query=python
[11] https://learnpythonthehardway.org/python3/
[12] https://www.py4e.com/lessons

W3Schools[13]: веб-сайт, который предлагает различные учебные пособия и ресурсы для изучения Python, а также других языков программирования и технологий веб-разработки.

More PyGames

Brick Breaker[14]

Sudoku[15]

Snake Game[16]

[13] https://www.w3schools.com/python/
[14] https://www.geeksforgeeks.org/brick-breaker-game-in-python-using-pygame/
[15] https://www.geeksforgeeks.org/building-and-visualizing-sudoku-game-using-pygame/
[16] https://www.edureka.co/blog/snake-game-with-pygame/