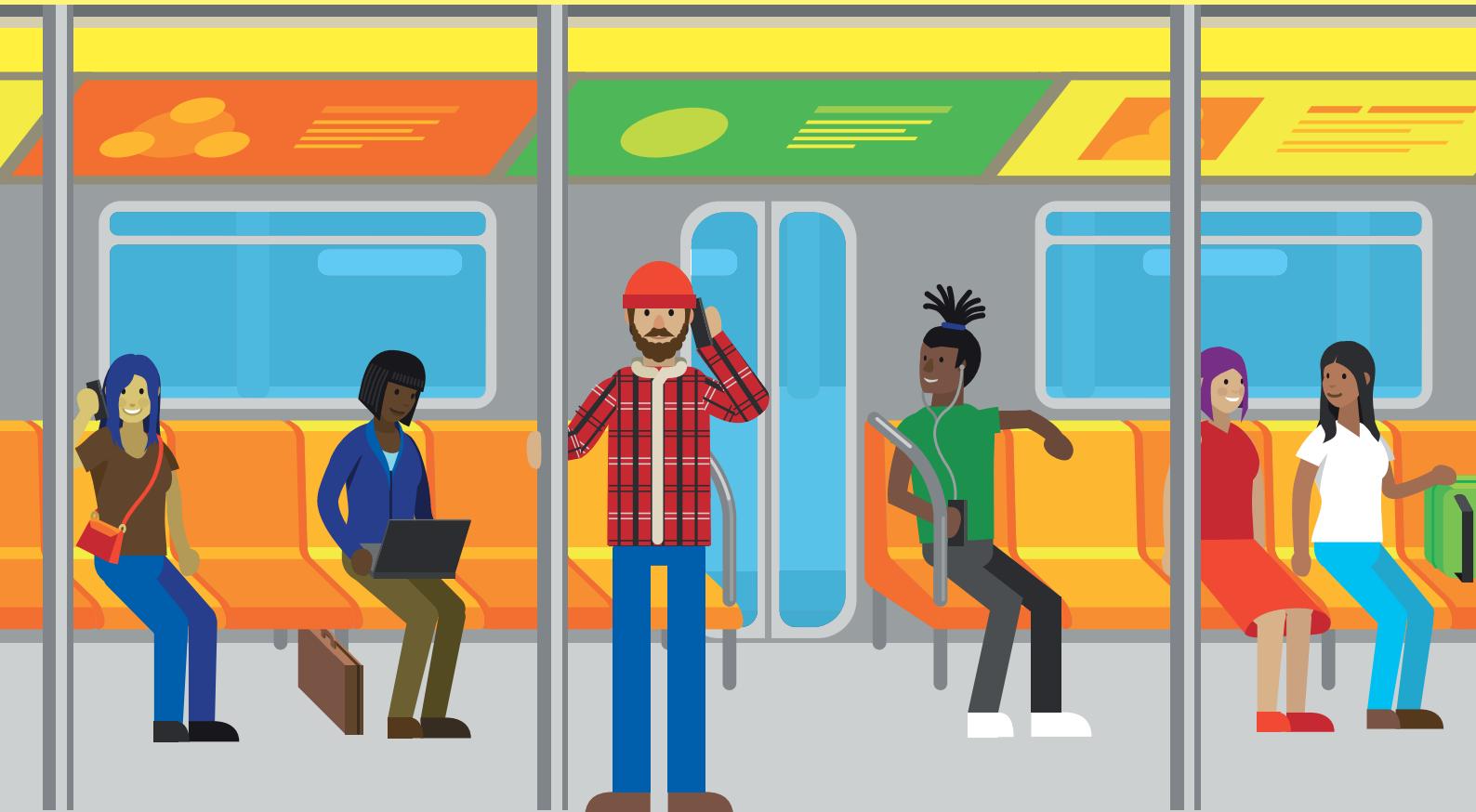


16.

Tworzenie gier za pomocą biblioteki pygame



Czego się nauczysz?

Pisanie gier to świetna zabawa. W przeciwieństwie do „właściwych” programów użytkowych, gry nie zawsze są powiązane z formalną specyfikacją i nie muszą robić niczego użytecznego. Po prostu muszą zapewniać fajną rozrywkę. Gry są doskonałym sposobem na eksperymentowanie z oprogramowaniem. Możesz pisać kod tylko po to, aby zobaczyć, co się dzieje, gdy działa, i przekonać się, czy wyniki są interesujące. Każdy programista powinien napisać w swoim życiu przynajmniej jedną grę komputerową. W tym rozdziale zaczniesz tworzyć gry. Dowiesz się, jak stworzyć kompletną grę, i poznasz framework, za pomocą którego będziesz mógł napisać większą liczbę gier według własnego projektu.

Wprowadzenie do biblioteki pygame	594
Rysowanie ilustracji za pomocą pygame	601
Pobieranie danych od użytkownika za pomocą pygame	606
Tworzenie postaci w grze	609
Dokończenie gry	629
Czego się nauczyłeś?	636

Wprowadzenie do biblioteki pygame

W tym podrozdziale zaczniemy korzystać z pygame. Stworzymy kilka figur i wyświetlimy je na ekranie. Bezpłatna biblioteka pygame zawiera wiele klas Pythona, które można wykorzystać do tworzenia gier. Funkcje modułu `snaps`, których używaliśmy w początkowych rozdziałach tej książki, zostały napisane przy użyciu frameworka pygame, więc prawdopodobnie posiadasz już tę bibliotekę na swoim komputerze (instrukcje pobierania znajdziesz w rozdziale 3.).

Warto zwrócić uwagę, że biblioteka pygame używa krotek do tworzenia pojedynczych elementów danych zawierających kolory i współrzędne identyfikujące elementy gier. Jeśli nie jesteś pewien, czym jest krotka, przed rozpoczęciem pracy opisanej w ramce „ZRÓB TO SAM” przeczytaj opis krotek w rozdziale 8.



ZRÓB TO SAM

Zimportuj pygame i narysuj kilka linii

Najlepszym sposobem, aby zrozumieć, jak działa pygame, jest zimportowanie jej i narysowanie czegoś. Na początek otwórz powłokę polecen Pythona w środowisku IDLE. Aby używać biblioteki pygame w programie, trzeba ją zimportować. Wprowadź poniższą instrukcję i naciśnij *Enter*:

```
>>> import pygame
```

Po zimportowaniu modułu pygame możemy zacząć korzystać z funkcji i klas, które on zawiera. Zanim będzie można użyć frameworka pygame do wyświetlania elementów w grze, trzeba go skonfigurować. W grze robi się to przez wywołanie funkcji `init` modułu pygame, tak jak pokazano poniżej:

```
>>> pygame.init()
```

Kiedy naciśniesz *Enter*, funkcja `init` ustawi różne elementy biblioteki pygame, z których każda wykonuje określone zadanie podczas działania gry. Biblioteka zawiera elementy do odczytu danych wprowadzanych przez użytkownika, odtwarzania dźwięków itd. Funkcja `init` zwraca krotkę, która informuje o tym, ile elementów zostało pomyślnie zainicjowanych, a ilu nie udało się zainicjować. Fakt nieprawidłowego zainicjowania elementu może wskazywać na to, że framework pygame nie został poprawnie zainstalowany. Jednak większość gier ignoruje tę wartość i zakłada, że wszystko jest w porządku.

```
>>> pygame.init()
(6, 0)
```

Powyższy wynik wskazuje, że sześć modułów zostało poprawnie skonfigurowanych. Nie ma żadnego, którego nie udałoby się zainicjować. Jeśliauważysz jakieś błędy — innymi słowy: jeśli druga wartość w krotce ma wartość różną od zera — sprawdź, czy framework pygame został poprawnie zainstalowany.

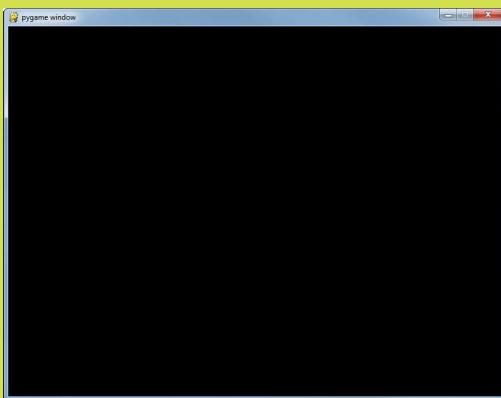
Następnie należy utworzyć obszar do rysowania. Obszar do rysowania ma określony rozmiar, ustawiany podczas jego tworzenia. Rozmiar jest podawany w pikselach (piksel to wielkość punktu na ekranie). Im więcej pikseli, tym lepsza jakość obrazu. W pikselach określa się również rozdzielcość ekranu kamery lub wideo. W naszym przykładzie będziemy używać ekranu o szerokości 800 pikseli i wysokości 600 pikseli. Do stworzenia obszaru rysowania możemy użyć krotki, tak jak pokazano poniżej:

```
>>> size = (800, 600)
```

Jak pamiętamy, krotka jest sposobem grupowania kilku elementów. Więcej informacji na ich temat znajdziesz w rozdziale 8. Kiedy już mamy krotkę opisującą rozmiar ekranu gry, możemy użyć tej wartości jako argumentu funkcji, która tworzy powierzchnię rysowania pygame.

```
>>> surface = pygame.display.set_mode(size)
```

Powyższa instrukcja tworzy obszar do rysowania, ustawia zmienną na referencję do niego, a następnie wyświetla go na ekranie. Powinieneś zobaczyć, że na ekranie wyświetliło się okno podobne do pokazanego poniżej.



Za pomocą poniższej instrukcji możesz zmienić tytuł okna do rysowni:

```
>>> pygame.display.set_caption('Niezwykła gra napisana przez Roba')
```

Powyższa funkcja zmienia tytuł okna, co można zobaczyć poniżej.



Teraz możemy rysować w wyświetlnym obszarze. Zaczniemy od narysowania kilku linii. Funkcja rysowania linii w pygame akceptuje cztery parametry:

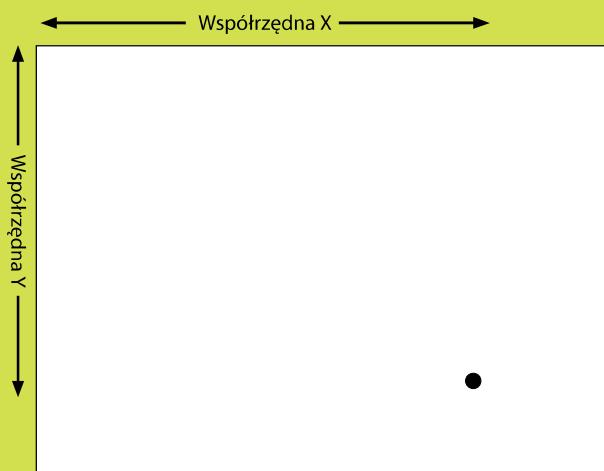
- powierzchnia do rysowania,
- pozycja początkowa linii,
- kolor rysowania,
- pozycja końcowa linii.

Dodajmy do siebie te elementy. Już utworzyliśmy powierzchnię, zatem możemy po prostu jej użyć. Kolor elementu w pygame jest wyrażony w postaci krotki zawierającej trzy wartości. Po raz pierwszy widzieliśmy ten mechanizm w rozdziale 3. Użyliśmy wtedy frameworka snaps do wykreślania tekstu. Każda wartość w krotce reprezentuje składową koloru odpowiednio czerwonego, zielonego i niebieskiego. Najniższa wartość wynosi 0, najwyższa to 255. Aby narysować czerwoną linię, możemy stworzyć krotkę, która zawiera pełną wartość koloru czerwonego i zerową dwóch pozostałych podstawowych kolorów. Wprowadź następującą krotkę:

```
>>> red = (255, 0, 0)
```

Teraz możemy ustawić pozycję początkową linii. Dla danej pozycji na ekranie wartość x określa, jak daleko ta pozycja znajduje się od lewej krawędzi, a wartość y określa odległość od górnej krawędzi w kierunku dołu ekranu. Określona lokalizacja jest wyrażana w postaci krotki zawierającej wartości współrzędnych (x , y). Działanie współrzędnych biblioteki pygame pokazano na poniższym rysunku. Ważną rzeczą do zapamiętania jest to, że początek układu — punkt o współrzędnych $(0, 0)$ — to lewy górny róg ekranu. Zwiększenie wartości x powoduje przesunięcie w prawo, a zwiększenie wartości y powoduje przesuwanie w dół ekranu.

Być może taki sposób działania grafiki nie jest intuicyjny. Większość wykresów, które rysujemy, ma początek układu współrzędnych w lewym dolnym rogu, a zwiększenie współrzędnej y przesuwa położenie w góre. Jednak umieszczenie początku układu w lewym górnym rogu jest standardową praktyką podczas rysowania grafiki na komputerze.



Mając to na uwadze, narysujmy linię od początku układu współrzędnych do pozycji (500, 300). Zaczniemy od stworzenia kilku krotek, które przechowują te wartości. Aby ustawić początkową i końcową pozycję linii, wpisz poniższe dwie instrukcje:

```
>>> start = (0, 0)
>>> end = (500, 300)
```

Teraz możemy wprowadzić instrukcję rysowania. Wpisz następujące wywołanie funkcji `line` z modułu `draw` frameworka `pygame`:

```
>>> pygame.draw.line(surface, red, start, end)
```

Kiedy wciśniesz `Enter`, zostanie narysowana linia, a funkcja `line`wróci obiekt `rect`, reprezentujący prostokąt, który otacza tę linię.

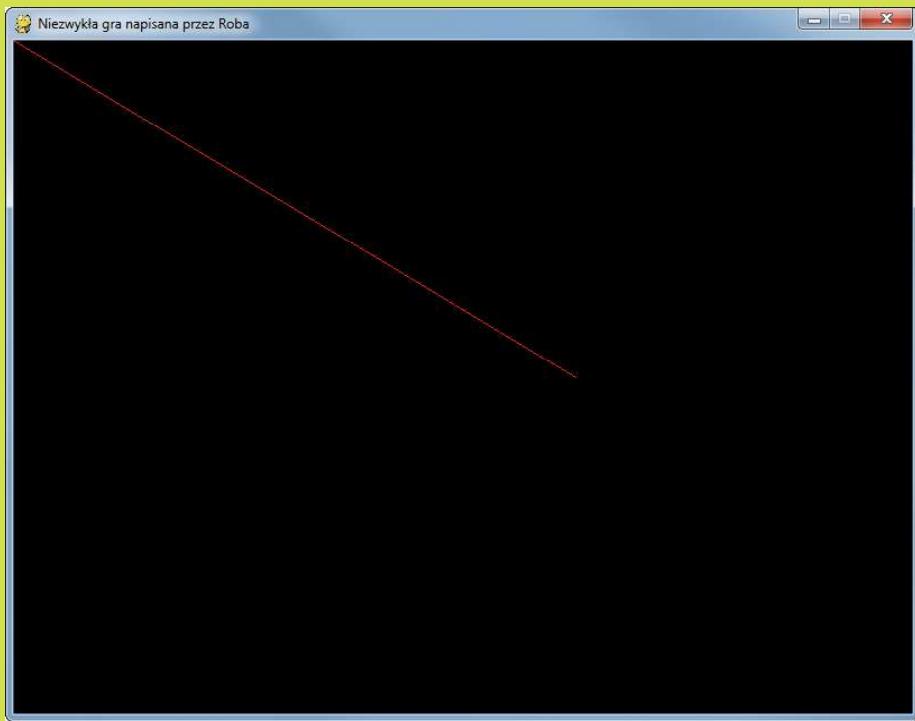
```
>>> pygame.draw.line(surface, red, start, end)
<rect(0, 0, 501, 301)>
```

Zignorujemy wartości zwrocone przez metody rysowania. Niestety, jeśli spojrzysz na okno gry, nie zobaczysz na ekranie żadnych linii. Operacje rysowania odbywają się w buforze „na zapleczu” zarządzanym przez `pygame`. Nie rysujemy bezpośrednio na ekranie, ponieważ nie chcemy, aby gracz widział każdą pojedynczą akcję rysowania. Zamiast tego wszystkie operacje rysowania wykonujemy w obszarze pamięci w komputerze (tzw. *back buffer*). Po zakończeniu rysowania kopujemy ten fragment pamięci do pamięci ekranu. Pamięć, która była wcześniej wyświetlana, staje się nową zawartością bufora *back buffer*, a proces rozpoczęty się od nowa.

W `pygame` do zamiany pamięci ekranu z zawartością *back buffer* służy funkcja `flip`. Aby linia wyświetliła się na ekranie, musimy wywołać funkcję `flip`, zatem wpisz poniższą instrukcję i naciśnij `Enter`.

```
>>> pygame.display.flip()
```

To wywołanie spowoduje wyświetlenie czerwonej linii na ekranie gry, tak jak pokazano na poniższym rysunku.



Jeśli nie chcesz mieć czarnego tła, możesz użyć funkcji `fill`, aby wypełnić ekran wybranym kolorem. Poniższe trzy instrukcje tworzą krotkę, która opisuje kolor biały, wypełnia *back buffer* tym kolorem, a następnie przełącza *back buffer*, aby wyświetlić biały ekran.

```
>>> white = (255, 255, 255)
>>> surface.fill(white)
>>> pygame.display.flip()
```

Jeśli to zrobisz, zauważysz, że utworzona przez nas czerwona linia została usunięta.

Z opisanych powyżej funkcji możemy skorzystać w celu stworzenia estetycznych obrazów. Poniższy program rysuje 100 kolorowych linii i 100 kolorowych kropek. Program wykorzystuje funkcje, które tworzą w obszarze wyświetlania losowe kolory i pozycje.

```
#EG 16-01 Funkcje rysowania pygame
import random
```

Demo korzysta z liczb losowych

```

import pygame                                         Program demo używa pygame

class DrawDemo:                                     Klasa zawierająca program demo

    @staticmethod                                         Ustaw tę metodę jako statyczną, ponieważ do jej użycia
    def do_draw_demo():                                nie ma potrzeby tworzenia obiektu klasy DrawDemo
        init_result = pygame.init()                      Metoda demonstrująca możliwości rysowania za pomocą pygame
        if init_result[1] != 0:                           Jeżeli liczba błędów jest różna od zera, to mamy problem
            print('biblioteka pygame nie została poprawnie zainstalowana')   Wyświetlenie komunikatu
            return                                         Anulowanie demonstracji

        width = 800                                      Ustawienie szerokości ekranu
        height = 600                                     Ustawienie wysokości ekranu
        size = (width, height)                          Ustawienie rozmiaru ekranu gry

        def get_random_coordinate():                   Funkcja pobierająca losową współrzędną
            X = random.randint(0,width-1)             Pobranie losowej wartości X
            Y = random.randint(0,height-1)            Pobranie losowej wartości Y
            return (X, Y)                            Zwrócenie krotki zawierającej współrzędne X i Y

        def get_random_color():                      Funkcja pobierająca losowy kolor
            red = random.randint(0,255)              Uzyskanie losowej wartości składowej czerwonej
            green = random.randint(0,255)            Uzyskanie losowej wartości składowej zielonej
            blue = random.randint(0,255)             Uzyskanie losowej wartości składowej niebieskiej
            return (red, green, blue)               Zwrócenie krotki zawierającej składowe czerwoną, zieloną i niebieską

            surface = pygame.display.set_mode(size)  Utworzenie powierzchni gry
            pygame.display.set_caption('Przykład rysunku')  Ustawienie podpisu okna

            red = (255, 0, 0)
            green = (0, 255, 0)
            blue = (0, 0, 255)
            black = (0, 0, 0)
            yellow = (255, 255, 0)
            magenta = (255, 0, 255)
            cyan = (0, 255, 255)
            white = (255, 255, 255)
            gray = (128, 128, 128)                  Utworzenie kilku krotek kolorów

            # wypełnienie ekranu białym tłem
            surface.fill(white)

            # narysowanie 100 losowych linii
            for count in range(100):

```

```

start = get_random_coordinate()
end = get_random_coordinate()
color = get_random_color()
pygame.draw.line(surface, color, start, end)

# narysowanie 100 kropek
dot_radius = 10
for count in range(100):
    pos = get_random_coordinate()
    color = get_random_color()
    radius = random.randint(5, 50)
    pygame.draw.circle(surface, color, pos, radius)

pygame.display.flip()

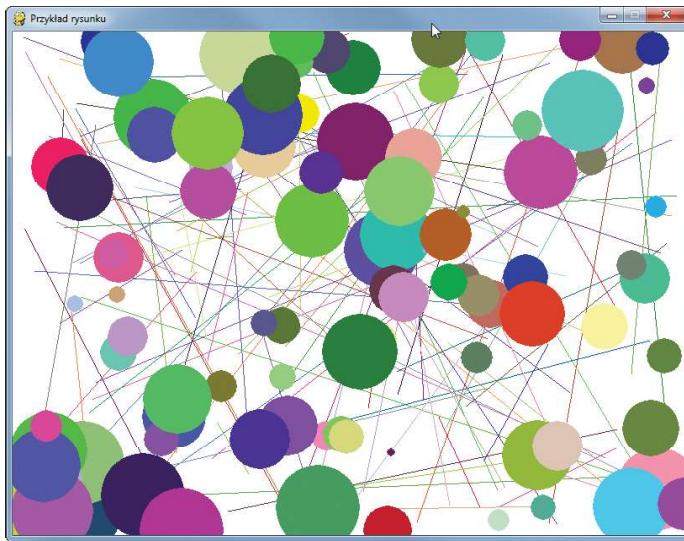
```

Przełączanie narysowanych elementów do pamięci ekranu

DrawDemo.do_draw_demo()

Wywołanie metody do_draw_demo klasy DrawDemo

Próba uruchomienia powyższego programu spowoduje wyświetlenie ekranu pokazanego na rysunku 16.1.



Rysunek 16.1. Rysowanie kropek i linii

Po uruchomieniu programu otrzymasz obraz, który wygląda podobnie, ale ma zupełnie inny układ linii i kropek. Dzieje się tak dlatego, ponieważ Twój program otrzyma zupełnie inną sekwencję liczb losowych niż te, które otrzymałem, gdy uruchomiłem program na swoim komputerze.



Tworzenie grafiki

Moglibyśmy utworzyć program, który co jakiś czas wyświetla inny wzorzec. Do określenia kolorów używanych we wzorcu moglibyśmy użyć pory dnia i bieżących warunków pogodowych. W ten sposób mógłby powstać obraz, który zmienia się w ciągu dnia (np. ma jasne kolory podstawowe rano, a bardziej łagodne i ciemniejsze kolory wieczorem). Gdyby było ciepło, kolory mogłyby mieć odcień czerwonego, a jeśli byłoby chłodniej, mogłyby mieć więcej niebieskiego. Należy zapamiętać, że można stworzyć dowolny kolor grafiki. Wystarczy wybrać odpowiednią ilość składowych czerwonej, zielonej i niebieskiej, które powinna zawierać.

Rysowanie ilustracji za pomocą pygame

Za pomocą biblioteki `pygame` można także rysować obrazy na ekranie. Obrazy mogą być ładowane z plików przechowywanych na Twoim komputerze. Wcześniej do rysowania obrazów używaliśmy funkcji `display_image` z biblioteki `snaps`. Teraz dowiemy się, jak do ładowania i wyświetlania obrazów używać framework'a `pygame`.

Typy plików graficznych

Istnieje wiele różnych formatów przechowywania obrazów na komputerach. Biblioteka `pygame` obsługuje ilustracje w jednym z dwóch następujących formatów:

- PNG — format PNG jest **bezstratny**, co oznacza, że zawsze przechowuje dokładną wersję obrazu. Pliki PNG mogą mieć również obszary przezroczyste, co jest ważne, gdy chcemy rysować jeden obraz na drugim.
- JPEG — format JPEG jest **stratny**, co oznacza, że obraz jest kompresowany, dzięki czemu zmniejsza się jego rozmiar, ale kosztem precyzyjnych szczegółów.

W tworzonych grach warto wykorzystywać obrazy w formacie JPEG do dużych obrazów tła oraz obrazy w formacie PNG do mniejszych obiektów narysowanych na pierwszym planie.

Jeśli nie masz własnych ilustracji, możesz wykorzystać te, które dostarczyłem w przykładowych plikach do tego rozdziału, ale będziesz mieć większą satysfakcję, jeśli użyjesz własnych obrazów.

Na rysunku 16.2 pokazano moje zdjęcie sera, które zastosujemy w przykładowej grze. Gracz będzie sterował serem i używał go do łapania krakersów poruszających się po ekranie. Jeśli chcesz, możesz użyć innego zdjęcia. Zdecydowanie zalecam, abyś to zrobił. Zapisałem obraz w formacie pliku PNG o szerokości 50 pikseli, który będzie odpowiedni dla używanego przez nas rozmiaru ekranu.



Rysunek 16.2. Ser

Aby dokonać konwersji obrazów na format PNG, możesz załadować obraz za pomocą programu Microsoft Paint, a następnie zapisać go w wybranym formacie. Używając Painta, możesz również skalować i przycinać obrazy — np. po to, by zmniejszyć liczbę pikseli w obrazie. Do wykonywania bardziej zaawansowanych operacji na ilustracjach polecam program *Paint.net*, dostępny do pobrania za darmo pod następującym adresem: www.getpaint.net. Kolejny świetny program do obróbki ilustracji to Gimp, dostępny dla większości maszyn. Można go pobrać ze strony www.gimp.org.

Ładowanie ilustracji do gry

Biblioteka `pygame` zawiera funkcję o nazwie `load`, która pozwala załadować ilustrację z pliku. Obraz do załadowania jest identyfikowany przez nazwę pliku. Funkcja `load` poszukuje pliku w lokalnym folderze. Innymi słowy: przeszukuje folder, z którego uruchomiono program. Widzieliśmy to zachowanie w rozdziale 8., kiedy pisaliśmy programy do przechowywania i ładowania danych z wykorzystaniem plików. Poniższa instrukcja ładuje obraz z pliku. Zmienna `cheeseImage` jest ustawiona tak, aby odwoływała się do ładowanego obrazu.

```
cheeseImage = pygame.image.load('cheese.png')
```

Teraz, gdy załadowaliśmy obraz, możemy narysować go na ekranie. W momencie rysowania obrazu dane, które go opisują, są kopowane do pamięci używanej przez ekran. Twórcy gier

nazywają to **blittingiem** danych graficznych na ekran. Biblioteka `pygame` zawiera funkcję o nazwie `blit`, która służy do kopiowania obrazu do pamięci ekranu. Metoda `blit` wymaga do działania dwóch informacji:

- obrazu do narysowania,
- współrzędnych na ekranie, gdzie ma wyświetlić się obraz.

Spróbujmy umieścić obraz sera w lewym górnym rogu ekranu. Krotkę opisującą tę pozycję tworzy poniższa instrukcja. Obie współrzędne `x` i `y` mają wartości zerowe.

```
cheesePos = (0,0)
```

Możemy teraz wywołać metodę `blit`, aby narysować ser. Do metody `blit` przekazujemy obszar wyświetlania, który stworzyliśmy podczas uruchamiania naszego programu gry.

```
surface.blit(cheeseImage, cheesePos)
```

Kompletny program, który rysuje ser na ekranie, zamieszczono poniżej:

```
# EG16-02 Rysowanie obrazów

import pygame

class ImageDemo:

    @staticmethod
    def do_image_demo():
        init_result = pygame.init() Inicjalizacja biblioteki pygame
        if init_result[1] != 0:
            print('biblioteka pygame nie została poprawnie zainstalowana') Zakończenie metody, jeśli uruchomienie pygame się nie powiodło
            return

        width = 800
        height = 600
        size = (width, height) Ustawienie rozmiarów ekranu

        surface = pygame.display.set_mode(size) Pobranie obszaru rysowania framework pygame
        pygame.display.set_caption('Przykład obrazu') Konfiguracja ekranu pygame

        white = (255, 255, 255)
        surface.fill(white) Wypełnienie ekranu białym tłem
```

```

cheeseImage = pygame.image.load('Cheese.png')           Załadowanie obrazu sera
cheesePos = (0,0)                                     Ustawienie pozycji sera w lewym górnym rogu ekranu
surface.blit(cheeseImage, cheesePos)                  Narysowanie sera
pygame.display.flip()                                Przełączenie pamięci ekranu, aby wyświetlić obraz sera

ImageDemo.do_image_demo()

```

Gdy uruchomimy ten program, narysuje on na ekranie kawałek sera, tak jak pokazano na rysunku 16.3. Zwróćmy uwagę, że po wyświetleniu obrazu znajduje się on w lewym górnym rogu okna wyświetlanego.



Rysunek 16.3. Ilustracja sera na ekranie

Ruchome obrazy

Do funkcji `blit` przekazujemy pozycję rysowania dla obrazu. Jeśli kilkakrotnie narysujemy obraz w różnych pozycjach, możemy stworzyć wrażenie, że obraz się porusza.

```

# EG16-03 Ruchomy ser

cheeseX = 40
cheeseY = 60                                     Ustawienie początkowej pozycji obrazu sera

clock = pygame.time.Clock()                      Utworzenie egzemplarza klasy clock z biblioteki pygame

for i in range(1,100):
    clock.tick(30)                             Przenieść ser 100 razy
    surface.fill((255,255,255))                Zatrzymanie gry tak, aby uzyskać 30 klatek na sekundę
    cheeseX = cheeseX + 1                      Wypełnienie ekranu białym tłem
    cheeseY = cheeseY + 1                      Zwiększenie pozycji X obrazu sera
    cheesePos = (cheeseX,cheeseY)               Zwiększenie współrzędnej Y pozycji sera
    surface.blit(cheeseImage, cheesePos)        Utworzenie krotki opisującej pozycję obrazu sera
    pygame.display.flip()                      Blitting obrazu sera na ekran
                                                Przełączenie bufora back buffer w celu aktualizacji ekranu

```



Ruchome obrazy

Na przykładzie programu *EG16-03 Ruchomy ser* możemy przeanalizować, w jaki sposób w grach uzyskujemy wrażenie, że obiekty się poruszają. Kiedy użyjesz środowiska IDLE do uruchomienia tego programu, powinieneś zauważyc, że ser przez chwilę majestatycznie przesuwa się w dół ekranu, a następnie się zatrzymuje. O szybkości ruchu decyduje parametr liczby wyświetlanych klatek gry na sekundę. Szybkość klatek to tempo, w jakim następuje przerysowywanie ekranu, wyrażone za pomocą liczby klatek na sekundę (ang. *frame per second* — fps). Klasa `Clock` z biblioteki pygame zapewnia metodę `tick`, do której należy przekazać liczbę klatek na sekundę wymaganą do działania gry. Program tworzy nowy obiekt `Clock` przed rozpoczęciem ruchu sera.

```
clock = pygame.time.Clock()
```

Klasa `Clock` udostępnia zbiór metod zarządzania czasem, które można wykorzystywać w grach. W tym przykładzie użyjemy metody `tick`, dzięki której gra będzie działać w stałym tempie. Bez obiektu `clock` gra działałaby tak szybko, jak Python może uruchomić program. Gra w takim tempie jest niemożliwa.

```
clock().tick(30)
```

Metoda `tick` spowoduje zatrzymanie gry do momentu rozpoczęcia następnego przedziału czasowego klatki. Znajdź powyższą instrukcję w programie i zmień wartość z 30 na 60. Teraz program będzie aktualizował ekran 60 razy na sekundę. Uruchom program, a przekonasz się, że ser porusza się dwa razy szybciej niż poprzednio, ponieważ metoda `tick` pozwala teraz na wyświetlanie 60 klatek na sekundę.

Jeśli zmienisz liczbę klatek na 5 (5 klatek na sekundę), zobaczysz, że ser porusza się powoli, i będziesz mógł śledzić każdy jego ruch.

Gracz uzyska dobry komfort gry, jeśli ekran będzie się aktualizował w tempie 60 klatek na sekundę. Gry na mniejszych urządzeniach — np. telefonach komórkowych czy tabletach — mogą wykorzystywać mniejsze wartości liczby klatek na sekundę, aby oszczędzać baterie.

Pobieranie danych od użytkownika za pomocą pygame

Teraz, gdy potrafimy poruszać obiektami kontrolowanymi przez program po ekranie, potrzebujemy sposobu interakcji gracza z grą. Gra otrzymuje dane wejściowe od użytkownika za pomocą zdarzeń biblioteki `pygame`. Zdarzenie to działanie użytkownika, np. naciśnięcie klawisza na klawiaturze lub poruszenie myszą. Po raz pierwszy zetknęliśmy się z tego typu zdarzeniami, gdy tworzyliśmy graficzny interfejs użytkownika przy użyciu biblioteki `Tkinter` w rozdziale 13. Aby odczytywać zdarzenia w bibliotece `Tkinter`, wiązaliśmy metodę ze zdarzeniem. Gdy zachodziło zdarzenie, była wywoływana metoda.

W bibliotece `pygame` zdarzenia są zarządzane inaczej. Podczas działania programu bazującego na `pygame` system `pygame` przechwytuje zdarzenia wejścia i umieszcza je w kolejce. Program gry musi regularnie sprawdzać kolejkę zdarzeń, aby się dowiedzieć, czy są jakieś zdarzenia, na które program powinien zareagować. Nas interesują zdarzenia klawiatury generowane po naciśnięciu lub zwolnieniu klawisza.



ZRÓB TO SAM

Zdarzenia w pygame

Aby przyjrzeć się, jak działają zdarzenia w `pygame`, wygenerujemy kilka zdarzeń i zobaczymy ich działanie. Aby utworzyć okno `pygame`, uruchom powłokę poleceń Pythona w środowisku IDLE i wpisz następujące instrukcje:

```
>>> import pygame  
>>> pygame.init()  
(6, 0)  
>>> size = (800, 600)  
>>> surface = pygame.display.set_mode(size)
```

Teraz kliknij myszą w oknie, które otworzyła `pygame`, i naciśnij kilka klawiszy. Każde naciśnięcie klawisza wygeneruje zdarzenie, które zostanie przechwycone przez `pygame`. Możemy teraz utworzyć pętlę, aby przyjrzeć się zdarzeniom, które zostały zapisane. Wróć do IDLE i wpisz następujący kod:

```
>>> for e in pygame.event.get():  
    print(e)
```

Metoda `get` zwraca kolekcję zdarzeń. Ta pętla wyświetli wszystkie zdarzenia w kolejce zdarzeń `pygame`. Po wprowadzeniu pustego wiersza za polecienniem `print` zobaczysz informacje o wszystkich zdarzeniach:

```

>>> for e in pygame.event.get():
print(e)
<Event(17-VideoExpose {})>
<Event(16-VideoResize {'size': (800, 600), 'w': 800, 'h': 600})>
<Event(1-ActiveEvent {'gain': 0, 'state': 1})>
<Event(2-KeyDown {'unicode': 'r', 'key': 114, 'mod': 0, 'scancode': 19})>
<Event(3-KeyUp {'key': 114, 'mod': 0, 'scancode': 19})>
<Event(2-KeyDown {'unicode': 'o', 'key': 111, 'mod': 0, 'scancode': 24})>
<Event(3-KeyUp {'key': 111, 'mod': 0, 'scancode': 24})>
<Event(2-KeyDown {'unicode': 'b', 'key': 98, 'mod': 0, 'scancode': 48})>
<Event(3-KeyUp {'key': 98, 'mod': 0, 'scancode': 48})>
<Event(1-ActiveEvent {'gain': 1, 'state': 1})>
>>>

```

Każde zdarzenie jest opisane przez słownik, w którym są zapisane informacje o zdarzeniu. Jeśli przejrzysz powyższe zdarzenia, zobaczyś, że użytkownik po kolej naciskał i zwalniał klawisze R, O i B.

Podczas działania gry należy sprawdzać kolejkę zdarzeń, aby się dowiedzieć, czy wprowadzono jakieś polecenia, które powinny spowodować przesuwanie obiektów na ekranie. Chcemy, aby ser poruszał się, gdy jest wcisnięty klawisz strzałki, i przestawał się poruszać po zwołnieniu tego klawisza. Za to zachowanie jest odpowiedzialny poniższy kod. Ponadto ten kod zawiera test, który powoduje zakończenie gry, gdy gracz naciśnie klawisz *Esc*.

# EG16-04 Sterowany ser	
cheeseX = 40	Ustawienie początkowej pozycji sera
cheeseY = 60	Ustawienie szybkości ruchu sera
cheeseYSpeed = 2	Ser nie porusza się w góre
cheeseMovingUp = False	Ser nie porusza się w dół
cheeseMovingDown = False	Utworzenie obiektu zegara
clock = pygame.time.Clock()	Wielokrotnie powtarzaj pętlę gry
while True:	Zaczekaj na rozpoczęcie następnej ramki
clock().tick(60)	Przetwarzanie zdarzeń
for e in pygame.event.get():	Czy zdarzenie dotyczy wcisnięcia klawisza?
if e.type == pygame.KEYDOWN:	Czy wcisnięto klawisz Escape?
if e.key == pygame.K_ESCAPE:	Zamknięcie pygame
pygame.quit()	Jeśli wcisnięto Escape, to pętla gry się kończy
return	Czy wcisnięto klawisz strzałki w góre?
elif e.key == pygame.K_UP:	Ustawienie flagi, która wskazuje, że ser porusza się w góre
cheeseMovingUp = True	

elif e.key == pygame.K_DOWN:	Czy wciśnięto klawisz strzałki w dół?
cheeseMovingDown = True	Ustawienie flagi, która wskazuje, że ser porusza się w dół
elif e.type == pygame.KEYUP:	Czy zdarzenie dotyczy zwolnienia klawisza?
if e.key == pygame.K_UP:	Czy zwolniono klawisz strzałki w górę?
cheeseMovingUp = False	Wyczyszczenie flagi wskazującej, że ser porusza się w góre
elif e.key == pygame.K_DOWN:	Czy zwolniono klawisz strzałki w dół?
cheeseMovingDown = False	Wyczyszczenie flagi wskazującej, że ser porusza się w dół
if cheeseMovingDown:	Czy ser porusza się w dół?
cheeseY = cheeseY+cheeseYSpeed	Przesuwanie sera w dół ekranu
if cheeseMovingUp:	Czy ser porusza się w góre?
cheeseY = cheeseY-cheeseYSpeed	Przesuwanie sera w góre ekranu
	Wyczyszczenie flagi wskazującej, że ser porusza się w dół



ANALIZA KODU

Pętle gry

Powyższy kod jest przykładem „pętli gry”. Możesz mieć kilka pytań na ten temat.

Pytanie: Do czego służy używana w programie zmienna e?

Odpowiedź: Zmienna e zawiera zdarzenie sprawdzane w pętli gry. Dla gry interesujące są tylko zdarzenia generowane po naciśnięciu lub zwolnieniu klawiszy. Gdy program wykryje naciśnięcie klawisza, sprawdza, który klawisz został naciśnięty. Jeśli to jest strzałka w góre, kod ustawia flagę, która wskazuje, że ser powinien poruszać się w góre; jeśli to strzałka w dół, kod ustawia flagę wskazującą na to, że ser powinien poruszać się w dół. Pętla gry zawiera także testy, które wyczyszczają flagę, jeśli klawisz zostanie zwolniony.

Pytanie: Dlaczego ser porusza się, gdy przytrzymuję wciśnięty klawisz?

Odpowiedź: Należy wziąć pod uwagę, że instrukcje w pętli gry są powtarzane 60 razy na sekundę. Tak więc program aktualizuje pozycję sera co jedną sześćdziesiątą sekundy. Jeśli klawisz jest wciśnięty, ruch sera nastąpi przy każdym przebiegu pętli gry. Obecnie wartość zmiennej cheeseYSpeed wynosi 2, co oznacza, że w ciągu sekundy ser przesunie się o 120 pikseli.

Pytanie: Jak zmienić prędkość ruchu sera?

Odpowiedź: Zmienna cheeseYSpeed określa prędkość sera w kierunku y (w góre i w dół ekranu). Aby ser poruszał się szybciej, można zwiększyć wartość tej zmiennej.

Pytanie: Dlaczego, aby przesunąć ser w dół ekranu, zwiększamy wartość y?

Odpowiedź: Jest tak, ponieważ w układzie współrzędnych używanym przez pygame początek układu (punkt, w którym wartości współrzędnych x i y są równe zero) znajduje się u góry ekranu. Zwiększenie wartości współrzędnej y powoduje przesunięcie sera w dół ekranu.

Pytanie: Co by się stało, gdyby gracz jednocześnie wcisnął klawisze strzałki w góre i strzałki w dół?

Odpowiedź: Po aktualizowaniu ser poruszałby się na przemian najpierw w górę, a następnie w dół. W rezultacie sprawiałby wrażenie, jakby wcale się nie poruszał.

Pytanie: Co by się stało, gdyby gracz przesunął ser poza ekran?

Odpowiedź: Możesz uruchomić przykładowy program, aby się dowiedzieć, co się stanie. Rysowanie obrazu poza ekranem nie spowoduje awarii programu, ale obiekt nie będzie widoczny. Aby nie dopuścić do wyjścia sera poza ekran, trzeba dodać kod, który zadba o to, by ser nigdy nie znajdował się poza ekranem.

Pytanie: Do czego służy metoda `pygame.quit()`?

Odpowiedź: Metoda `pygame.quit()` jest wywoływana, gdy użytkownik naciśnie klawisz `Escape`, aby zakończyć grę. Zamiera pygame i powoduje zamknięcie okna gry.

Tworzenie postaci w grze

W grze, którą utworzymy, będą wyświetlane trzy różne typy obiektów:

- **Ser** — gracz będzie sterował serem po ekranie.
- **Krakersy** — gracz spróbuje schwytać krakersa za pomocą kawałka sera.
- **Zabójczy pomidor** — pomidor będzie ścigał ser.

Każdy z tych obiektów ekranowych jest egzemplarzem klasy `Sprite`. Można go traktować jak obraz, który jest częścią ekranu gry. Klasa `Sprite` zawiera obraz narysowany na ekranie, informacje o pozycji na ekranie i implementację zbioru zachowań. Każdy obiekt ekranowy w grze będzie realizować następujące działania:

- Wyświetlać się na ekranie.
- Aktualizować się. Jeśli obiekt reprezentuje ser, porusza się w odpowiedzi na klawisze wciskane przez gracza. Jeśli jest zabójczym pomidorem, ściga ser.
- Resetować się. Kiedy rozpoczynamy nową grę, musimy zainicjować obiekt ekranowy w położeniu początkowym.

Postacie gry mogą mieć również inne zachowania, ale te, które wymieniliśmy, są podstawowymi operacjami, jakie muszą wykonywać. Te zachowania możemy umieścić w klasie:

```
class Sprite:  
    """  
        Postać w grze. Pozwala na tworzenie klas potomnych,  
        aby tworzyć postacie o określonych zachowaniach  
    """  
  
    def __init__(self, image, game):  
        """  
            Inicjalizacja postaci  
            image reprezentuje obraz wymagany do narysowania postaci  
            Domyślna pozycja na ekranie to początek układu współrzędnych w punkcie (0,0)  
            game reprezentuje obiekt gry, w której występuje ta postać  
        """  
  
        self.image = image  
        self.position = [0, 0]  
        self.game = game  
        self.reset()  
  
    def update(self):  
        """  
            Metoda wywoływana w pętli gry w celu aktualizacji  
            statusu postaci  
            W klasie bazowej nie robi nic  
        """  
  
        pass  
  
    def draw():  
        """  
            Rysuje postać na ekranie  
            w jej bieżącej pozycji  
        """  
  
        self.game.surface.blit(self.image, self.position)  
  
    def reset(self):  
        """  
            Metoda wywoływana na początku gry w celu  
            zresetowania postaci  
        """  
  
        pass
```

To jest klasa bazowa dla wszystkich postaci w grze

Metoda wywoływana w celu skonfigurowania atrybutów postaci

Zapisanie obrazu postaci

Ustawienie pozycji postaci w lewym górnym rogu

Zapisanie referencji do gry w obiekcie postaci

Zresetowanie postaci

Metoda wywoływana, gdy postać gry ma być zaktualizowana

Metoda wywoływana w celu zażądania od postaci narysowania się

Metoda wywoływana w celu zażądania od postaci zresetowania się



Klasa bazowa Sprite

Powyższy kod definiuje klasę bazową dla wszystkich postaci w grze. Możesz mieć kilka pytań na ten temat.

Pytanie: Do czego służy parametr game w metodzie inicjującej?

Odpowiedź: Kiedy gra tworzy nową postać, musi poinformować obiekt o grze, której ta postać jest częścią, ponieważ niektóre postacie będą musiały użyć informacji przechowywanych w obiekcie gry. Na przykład jeśli serowi uda się złapać krakersa, trzeba będzie zaktualizować wartość wyniku.

Programiści powiedzieliby, że klasy reprezentujące grę i postać są ze sobą ściśle powiązane. Zmiany w kodzie klasy CrackerChaseGame mogą mieć wpływ na zachowanie postaci w grze. Jeśli programista klasy CrackerChaseGame zmieni nazwę zmiennej, która przechowuje wynik, ze score na game_score, to próba wykonania metody Update w klasie Cheese po schwytniu krakersa zawiedzie. Wiele ścisłych sprzężeń pomiędzy klasami w dużym systemie nie jest dobrym pomysłem, ale w przypadku naszej gry znacznie ułatwia programowanie, więc myślę, że rozsądne jest, aby program działał w ten sposób.

Pytanie: Dlaczego metody aktualizacji i resetowania są puste?

Odpowiedź: Klasę Sprite można porównać do szablonu dla klas potomnych. Niektóre elementy gry wymagają metod zaimplementowania zachowań aktualizacji i resetowania. Ser będzie potrzebował metody resetowania, która na początku gry umieści obiekt reprezentujący ser na środku ekranu. Będzie też potrzebował metody aktualizacji, odpowiedzialnej za poruszanie nim po ekranie. Klasa reprezentująca ser będzie podklassą klasy Sprite, w której znajdą się własne wersje wymienionych metod.

Pytanie: Jak działa metoda draw?

Odpowiedź: Metoda draw jest wywoływana w celu zażądania od postaci gry narysowania siebie na ekranie.

```
def draw(self):
    """
    Rysuje obraz postaci na ekranie w jej
    bieżącym położeniu
    """
    self.game.surface.blit(self.image, self.position)
```

Obiekt gry, której częścią jest postać, zawiera atrybut o nazwie surface, który reprezentuje powierzchnię do rysowania dla tej gry. Powyższa metoda znajduje atrybut game w obiekcie postaci, która się rysuje. Atrybut game został ustawiony podczas tworzenia postaci. Wykorzystuje on właściwość surface obiektu game w celu wykonania blittingu obrazu postaci na ekranie.

Klasa `Sprite` nie robi zbyt wiele, ale można z niej skorzystać do zarządzania obrazem tła dla gry. Gra będzie się odbywać na tle „obrusa”. Tło możemy porównać do bardzo dużej postaci, która wypełnia ekran. Możemy teraz stworzyć pierwszą wersję gry zawierającą pętlę, która wyświetla samo tło.

```
class CrackerChase: Klasa zawierająca całą grę
    ...
    Odtwarza niesamowitą grę o pościgach za krakersami
    ...

    def play_game(self): Metoda wywoływana w celu uruchomienia gry
        ...
        Rozpoczęcie gry
        Zwraca sterowanie, gdy gracz wyjdzie
        z gry
        ...
        init_result = pygame.init() Inicjalizacja biblioteki pygame
        if init_result[1] != 0:
            print('biblioteka pygame nie została poprawnie zainstalowana')
            return Zakończenie programu, jeśli na tym komputerze nie zainstalowano biblioteki pygame
        self.width = 800
        self.height = 600 Ustawienie szerokości i wysokości ekranu gry
        self.size = (self.width, self.height) Utworzenie krotki definiującej rozmiar ekranu
        self.surface = pygame.display.set_mode(self.size) Utworzenie powierzchni do rysowania
        pygame.display.set_caption('Pościg za krakersami') Ustawienie podpisu ekranu gry
        background_image = pygame.image.load('background.png')
        self.background_sprite = Sprite(image=background_image,
                                         game=self) Informacja dla obiektu Sprite, do której gry należy
        clock = pygame.time.Clock()
        while True: Utworzenie obiektu zegara
            clock.tick(60) Pętla gry, która działa w nieskończoność
            for e in pygame.event.get(): Ustawienie odświeżania na 60 razy na sekundę
                if e.type == pygame.KEYDOWN: Pobranie zdarzeń z pygame
                    if e.key == pygame.K_ESCAPE: Czy wciśnięto klawisz?
                        pygame.quit() Zamknięcie ekranu gry
                        return Powrót z metody gry
                self.background_sprite.draw() Zażądanie od obiektu reprezentującego tło narysowania się
                pygame.display.flip() Przełączenie bufora back buffer na pierwszy plan
Jeśli wciśnięto klawisz Escape, zakończ grę
Utworzenie obiektu reprezentującego tło
Załadowanie obrazu tła
```



Klasa Game

Powyższy kod definiuje klasę, która będzie implementować naszą grę. Możesz mieć kilka pytań na ten temat.

Pytanie: W jaki sposób gra przekazuje referencję do siebie w konstruktorze postaci?

Odpowiedź: Wiemy, że gdy w klasie wywoływana jest metoda, używa się parametru `self` zawierającego referencję do obiektu, w którym działa metoda. Możemy przekazać parametr `self` do innych części gry, które tego potrzebują:

```
self.background_sprite = Sprite(image=background_image, game=self)
```

Powyższy kod tworzy nowy egzemplarz klasy `Sprite` i ustawia wartość argumentu `game` na `self`. Dzięki temu obiekt reprezentujący postać wie, jakiej gry jest częścią.

Pytanie: Dlaczego w celu narysowania postaci w grze wywoywana jest metoda `draw` obiektu reprezentującego postać? Czy gra nie może po prostu narysować obrazu przechowywanego wewnątrz obiektu postaci?

Odpowiedź: To bardzo ważne pytanie. Sprowadza się do odpowiedzialności obiektów za realizację określonych zadań. Czy postać gry powinna być odpowiedzialna za rysowanie na ekranie, czy też to obiekt gry powinien wykonywać tę operację? Myślę, że rysowanie powinno być zadaniem postaci, ponieważ w ten sposób programista uzyskuje dużo większą elastyczność.

Na przykład dodanie śladów dymu do niektórych postaci w grze poprzez rysowanie ilustracji „dymu” za postacią byłoby o wiele łatwiejsze, gdyby można było po prostu dodać kod do postaci reprezentujących dym, a nie do gry, która musiałaby ustalić, jakie postacie potrzebują śladów dymu, i narysować je za każdym razem inaczej, w zależności od postaci.

Pytanie: Czy to oznacza, że po uruchomieniu gry za każdym razem będzie przerysowywany cały ekran, nawet jeśli nic się na nim nie zmieniło?

Odpowiedź: Tak. Można by pomyśleć, że to marnowanie mocy komputera, ale tak działa większość gier. O wiele łatwiej jest narysować wszystko od zera, niż śledzić zmiany na ekranie, by przerysowywać tylko te fragmenty, które uległy zmianie.

Oto kod odpowiedzialny za uruchomienie gry:

```
# EG16-05 Rysowanie tła  
  
game = CrackerChase() Utworzenie egzemplarza obiektu gry  
game.play_game() Uruchomienie gry
```

Dodanie postaci gracza

Postać gracza będzie reprezentowana jako kawałek sera sterowany przez użytkownika po ekranie. Widzieliśmy, jak gra może reagować na zdarzenia klawiatury. Teraz stworzymy postać gracza i wprowadzimy w grze kod odpowiedzialny za kierowanie nią. Obiekt gracza w naszej grze implementuje poniższa klasa `Cheese`.

```
class Cheese(Sprite):  
    '''  
    Obiekt sera sterowany przez gracza  
    '''  
  
    def reset(self): Przesłonięcie metody reset z klasy bazowej  
        '''  
        Zresetuj pozycję sera i zatrzymaj ruch  
        Przywrócenie prędkości ruchu do wartości początkowej  
        '''  
        Wyśrodkowanie sera na ekranie w poziomie  
  
        self.movingUp = False Zatrzymanie ruchu sera w górę  
        self.movingDown = False Zatrzymanie ruchu sera w dół  
  
        self.position[0] = (self.game.width - self.image.get_width()) / 2  
        self.position[1] = (self.game.height - self.image.get_height()) / 2  
        self.movement_speed=[5,5] Ustawienie początkowej prędkości ruchu sera  
                                Wyśrodkowanie sera na ekranie w pionie  
                                Wyśrodkowanie sera na ekranie w poziomie  
  
    def update(self):  
        '''  
        Zresetuj pozycję sera i zatrzymaj jego ruch  
        po ekranie  
        '''  
        if self.movingUp: Jeśli poruszamy się w góre, przesuń ser w góre  
            self.position[1] = self.position[1] - (self.movement_speed[1])
```

```

if self.movingDown: Jeśli poruszamy się w dół, przesuń ser w dół
    self.position[1] = self.position[1] + (self.movement_speed[1])

if self.position[0] < 0: Zatrzymanie ruchu po osiągnięciu lewej krawędzi ekranu
    self.position[0]=0
if self.position[1] < 0: Zatrzymanie ruchu po osiągnięciu górnej krawędzi ekranu
    self.position[1]=0 Zatrzymanie ruchu po osiągnięciu prawej krawędzi ekranu
if self.position[0] + self.image.get_width() > self.game.width:
    self.position[0] = self.game.width - self.image.get_width()
if self.position[1] + self.image.get_height() > self.game.height:
    self.position[1] = self.game.height - self.image.get_height()

Zatrzymanie ruchu po osiągnięciu dolnej krawędzi ekranu
def StartMoveUp(self): Metoda wywoływana w celu rozpoczęcia ruchu sera w góre ekranu
    'Rozpoczęcie ruchu sera w góre'
    self.movingUp = True Ustawienie flagi ruchu w góre na True

def StopMoveUp(self): Metoda wywoływana w celu zatrzymania ruchu sera w góre ekranu
    'Zatrzymanie ruchu sera w góre'
    self.movingUp = False Ustawienie flagi ruchu w góre na False

# 'Tutaj znajdują się inne metody obsługi ruchu sera...'

```



ANALIZA KODU

Postać gracza

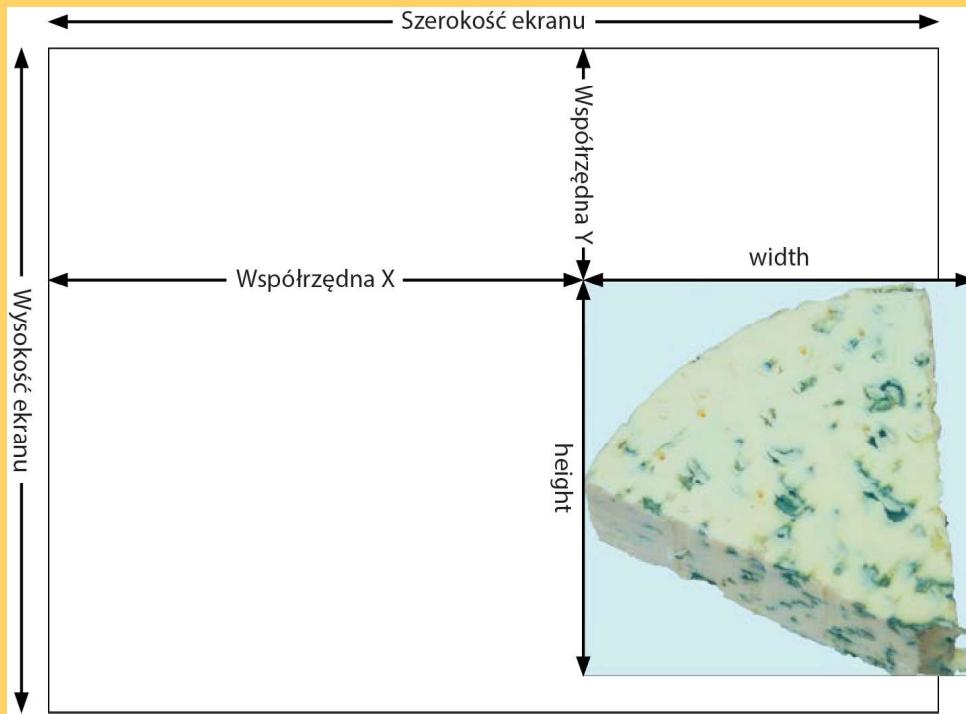
Powyższy kod definiuje postać sera. Pominąłem niektóre metody ruchu, aby zaoszczędzić miejsca w książce. Kompletny kod można znaleźć w przykładowym programie *EG16-06 Gracz-ser* w kodzie z przykładami do tego rozdziału. Możesz mieć kilka pytań na ten temat.

Pytanie: Dlaczego klasa Cheese nie ma metod `__init__` ani `draw`?

Odpowiedź: Klasa `Cheese` jest podklassą stworzonej wcześniej klasy `Sprite`. To oznacza, że klasa `Cheese` dziedziczy obie te metody z klasy `Sprite`.

Pytanie: Do czego służą metody `get_width` i `get_height`?

Odpowiedź: Te metody są dostarczane przez klasę `image` z biblioteki `pygame`. Umożliwiają określenie wymiarów obrazu. Używamy ich, aby nie dopuścić do tego, by gracz mógł przemieścić ser poza ekran.



Powyższa ilustracja pokazuje, jak to działa. Program zna położenie sera oraz szerokość i wysokość ekranu. Jeśli pozycja x zwiększcza o szerokość sera jest większa niż szerokość ekranu (tak jest na powyższym rysunku), to metoda aktualizacji dla sera umieści ser w położeniu odpowiadającym prawej krawędzi:

```
if self.position[0] + self.image.get_width() > self.game.width:  
    self.position[0] = self.game.width - self.image.get_width()
```

Pozycja postaci jest przechowywana na liście, a element na pozycji 0 przechowuje współrzędną x. Postać może wykorzystać referencję do obiektu game, aby uzyskać szerokość ekranu, oraz metodę get_width, aby uzyskać szerokość obrazu postaci. Zauważmy, że na powyższej ilustracji ser nie wychodzi poza dolną krawędź ekranu. Wymuszenie od postaci pozostania na ekranie w ten sposób to tzw. *clamping* (dosł. mocowanie).

Klasa Cheese używa również szerokości i wysokości obrazu postaci, aby ustawić ser na środku ekranu po jego zresetowaniu.

```
self.position[0] = (self.game.width - self.image.get_width()) / 2  
self.position[1] = (self.game.height - self.image.get_height()) / 2
```

Sterowanie postacią gracza

Klasa game tworzy egzemplarz postaci sera i wykorzystuje zdarzenia klawiatury, aby wysyłać do niego komunikaty w celu sterowania jego ruchem. Kod klasy game, który jest za to odpowiedzialny, zamieszczono poniżej. Aby zobaczyć, jak to działa, można uruchomić przykładowy program EG16-06 Gracz-ser. Gracz może poruszać serem po ekranie, ale ser nie przesunie się poza krawędź ekranu.

```
cheese_image = pygame.image.load('cheese.png')           Załadowanie obrazu sera
self.cheese_sprite = Cheese(image=cheese_image, game=self)  Utworzenie krotki opisującej pozycję obrazu sera

clock = pygame.time.Clock()                            Utworzenie obiektu zegara kontrolującego grę

while True:                                         Początek pętli gry
    clock.tick(60)                                    Ustawienie odświeżania gry na 60 klatek na sekundę
    for e in pygame.event.get():                     Przetwarzanie zdarzeń w grze
        if e.type == pygame.KEYDOWN:                  Czy wciśnięto klawisz?
            if e.key == pygame.K_ESCAPE:              Czy naciśnięto klawisz Escape?
                pygame.quit()                         Zamknięcie gry
                return
            elif e.key == pygame.K_UP:                 Czy naciśnięto klawisz strzałki w górę?
                self.cheese_sprite.StartMoveUp()       Rozpoczęcie ruchu sera w górę
            elif e.key == pygame.K_DOWN:              Czy naciśnięto klawisz strzałki w dół?
                self.cheese_sprite.StartMoveDown()     Rozpoczęcie ruchu sera w dół
            'Tutaj znajdują się inne metody obsługi ruchu sera...'

    self.background_sprite.draw()                    Narysowanie tła
    self.background_sprite.update()                 Aktualizacja tła
    self.cheese_sprite.draw()                      Narysowanie sera
    self.cheese_sprite.update()                   Aktualizacja obrazu sera
    pygame.display.flip()                         Przełączenie bufora wyświetlenia, aby działania rysowania stały się widoczne
```

Postać krakersa

Poruszanie serem po ekranie jest zabawne przez chwilę, ale trzeba jeszcze dodać jakieś cele dla gracza. Celem są krakersy, które gracz musi łapać za pomocą sera. Po złapaniu krakersa wynik gry się zwiększa, a krakers przechodzi na inną losową pozycję na ekranie. Klasa Cracker jest podklassą klasy Sprite:

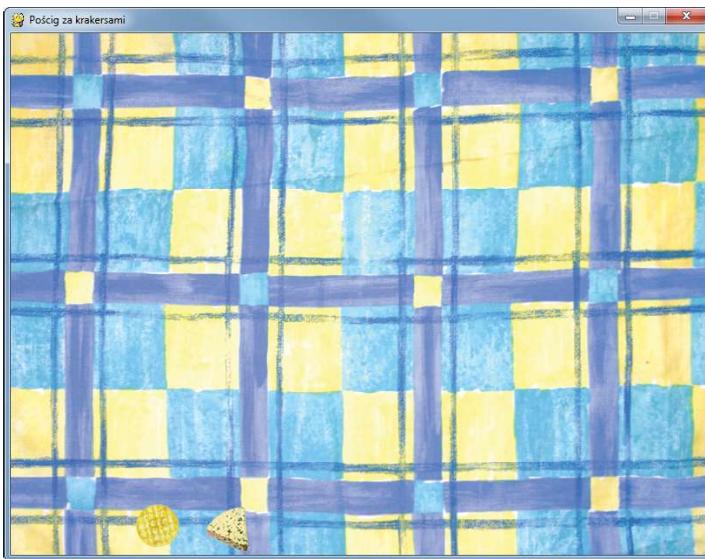
```

class Cracker(Sprite):
    """
    Krakers jest celem dla sera
    Po zresetowaniu przechodzi do nowego losowego miejsca
    na ekranie
    """
    def reset(self):
        self.position[0] = random.randint(0,
            self.game.width-self.image.get_width())
        self.position[1] = random.randint(0,
            self.game.height-self.image.get_height())

```

Klasa `Cracker` jest bardzo mała, ponieważ większość swoich zachowań uzyskuje ze swojej klasy bazowej — `Sprite`. Zawiera tylko jedną metodę — `reset` — która używa generatora liczb losowych Pythona do wybrania losowej pozycji krakersa. Aby dodać krakersa do gry, należy utworzyć jego egzemplarz, a następnie narysować go w pętli gry. Sposób działania tego mechanizmu pokazano w przykładowym programie *EG16-07 Ser i krakersy*.

Działającą grę pokazano na rysunku 16.4. Z rysunku widać, że z ta grą są co najmniej dwa problemy. Po pierwsze, krakers wydaje się być nad serem. Jeśli ser ma „łapać” krakersy, to wyglądałoby lepiej, gdyby to ser był „nad” krakersem. Możemy to naprawić, zmieniając kolejność rysowania elementów gry. Framework `pygame` umieszcza obrazy na ekranie w kolejności ich rysowania. Drugim problemem z tą grą jest to, że jest trochę nudna. Myślę, że potrzebujemy więcej krakersów jako dodatkowych celów.



Rysunek 16.4. Ser i krakers

Dodanie wielu egzemplarzy klasy Sprite

Moglibyśmy zwiększyć liczbę krakersów, tworząc kolejne egzemplarze klasy Cracker:

```
cracker_image = pygame.image.load('cracker.png')
self.cracker1 = Cracker(image=cracker_image, game=self)
self.cracker2 = Cracker(image=cracker_image, game=self)
self.cracker3 = Cracker(image=cracker_image, game=self)
```

Powyższy kod utworzyłby trzy krakersy o nazwach `cracker1`, `cracker2` i `cracker3`. To by działało, ale byłoby bardzo trudno zarządzać krakersami, ponieważ gra musiałaby się aktualizować i za każdym razem oddziennie rysować każdą z tych postaci. Gdyby użytkownicy gry zażądali 50 krakersów na ekranie, byłoby to prawdziwym problemem. Zawsze gdy mieliśmy ten problem w przeszłości, używaliśmy do jego rozwiązania jakiegoś rodzaju kolekcji (zazwyczaj listy). Tutaj także możemy to zrobić.

```
self.sprites = []
cracker_image = pygame.image.load('cracker.png')
for i in range(20):
    cracker_sprite = Cracker(image=cracker_image, game=self)
    self.sprites.append(cracker_sprite)
```

Annotations from left to right:

- Utworzenie listy, która będzie zawierać wszystkie postacie w grze
- Załadowanie obrazu krakersa
- Utworzenie pętli for, która wykona się 20 razy
- Utworzenie egzemplarza klasy Cracker
- Dodanie postaci do listy postaci w grze

Powyższe instrukcje tworzą 20 obiektów reprezentujących krakersy. Gra zawiera teraz listę o nazwie `sprites`, która zawiera wszystkie postacie występujące w grze.

```
for sprite in self.sprites:
    sprite.update()

for sprite in self.sprites:
    sprite.draw()
```

Powyżej znajdują się instrukcje, których możemy użyć w pętli gry, aby zaktualizować i narysować postacie krakersów. Możesz zobaczyć, jak to działa, w przykładowej grze *EG16-08 Ser i krakersy*. W tej wersji gry dodano do listy `sprites` także obiekty reprezentujące ser i tło. W związku z tym wszystkie elementy w grze są rysowane i aktualizowane przez powyższe dwie pętle. Grę w bieżącej formie pokazano na rysunku 16.5. Aby dodać więcej krakersów, wystarczy zmienić limit wywołania `range` w pętli `for`, która je tworzy.

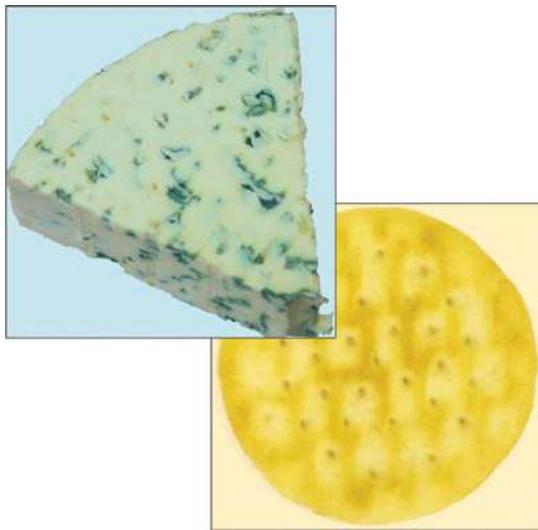


Rysunek 16.5. Ser i wiele krakersów

Łapanie krakersów

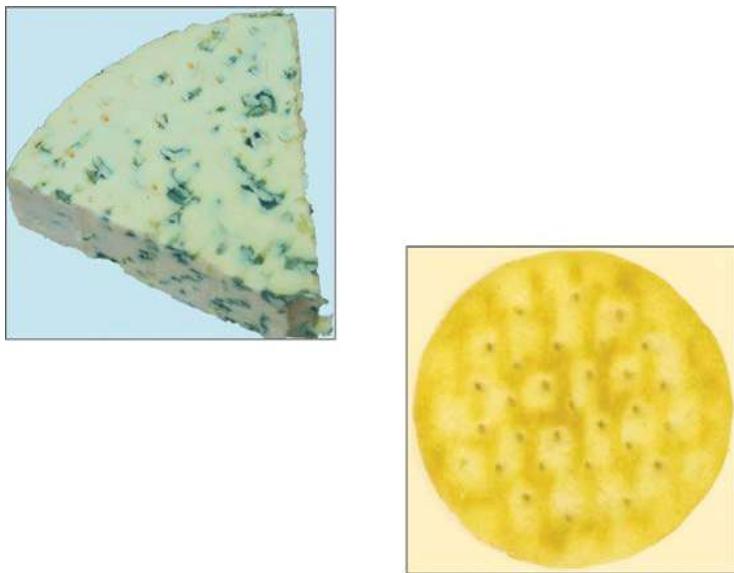
W grze jest teraz dużo krakersów i kawałek sera, który może je ścigać. Ale gdy ser „zlapie” krakersa, nic się nie dzieje. Trzeba dodać zachowanie klasy `Cracker`, które wykryje sytuację, kiedy krakers zostanie „złapany” przez ser. Krakers jest złapany przez ser, gdy ser porusza się „nad” krakersem. Gra może wykryć, kiedy tak się dzieje, sprawdzając, czy prostokąty otaczające dwie postacie przecinają się.

Ser w procesie łapania krakersa zilustrowano na rysunku 16.6. Prostokąty wokół obrazów serów i krakersów to tzw. **ramki ograniczające** (ang. *bounding box*). Gdy jedna ramka ograniczająca porusza się „wewnętrz” innej, mówimy, że te dwie ramki się przecinają. Kiedy krakers się zaktualizuje, sprawdzi, czy krzyżuje się z serem.



Rysunek 16.6. Przecinające się postacie gry

Sposób działania testu pokazano na rysunku 16.7. Na tym rysunku dwie postacie nie przecinają się, ponieważ prawa krawędź sera znajduje się po lewej stronie lewej krawędzi krakersa. Innymi słowy: pozycja sera jest zbyt daleko w lewo, aby ser przecinał się z krakersem. Byłoby to również prawdą, gdyby ser był powyżej, poniżej lub po prawej stronie krakersa. Możemy stworzyć metodę, która sprawdza prawdziwość dla tych czterech warunków. Jeśli którykolwiek z nich jest prawdziwy, prostokąty się nie przecinają.



Rysunek 16.7. Nieprzecinające się postacie gry

```

def intersects_with(self, target):
    """
    Zwraca True, jeśli ta postać przecina się z
    obiektem target przekazanym za pomocą parametru
    """

    max_x = self.position[0]+self.image.get_width() Odczytanie prawej krawędzi tej postaci
    max_y = self.position[1]+self.image.get_height() Odczytanie dolnej krawędzi tej postaci

    target_max_x = target.position[0]+target.image.get_width() Odczytanie prawej krawędzi celu
    target_max_y = target.position[1]+target.image.get_height() Odczytanie dolnej krawędzi celu

    if max_x < target.position[0]: Czy ta postać znajduje się po lewej?
        return False

    if max_y < target.position[1]: Czy ta postać jest poniżej?
        return False

    if self.position[0] > target_max_x: Czy ta postać znajduje się po prawej?
        return False

    if self.position[1] > target_max_y: Czy ta postać jest powyżej?
        return False

    # jeśli dotarliśmy do tego miejsca, to obiekty się przecinają
    return True Zwraca True, ponieważ postacie się przecinają

```

Metoda jest atrybutem obiektu `Sprite`. Zwraca wartość `True`, jeśli postać przecina się z określonym obiektem docelowym. Dodajemy tę metodę do klasy `Sprite`, aby mogły z niej korzystać wszystkie postacie. Teraz możemy dodać do klasy `Cracker` metodę aktualizacji, która sprawdza, czy krakers krzyżuje się z serem:

```

def update(self):
    if self.intersects_with(game.cheese_sprite): Czy zostaliśmy złapani?
        self.captured_sound.play() Odtworzenie efektu dźwiękowego towarzyszącego złapaniu krakersa
        self.reset() Zresetowanie pozycji krakersa

```

Obsługa dźwięku

Powyzsza metoda `update` odtwarza efekt dźwiękowy, gdy krakers zostanie „schwytyany” przez ser. Framework pygame udostępnia klasę `Sound`, której można użyć do zarządzania odtwarzaniem dźwięku. W momencie tworzenia egzemplarza klasy `Sound` przekazujemy do niej nazwę pliku zawierającego dane dźwięku.

```
cracker_eat_sound = pygame.mixer.Sound('burp.wav')
```

Powyzsza instrukcja tworzy egzemplarz klasy `Sound` o nazwie `cracker_eat_sound` na podstawie pliku dźwiękowego *burp.wav*. Ten dźwięk przekazujemy do konstruktora obiektu `Cracker`:

```
cracker_sprite = Cracker(image=cracker_image, game=self,  
                           captured_sound=cracker_eat_sound)
```

Zapisanie dźwięku schwytyania
w obiekcie reprezentującym krakersa

Aby przechowywać dźwięk w obiekcie krakersa, musimy zmodyfikować metodę `__init__` w klasie `Cracker`:

```
def __init__(self, image, game, captured_sound):  
    super().__init__(image, game)  
    self.captured_sound = captured_sound
```

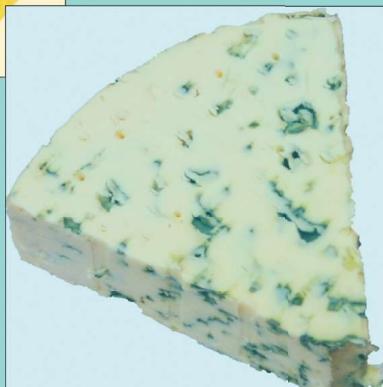
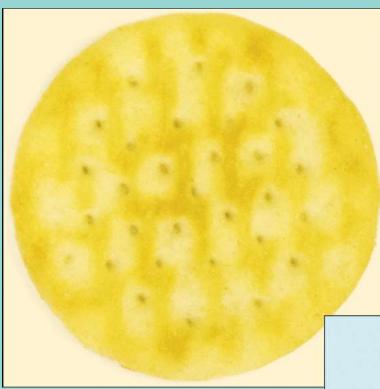
Wywołanie konstruktora klasy bazowej
Ustawienie atrybutu `sound` obiektu
reprezentującego krakersa

Atrybutu `captured_sound` w obiekcie `Cracker` możemy użyć do odtworzenia efektu dźwiękowego, gdy ten krakers zostanie schwytyany. W obecnej wersji gry wszystkie krakersy, gdy są jedzone, wydają takie same dźwięki. Jednak gdybyśmy chcieli, moglibyśmy użyć różnych efektów dźwiękowych dla każdego krakersa. Na łapanie krakersów pozwala graczowi przykładowy program *EG16-09 Łapanie krakersów*. Po złapaniu krakersa wynik gry się zwiększa, a krakers jest przesuwany do innej, losowej pozycji na ekranie.

Do tworzenia własnych efektów dźwiękowych można użyć programu Audacity, który pozwala na przechwytywanie i edycję dźwięków. Jest do pobrania za darmo ze strony www.audacity-team.org i jest dostępny dla większości systemów operacyjnych.



Nieprawidłowe wykrywanie kolizji



Wykorzystywanie ramek ograniczających do wykrywania kolizji wiąże się z pewnymi problemami.

Na powyższej ilustracji widać, że ser i krakers nie stykają się ze sobą, ale algorytm zaimplementowany w grze wykryje, że kolizja istnieje. Nie powinno to być zbyt dużym problemem w przypadku naszej gry. W tej formie zadanie gracza jest łatwiejsze, ponieważ aby zdobyć punkt, niekoniecznie trzeba znaleźć się dokładnie nad krakersem. Gracz może mieć jednak powody do narzekania, jeśli z powodu tego problemu gra zdecyduje, że został złapany przez zabójczego pomidora. Istnieją trzy sposoby rozwiązywania tego problemu.

- Gdy ramki ograniczające się przecinają (tak jak powyżej), można sprawdzić przecinający się prostokąt (część wspólną przecinających się ramek), aby zobaczyć, czy są w nim jakieś wspólne piksele. Takie postępowanie zapewnia bardzo precyzyjne wykrywanie kolizji, ale spowalnia grę.
- Ewentualnie można wykrywać kolizje, wykorzystując odległość zamiast przecięcia. Ten sposób działa dobrze, jeśli postacie w grze są w większości okrągłe.
- Ostatnie rozwiązywanie podoba mi się najbardziej. Móglbym spowodować, aby wszystkie obrazy w grze były prostokątne. Wtedy postacie wypełniają ramki ograniczające, a gracz zawsze widzi, kiedy nastąpiła kolizja.

KĄCIK PROGRAMISTY

Gdy piszesz grę, kontrolujesz wszechświat

Jednym z powodów, dla których lubię pisać gry, jest to, że mam pełną kontrolę nad tym, co robię. Jeśli rozwiążę problem dla klienta, muszę dostarczyć konkretne wyniki. Ale w grze, gdy napotkam problem, mogę zmienić to, co ona robi. Mogę również przedefiniować rozgrywkę, jeśli popełnię błąd w programie. Czasami w rezultacie powstaje ciekawsze zachowanie niż to, które próbowałem stworzyć. Przydarzyło mi się to w wielu sytuacjach.

Implementacja zabójczego pomidora

W obecnej formie gra niezbyt przypomina grę. Na gracza nie czyhają żadne niebezpieczeństwa. Kiedy tworzysz grę, ustawiasz cel, który gracz próbuje osiągnąć. Następnie dodajesz elementy, które sprawią, że osiągnięcie tego celu będzie trudne. W przypadku gry *Pościg za krakersami* chcę dodać „zabójcze pomidory”, które będą nieustannie ścigać gracza. Chcę, aby w trakcie gry gracz był ścigany przez coraz większą liczbę pomidorów. Będzie ich przybywać tak długo, aż gra zacznie polegać wyłącznie na przeżyciu. Pomidory będą interesujące, ponieważ wprowadzę do nich mechanizmy **sztucznej inteligencji i fizyki**.

Dodanie „sztucznej inteligencji” do postaci

Implementacja sztucznej inteligencji wydaje się bardzo trudna do osiągnięcia, ale w przypadku tej gry jest wyjątkowo prosta. W gruncie rzeczy sztuczna inteligencja w grze oznacza po prostu stworzenie programu, który w określonej sytuacji zachowywałby się tak jak człowiek. Gdybym mnie ścigał, próbowałby się do mnie zbliżyć. Kierunek, w którym byś się poruszał, zależałby od mojej pozycji względem Ciebie. Gdybym był z lewej strony, poruszałby się w lewo itd.

Analogiczne zachowanie możemy zaimplementować w postaci zabójczego pomidora:

```
if game.cheese_sprite.position[0] > self.position[0]:  
    self.x_speed = self.x_speed + self.x_accel  
else:  
    self.x_speed = self.x_speed - self.x_accel  
  
if game.cheese_sprite.position[1] > self.position[1]:  
    self.y_speed = self.y_speed + self.y_accel  
else:  
    self.y_speed = self.y_speed - self.y_accel
```

Czy gracz znajduje się po prawej stronie pomidora?
Przyspieszenie ruchu w prawo
Przyspieszenie ruchu w lewo
Czy gracz znajduje się poniżej pomidora?
Przyspieszenie ruchu w dół
Przyspieszenie ruchu w górę

Powyższe instrukcje warunkowe pokazują, w jaki sposób można stworzyć inteligentnego zabójczego pomidora. Program porównuje pozycje x obiektów cheese_sprite i tomato. Jeśli ser znajduje się po prawej stronie pomidora, szybkość pomidora w kierunku x zwiększa

się, tak aby poruszał się w prawo. Jeśli ser znajduje się po lewej stronie pomidora, przyspieszenie następuje w przeciwnym kierunku. Kod zamieszczony powyżej powtarza ten proces dla pozycji dwóch postaci w pionie. W efekcie powstał pomidor, który inteligentnie porusza się w kierunku sera. Zwrócmy uwagę, że moglibyśmy stworzyć „tchórzliwego” pomidora, który ucieka od gracza, dokonując ujemnych przyspieszeń. W ten sposób pomidor przyspieszałby w kierunku przeciwnym do sera.

KĄCIK PROGRAMISTY

Dzięki użyciu „sztucznej inteligencji” gry stają się dużo bardziej interesujące

Istnieje wiele dyskusji na temat tego, czy „sztuczna inteligencja w grze” jest „właściwą” sztuczną inteligencją. Bardzo interesujący dialog na temat tego problemu można znaleźć tutaj: <https://software.intel.com/en-us/articles/designing-artificial-intelligence-for-games-part-1>. Uważam, że tego rodzaju oprogramowanie można nazwać „sztuczną inteligencją”, ponieważ gracze w obliczu takich postaci jak zabójczy pomidor naprawdę reagują tak, jakby wchodzili w interakcję z postacią intelligentną. Dzięki zaimplementowaniu w obiektach gier rodzaju inteligencji podobnej do opisanej powyżej możemy sprawić, że gra stanie się znacznie bardziej przekonująca.

Dodanie mechanizmów „fizyki” postaci

Za każdym razem gdy gra się aktualizuje, może zaktualizować położenie obiektów na ekranie. Odcinek, który pokonuje każdy obiekt podczas aktualizacji gry, określa **prędkość** obiektu. Gdy gracz się porusza, pozycja sera jest aktualizowana o wartość 5. Innymi słowy: gdy gracz przytrzymuje klawisz strzałki, pozycja sera w tym kierunku zmienia się o 5. Aktualizacje występują 60 razy na sekundę, ponieważ taka jest szybkość, z jaką działa pętla gry. Innymi słowy: w ciągu jednej sekundy ser poruszy się o 300 pikseli ($60 * 5$). Możemy zwiększyć prędkość sera, dodając przy każdej aktualizacji większą wartość do jego pozycji. Gdybyśmy użyli prędkości 10, okazałoby się, że ser porusza się dwa razy szybciej.

Przyspieszenie to wielkość zmiany wartości prędkości. Poniższe instrukcje aktualizują wartość `x_speed` obiektu pomidora o wartość przyspieszenia, a następnie wykorzystują tę wartość do obliczenia pozycji pomidora.

```
self.x_speed = self.x_speed + self.x_accel                          Dodanie przyspieszenia do prędkości  
self.position[0] = self.position[0] + self.x_speed                  Aktualizacja pozycji postaci
```

Początkowa prędkość pomidora jest ustalona na zero, więc przy każdej aktualizacji pomidora prędkość (a zatem pokonany odcinek) się zwiększy. Jeśli zrobimy to w połączeniu ze „sztuczną inteligencją”, uzyskamy obiekt pomidora, który będzie się szybko poruszał w kierunku gracza.

Gdybyśmy po prostu pozwolili pomidorowi na ciągłe przyspieszanie, okazałoby się, że pomidor stawałby się coraz szybszy, a rywalizowanie z nim zrobiłoby się niemożliwe.

Poniższa instrukcja dodaje trochę „tarcia”, by spowolnić ruch pomidora. Wartość tarcia jest mniejsza niż 1, więc za każdym razem gdy pomnożymy prędkość przez tarcie, prędkość się zmniejszy, co spowoduje zwalnianie pomidora z biegiem czasu.

```
self.x_speed = self.x_speed * self.friction_value
```

Pomnożenie prędkości przez tarcie

Wartości tarcia i przyspieszenia są ustawiane w metodzie `reset` dla obiektu pomidora:

```
def reset(self):
    self.entry_count = 0
    self.friction_value = 0.99
    self.x_accel = 0.2
    self.y_accel = 0.2
    self.x_speed = 0
    self.y_speed = 0
    self.position = [-100, -100]
```

Po kilku eksperymentach doszedłem do wartości przyspieszenia 0,2 i wartości tarcia 0,99. Gdybym chciał stworzyć postać, która ściga mnie szybciej, mógłbym zwiększyć przyspieszenie. Gdybym chciał stworzyć postać, która mocniej spowalnia, mogę zwiększyć tarcie. Manipulowanie tymi wartościami to świetna zabawa. Możemy tworzyć postacie, które poruszają się powoli w kierunku gracza, a dzięki zastosowaniu ujemnego przyspieszenia możemy spowodować, że będą od niego uciekały.

KĄCIK PROGRAMISTY

Gdy piszesz grę, zawsze możesz oszukiwać

Kiedy piszesz grę, zawsze powinieneś zacząć od najprostszego, najszybszego sposobu uzyskania efektu, a następnie poprawiać go, jeśli to konieczne.

„Fizyka”, której używam, nie jest tak naprawdę dokładną symulacją fizycznych obiektów. Sposób, w jaki zaimplementowałem tarcie, nie jest zbyt realistyczny, ale działa i daje graczowi dobre wrażenia. Uważam za interesujący fakt, że za pomocą sześciu lub siedmiu linijek kodu w Pythonie można stworzyć coś, co zachowuje się w tak wiarygodny sposób. W grze *Pościg za krakersami* wykorzystano bardzo prosty mechanizm wykrywania kolizji, sztuczną inteligencję i zasady fizyki, a pomimo to gra się bardzo przyjemnie. Naprawdę odnosi się wrażenie, jakby ścigały nas pomidory. Tworzenie dokładnego modelu fizycznego wymagałoby dużo dodatkowej pracy, a wniosłoby do gry bardzo niewiele.

Tworzenie wersji gry z opóźnieniem

Ważne jest, aby gra była progresywna. Gdyby zaczęła się od wyświetlenia mnóstwa zabójczych pomidorów, gracz nie przetrwałby długo i nie byłby zadowolony z tego doświadczenia. Chciałbym, aby kolejny pomidor pojawiał się co 5 sekund. Możemy to zrobić, nadając każdemu pomidorowi podczas jego tworzenia wartość „wejściowego opóźnienia”:

```
tomato_image = pygame.image.load('tomato.png')

for entry_delay in range(300,3000,300):
    tomato_sprite = Tomato(image=tomato_image,
                           game=self,
                           entry_delay=entry_delay)
    self.sprites.append(tomato_sprite)
```

The diagram shows the flow of the code from top to bottom. It consists of four horizontal bars: a grey bar for the imports and loop setup, and three teal bars for the instantiation and addition of the Tomato objects. Each bar has a corresponding text label to its right.

- Pierwsza linia: `tomato_image = pygame.image.load('tomato.png')` → **Pętla do generowania wartości opóźnienia wejścia**
- Druga linia: `for entry_delay in range(300,3000,300):` → **Utworzenie nowego obiektu pomidora**
- Trzecia linia: `tomato_sprite = Tomato(image=tomato_image,` → **Przypisanie do obiektu pomidora wartości opóźnienia wejścia**
- Czwarta linia: `game=self,` → **Dodanie pomidora do listy postaci**
- Piąta linia: `entry_delay=entry_delay)`
- szósta linia: `self.sprites.append(tomato_sprite)`

W tym kodzie użyto wersji funkcji `range`, której jeszcze nie widzieliśmy. Pierwszy argument funkcji `range` to wartość początkowa, która w tym przypadku wynosi 300. Drugi argument to górny limit, a trzeci argument to „krok” między wartościami. W ten sposób uzyskamy wartości `entry_delay`, które zaczynają się od 300, a następnie stopniowo „idą w górę” do 2700 (pamiętaj, że górny limit to wartość 3000).

Metoda `__init__` w klasie `Tomato` przechowuje wartość `entry_delay` i służy do opóźnienia pojawienia się postaci:

```
def update(self):

    self.entry_count = self.entry_count + 1
    if self.entry_count < self.entry_delay:
        return
```

The diagram shows two horizontal bars: a grey bar for the update method body and a teal bar for the condition check. Each bar has a corresponding text label to its right.

- Grey bar: `def update(self):`
- Teal bar: `self.entry_count = self.entry_count + 1` → **Zwiększenie licznika wejść o 1**
- Teal bar: `if self.entry_count < self.entry_delay:` → **Jeśli licznik wpisów jest mniejszy od opóźnienia, zwróć sterowanie**
- Grey bar: `return`

Metoda `update` jest wywoływana 60 razy na sekundę. Pierwszy pomidor ma opóźnienie wynoszące 300, co oznacza, że pojawi się w chwili $300/60$ sekund, czyli 5 sekund po rozpoczęciu gry. Następny pomidor pojawi się 5 sekund po tym itd., aż do ostatniego. Sposób działania tego mechanizmu ilustruje przykładowy program *EG16-10 Zabójczy pomidor*. Pojawienie się kilku pomidorów na raz, które Cię ścigają, może być dość irytujące.

Dokończenie gry

Mamy zatem program, który pozwala trochę pograć. Teraz trzeba go przekształcić we właściwą grę. Aby to zrobić, należy dodać ekran startowy, zapewnić sposób rozpoczęcia gry, wykrycia jej zakończenia i zarządzania nim oraz — ze względu na znaczne podniesienie atrakcyjności gry — dodanie zapamiętywania najlepszego wyniku.

Dodanie ekranu startowego

Ekran startowy to — jak łatwo zgadnąć — miejsce, w którym rozpoczynamy grę. Następnie, gdy gra jest skończona, program powraca do ekranu startowego. Ekran startowy w grze *Pościg za krakersami* możemy dodać, korzystając z wartości flagi określającej tryb gry.

```
def start_game(self):
    for sprite in self.sprites:
        sprite.reset()
    self.score=0
    self.game_running = True
```

Zresetowanie wszystkich postaci
Wyzerowanie wyniku gry
Ustawienie flagi, która wskazuje, że gra się toczy

Powyżej znajduje się metoda, która rozpoczyna grę. Resetuje wszystkie postacie, ustawia wynik na zero, a następnie ustawia flagę game_running na True. Flaga game_running steruje zachowaniem pętli gry:

```
while True:
    clock.tick(60)
    if self.game_running:
        self.update_game()
        self.draw_game()
    else:
        self.update_start()
        self.draw_start()
    pygame.display.flip()
```

Nieskończona pętla gry
Ustawienie wartości tempa wyświetlania na 60 klatek na sekundę
Czy gra jest aktywna?
Aktualizacja gry
Narysowanie gry
Aktualizacja ekranu startowego
Narysowanie ekranu startowego
Wyświetlenie bufora back buffer

Jest to główna pętla gry. Kod, który aktualizuje grę i rysuje ją, jest teraz w metodach, które są wywoływane, jeśli gra jest uruchomiona. Jeżeli gra nie jest uruchomiona, są wywoływane metody, które aktualizują i rysują ekran startowy.

```
def update_start(self):
    for e in pygame.event.get():
        if e.type == pygame.KEYDOWN:
            if e.key == pygame.K_ESCAPE:
                pygame.quit()
                sys.exit()
            elif e.key == pygame.K_g:
                self.start_game()
```

Przetwarzanie wszystkich zdarzeń pygame
Czy wciśnięto klawisz?
Czy wciśnięto klawisz Escape?
Zamknięcie pygame
Wyjście z programu

Metoda aktualizacji ekranu startowego sprawdza dwa klawisze:

- Jeśli użytkownik nacisnął klawisz *G*, uruchamiana jest metoda `start_game`, która rozpoczyna grę.
- Jeśli wciśnięto klawisz *Escape*, metoda zakończy pygame przez wywołanie metody `quit`, a następnie wykorzysta metodę `exit` z modułu `sys` w celu zakończenia programu.

Wykorzystanie metody exit do zamknięcia Pythona

Metoda `exit` znajduje się w module `sys`, co oznacza, że trzeba zaimportować ten moduł:

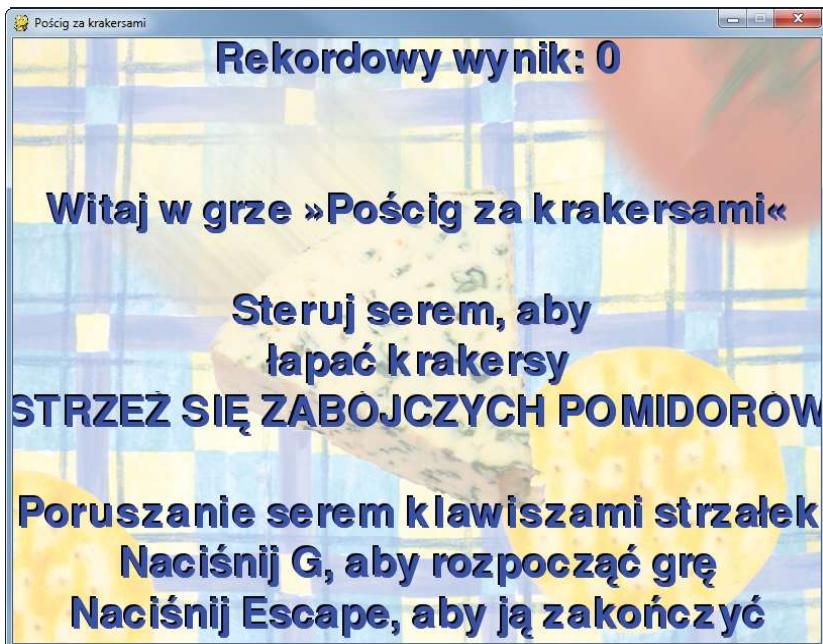
```
import sys
```

Po zainportowaniu modułu `sys` można wywołać funkcję `exit` z tego modułu, aby natychmiast zakończyć Pythona.

```
sys.exit()
```

Rysowanie tekstu za pomocą pygame

Ekran startowy wyświetla informacje dla gracza, tak jak pokazano na rysunku 16.8. Framework pygame zawiera mechanizmy pozwalające rysować tekst na ekranie. Moduł używa obiektu `Font`, który jest tworzony podczas uruchamiania gry.



Rysunek 16.8. Ekran startowy

```
self.font = pygame.font.Font(None, 60)
```

Inicjalizator czcionki pobiera dwa parametry: krój czcionki i jej rozmiar. Powyższa instrukcja ustawia `None` dla projektu czcionki. Powoduje to wybranie domyślnej czcionki modułu `pygame`. Zastosowanie rozmiaru 60 daje tekst o rozmiarze, który dobrze pasuje do gry. Aby umieścić komunikat na ekranie, gra najpierw renderuje tekst za pomocą obiektu `font`.

```
text = self.font.render('witaj świecie', True, (255,0,0))
```

Metoda `render` wymaga trzech argumentów:

- Pierwszy to ciąg zawierający tekst do wyrenderowania.
- Drugi argument wybiera tzw. **aliasing**. Ta technika wygładza krawędzie znaków. Należy jej użyć, aby tekst wyglądał estetycznie.
- Trzeci argument określa kolor tekstu. Zawiera składowe czerwoną, niebieską i zieloną, które powinny składać się na kolor tekstu. Maksymalna intensywność kolorów to 255.

Powyższy kod wyrenderuje komunikat „witaj świecie” w kolorze jaskrawoczerwonym.

Następnym krokiem po wyrenderowaniu tekstu jest przeniesienie go na ekran. Robimy to w ten sam sposób, w jaki umieszczamy na ekranie obrazy.

```
self.surface.blit(text, (0,0))
```

Pierwszym argumentem metody `blit` jest tekst do narysowania. Drugi to jego położenie na ekranie. Powyższa instrukcja wyświetli komunikat „witaj świecie” w lewym górnym rogu ekranu. Program może uzyskać szerokość i wysokość renderowanego tekstu. Wartości te mogą być użyte w celu wyśrodkowania tekstu na ekranie. Klasa `CrackerChase` zawiera krótką metodę, która rysuje tekst na ekranie:

```
def display_message(self, message, y_pos):
    """
    Wyświetla komunikat na ekranie
    Pierwszym argumentem jest tekst komunikatu
    Drugi argument to pozycja
    tekstu w pionie
    Metoda rysuje tekst wyśrodkowany na ekranie
    Jest narysowany z czarnym cieniem
    """
    shadow = self.font.render(message, True, (0,0,0)) Renderowanie tekstu czarnym kolorem
    text = self.font.render(message, True, (0,0,255)) Renderowanie tekstu niebieskim kolorem
    text_position = [self.width/2 - text.get_width()/2, y_pos] Narysowanie cienia
    self.surface.blit(shadow, text_position) Przesunięcie pozycji rysowania w dół
    text_position[0] += 2 Przesunięcie pozycji rysowania w dół
    text_position[1] += 2 Narysowanie tekstu
    self.surface.blit(text, text_position) Obliczenie pozycji tekstu
```

Ta metoda w istocie rysuje tekst dwukrotnie. Za pierwszym razem tekst jest rysowany kolorem czarnym, a następnie tekst jest rysowany ponownie, tym razem w kolorze niebieskim. Podczas rysowania za drugim razem tekst jest nieznacznie przesuwany, tak aby wyglądało, że czarny tekst jest cieniem.

W metodzie użyto operatora `+=`, który można wykorzystać do zwiększenia wartości zmiennej. Zamiast pisać:

```
text_position[0] = text_position[0]+2
```

możesz napisać:

```
text_position[0] += 2
```

Istnieją podobne operatory do odejmowania ($-=$), mnożenia ($*=$) i dzielenia ($/=$).

Jeśli przyjrzesz się dokładnie rysunkowi 16.8, zauważysz, że wynikiem tego dodatkowego rysowania jest uzyskanie efektu trójwymiarowości tekstu, co sprawia, że staje się on lepiej widoczny na ekranie.

KĄCIK PROGRAMISTY

Nie obawiaj się „zatrudniania” sprzętu graficznego

Można by pomyśleć, że rysowanie całego tekstu na ekranie tylko po to, aby uzyskać efekt cienia, jest dość ekstrawaganckie. Jednak nowoczesny sprzęt graficzny doskonale radzi sobie z wykonywaniem wielu tysięcy operacji rysowania na sekundę. Kiedyś zdarzyło mi się narysować tekst 20 razy tylko po to, aby uzyskać estetyczny efekt zamazanego cienia. Jeśli uważasz, że coś może wyglądać dobrze, radzę spróbować, a martwić się wydajnością tylko wtedy, gdy po tym, jak to zrobisz, gra będzie sprawiała wrażenie, że działa bardzo wolno.

```
def draw_start(self):
    self.start_background_sprite.draw()
    self.display_message(message='Rekordowy wynik: ' + str(self.top_score), y_pos=0)
    self.display_message(message='Witaj w grze Pościg za krakersami', y_pos=150)
    self.display_message(message='Steruj serem, aby ', y_pos=250)
    self.display_message(message='łapać krakersy', y_pos=300)
    self.display_message(message='STRZEŻ SIĘ ZABÓJCZYCH POMIDORÓW', y_pos=350)
    self.display_message(message='Użyj strzałek do poruszania serem', y_pos=450)
    self.display_message(message='Naciśnij G, aby rozpocząć grę', y_pos=500)
    self.display_message(message='Naciśnij Escape, aby ja zakończyć', y_pos=550)
```

Powyżej znajduje się metoda `draw_start` obiektu `game`, która rysuje obraz tła, a następnie wyświetla na ekranie komunikaty pomocy.

KĄCIK PROGRAMISTY

Zadbaj o to, aby powiedzieć użytkownikom, jak należy grać w grę

W swojej długiej i wybitnej karierze informatycznej brałem udział w kilku konkursach tworzenia gier. Straciłem rachubę w liczeniu gier, w które próbowałem grać i nie udało mi się, ponieważ nie wieałem, co mam robić. Problem polega zwykle na tym, że wszyscy koncentrują się na tworzeniu gry, a nie na mówieniu ludziom, jak w nią grać. Niepowodzenie w użyciu gry, gdy zastanawiasz się, które klawisze powinieneś nacisnąć, to nie jest zbyt dobry początek, dlatego pamiętaj o zaprezentowaniu czytelnych instrukcji od samego startu.

Zakończenie gry

Ekran startowy pozwala graczowi na rozpoczęcie gry. Widzieliśmy, że gra ma dwa stany zarządzane za pomocą atrybutu `game_running`. Ten atrybut jest ustawiony na `True`, gdy gra zaczyna działać, i `False`, gdy wyświetla się ekran startowy. Teraz musimy stworzyć kod, który zarządza wartością `game_running`. Na początku tego podrozdziału zobaczyliśmy, że obiekt `game` zawiera metodę, która rozpoczęła grę. Gra zawiera również metodę odpowiedzialną za jej zakończenie.

```
def end_game(self):
    self.game_running = False
    if self.score > self.top_score:
        self.top_score = self.score
```

Metoda `end_game` ustawia flagę `game_running` na `False`. Aktualizuje również wartość `top_score`. Jeśli bieżąca wartość pola `_score` jest większa niż dotychczasowy najwyższy wynik, to następuje aktualizacja najlepszego wyniku.

KĄCIK PROGRAMISTY

Wprowadzenie rejestracji rekordowego wyniku sprawia, że gra jest znacznie bardziej interesująca

Dodanie mechanizmu rejestracji rekordowego wyniku w grze znacznie podnosi jej atrakcyjność. Gracze poświęcają mnóstwo energii, dając do pobicia swoich poprzednich wyników. Dobrym udonieniem naszej gry byłoby zapisanie rekordowego wyniku w pliku i załadowanie go w momencie rozpoczęcia gry.

Wykrywanie zakończenia gry

Gra kończy się, gdy gracz zderzy się z zabójczym pomidorem. Sytuację tę można wykryć w metodzie `update` obiektu reprezentującego pomidor:

```
def update(self):
    'Tutaj znajdzie się kod aktualizacji pozycji dla pomidora'
    if self.intersects_with(game.cheese_sprite):
        self.game.end_game()
```

Aby gra stała się bardziej interesująca, możemy dodać do niej więcej logiki. Moglibyśmy wprowadzić dla gracza wartość opisującą jego kondycję, która zmniejszałaby się przy każdym zderzeniu z pomidorem. Moglibyśmy sprawić, by ta kondycja z czasem się odbudowywała. Moglibyśmy nawet dodać tradycyjne „trzy życia”, które często występują w grach tego rodzaju.

KĄCIK PROGRAMISTY

Zawsze staraj się tworzyć gry, w które da się grać

Inną rzeczą, którą zauważałem podczas oceniania konkursów tworzenia gier, było to, że niektóre zespoły tworzyły świetny fragment rozgrywki, ale nie dołączały jej do gry. Zaczynałeś grać w tę grę i odkrywałeś, że ona nigdy się nie kończy. Upewnij się, że gra jest kompletna. Gra powinna zawierać początek, środek i zakończenie. Jak przekonalismy się w tym podrozdziale, nie jest to trudne, ale kiedy programiści zaczynają budować grę, wydaje się, że na ostatnią chwilę odkładają zadanie napisania kodu rozpoczęcia i końca gry. W efekcie to, co stworzą, przypomina bardziej techniczne demo niż grę, a to nie jest dokładnie to samo. Dzięki zadaniu, by gra była w pełni grą od samego początku, ułatwiasz użytkownikom testowanie jej oraz przekazywanie komentarzy.

Punktacja gry

Za każdym razem gdy ser zderzy się z krakersem, następuje zwiększenie wartości wyniku gry. Wynik jest aktualizowany w metodzie `update` obiektu reprezentującego krakersa:

```
def update(self):
    if self.intersects_with(game.cheese_sprite):
        self.captured_sound.play()
        self.reset()
        self.game.score += 10
```

Aktualizacja wyniku gry

Wynik jest wyświetlany na ekranie za każdym razem, gdy jest rysowany ekran gry. Służy do tego metoda `draw_game`.

```
def draw_game(self):
    for sprite in self.sprites:
        sprite.draw()
    status = 'Wynik: ' + str(game.score)
    self.display_message(status, 0)
```

Narysowanie wszystkich elementów gry

Utworzenie komunikatu z wynikiem

Wyświetlenie wyniku w górnej części ekranu

Kompletną grę można znaleźć w folderze *EG16-11 Kompletna gra*. Fajnie jest grać w krótkich seriach, szczególnie jeśli chce się pobić swój rekord. Mój dotychczasowy rekord to 380, ale nigdy nie byłem dobry w grach wideo.



ZRÓB TO SAM: WYZWANIE PROGRAMISTYCZNE

Stwórz własną grę

Gra Pościg za krakersami może być używana jako podstawa każdej gry planszowej z „duszkami”. Możesz zmienić grafikę, stworzyć nowe typy wrogów, przystosować grę dla dwóch graczy lub dodać atrakcyjne efekty dźwiękowe. Kiedy na początku tej książki powiedziałem, że programowanie jest najbardziej kreatywną rzeczą, jakiej możesz się nauczyć, to mówiłem właśnie o tym. Możesz stworzyć grę o nazwie *Szalona szafa*, w której będziesz ścigany przez wieszaki podczas szukania skarpetki do pary. Możesz stworzyć grę *Ratunek dla kosmicznego morsa*, w której będziesz sterować międzyplanetarnym morsem, aby przeprowadzić go przez pole minowe asteroid. Możesz zbudować wszystko, co zdołasz wymyślić. Jednak mam jedną przestrogę. Nie staraj się zrealizować zbyt wielu pomysłów. Widziałem wiele zespołów deweloperskich, których członkowie frustrowali się z powodu braku możliwości zrealizowania wszystkich swoich pomysłów na raz. O wiele bardziej sensowne jest stworzenie czegoś prostego, co działa, a następnie stopniowe dodawanie nowych elementów.

Czego się nauczyłeś?

W tym rozdziale stworzyłeś prawdziwą grę i odkryłeś, w jaki sposób framework `pygame` umożliwia pracę z grafiką i dźwiękiem. Dowiedziałeś się, że hierarchia klas, z klasą bazową `Sprite` i różnymi obiektami gry zaimplementowanymi jako podklasy tej klasy bazowej, to świetny sposób na implementację postaci w grze. Odkryłeś również, że gry działają na zasadzie „pętli gry”, która wielokrotnie aktualizuje i rysuje obiekty na ekranie. Użyłeś mechanizmu zdarzeń frameworka `pygame` do przechwytywania wejścia z klawiatury i wykorzystałeś zdarzenia do sterowania obiektami na ekranie. Przekonałeś się, że „sztuczną inteligencję” można wprowadzić za pomocą kilku warunków, a „fizykę” zaimplementować za pomocą kilku obliczeń. Zaimplementowałeś także ekran startowy i ekran gry, aby zapewnić graczom wrażenie „kompletności”.

Mam nadzieję, że wpadłeś również na kilka własnych pomysłów i wykorzystałeś je do stworzenia większej liczby gier.

Oto kilka pytań na temat tworzenia gier, na które warto znać odpowiedzi.

Czy wszystkie gry działają w pętli gry?

Większość gier działa w ten sposób. Tekstowa gra przygodowa działa poprzez czytanie tekstu, który wpisujesz, i udzielanie odpowiedzi, ale większość współczesnych gier działa w pętli.

Dlaczego metody odpowiedzialne za rysowanie i aktualizację ekranu są odrębne?

Można się zastanawiać, dlaczego w grze wydzieliłem zachowania związane z rysowaniem i aktualizacją. Chociaż są to odrębne metody, zawsze wydają się być wywoływanie razem. Dlaczego nie ma jednej metody (np. `do_game`), która realizuje oba te zadania?

Problem dotyczy wydajności. W przypadku prostych gier, takich jak *Pościg za krakersami*, rysowanie i aktualizowanie w tym samym rytmie sprawdza się doskonale. Jeśli jednak korzystasz z platformy o niskiej wydajności, możesz chcieć aktualizować grę z inną częstotliwością niż rysować. Powodem tego stanu rzeczy jest fakt, że użytkownicy znacznie bardziej tolerują „migotanie” ekranu gry niż zmiany szybkości jej aktualizacji. Spowolnienie aktualizacji ekranu gry może spowodować problemy z wykrywaniem kolizji (np. pociski przechodzą przez obiekty, a gra tego nie zauważa). Z tego powodu w grze należy oddzielić rysowanie od aktualizacji, aby w razie potrzeby można było uruchomić te dwa procesy z różnymi prędkościami.

Jak utworzyć tryb „zachęcania” w mojej grze?

Obecnie nasza gra ma tylko dwa stany: ekran startowy i ekran gry. Wiele gier ma również ekran „trybu zachęcania”, który wyświetla przykładową rozgrywkę. Tworzenie takiego ekranu jest dość łatwe. Moglibyśmy stworzyć „gracza AI”, który losowo przesuwałby ser po ekranie, a następnie po prostu uruchomić grę z tym losowym graczem. Moglibyśmy dodać zachowanie „trybu zachęcania” do obiektów reprezentujących pomidory, tak aby poruszały się w pewnej odległości od gracza, dzięki czemu w trybie demonstracyjnym gra trwałaaby dłużej.

W jaki sposób sprawić, aby w każdej sesji rozgrywka przebiegała tak samo?

Do obliczania pozycji krakersów w grze użyto generatora liczb losowych Pythona, co oznacza, że przy każdym uruchomieniu gry krakersy znajdują się w innym położeniu. Możemy użyć funkcji `seed` z modułu `random`, aby przekazać do generatora liczb losowych Pythona to samo „ziarno” przed każdą grą. W takiej sytuacji przy każdym uruchomieniu gry krakersy byłyby rysowane i „przywracane do życia” w tej samej kolejności. Zdeterminowany gracz mógłby nauczyć się wzorca i wykorzystać tę wiedzę w celu osiągnięcia dobrego wyniku.

Czy autor gry zawsze osiąga najlepsze wyniki w tej grze?

Najczęściej nie. Często jestem zaskoczony, że inni gracze osiągają znacznie lepsze wyniki w grze w gry, które stworzyłem. Czasami nawet próbują mi pomóc podpowiedziami i wskazówkami na temat tego, co powiniensem zrobić, żeby uzyskać lepszy rezultat.