ship.py

Файл ship.py содержит класс `Ship`. В этом классе определен метод `__init__()`, метод `update()` для управления позицией корабля и метод `blitme()` для вывода изображения корабля на экран. Изображение корабля хранится в файле ship.bmp, который находится в папке images.

УПРАЖНЕНИЯ

12-3. Ракета: создайте игру, у которой в исходном состоянии в центре экрана находится ракета. Игрок может перемещать ракету вверх, вниз, вправо и влево четырьмя клавишами со стрелками. Проследите за тем, чтобы ракета не выходила за края экрана.

12-4. Клавиши: создайте файл Pygame, который создает пустой экран. В цикле событий выводите значение атрибута event.key при обнаружении события pygame.KEYDOWN. Запустите программу, нажимайте различные клавиши и понаблюдайте за реакцией Pygame.

Стрельба

А теперь добавим в игру возможность стрельбы. Мы напишем код, который выпускает пулю (маленький прямоугольник) при нажатии игроком клавиши «пробел». Пули летят вертикально вверх, пока не исчезнут у верхнего края экрана.

Добавление настроек

Сначала добавим в settings.py новые настройки для значений, управляющих поведением класса `Bullet`. Эти настройки добавляются в конец метода `__init__()`:

settings.py

```
      def __init__(self):
        ...
        # Параметры пули
        self.bullet_speed_factor = 1
        self.bullet_width = 3
        self.bullet_height = 15
        self.bullet_color = 60, 60, 60
```

Эти настройки создают темно-серые пули с шириной 3 пиксела и высотой 15 пикселов. Пули двигаются немного медленнее, чем корабль.

Создание класса Bullet

Теперь создадим файл bullet.py для хранения класса `Bullet`. Первая часть файла bullet.py выглядит так:

bullet.py

```
import pygame
from pygame.sprite import Sprite

class Bullet(Sprite):
```

```
        """Класс для управления пулями, выпущенными кораблем."""

        def __init__(self, ai_settings, screen, ship):
            """Создает объект пули в текущей позиции корабля."""
            super(Bullet, self).__init__()
            self.screen = screen

            # Создание пули в позиции (0,0) и назначение правильной позиции.
❶           self.rect = pygame.Rect(0, 0, ai_settings.bullet_width,
                ai_settings.bullet_height)
❷           self.rect.centerx = ship.rect.centerx
❸           self.rect.top = ship.rect.top

            # Позиция пули хранится в вещественном формате.
❹           self.y = float(self.rect.y)

❺           self.color = ai_settings.bullet_color
            self.speed_factor = ai_settings.bullet_speed_factor
```

Класс `Bullet` наследует от класса `Sprite`, импортируемого из модуля `pygame.sprite`. Работая со *спрайтами* (sprite), разработчик группирует связанные элементы в своей игре и выполняет операцию со всеми сгруппированными элементами одновременно. Чтобы создать экземпляр пули, методу `__init__()` необходимо передать экземпляры `ai_settings`, `screen` и `ship`, а вызов `super()` необходим для правильной реализации наследования от `Sprite`.

ПРИМЕЧАНИЕ

Вызов super(Bullet, self).__init__() использует синтаксис Python 2.7. В Python 3 этот синтаксис тоже работает, хотя вызов также можно записать в более простой форме super().__init__().

В точке ❶ создается атрибут `rect` пули. Пуля не создается на основе готового изображения, поэтому прямоугольник приходится строить «с нуля» при помощи класса `pygame.Rect()`. При создании экземпляра этого класса необходимо задать координаты левого верхнего угла прямоугольника, его ширину и высоту. Прямоугольник инициализируется в точке (0, 0), но в следующих двух строках он перемещается в нужное место, так как позиция пули зависит от позиции корабля. Ширина и высота пули определяются значениями, хранящимися в `ai_settings`.

В точке ❷ атрибуту `centerx` пули присваивается значение `rect.centerx` корабля. Пуля должна появляться y верхнего края корабля, поэтому верхний край пули совмещается с верхним краем прямоугольника корабля для имитации «выстрела» из корабля ❸.

Координата y пули хранится в вещественной форме для внесения более точных изменений в скорость пули ❹. В точке ❺ настройки цвета и скорости пули сохраняются в `self.color` и `self.speed_factor`.

А вот как выглядит вторая часть bullet.py, `update()` и `draw_bullet()`:

bullet.py

```
    def update(self):
        """Перемещает пулю вверх по экрану."""
```

```
            # Обновление позиции пули в вещественном формате.
❶           self.y -= self.speed_factor
            # Обновление позиции прямоугольника.
❷           self.rect.y = self.y
        def draw_bullet(self):
            """Вывод пули на экран."""
❸           pygame.draw.rect(self.screen, self.color, self.rect)
```

Метод `update()` управляет позицией пули. Когда происходит выстрел, пуля двигается вверх по экрану, что соответствует уменьшению координаты *y*; следовательно, для обновления позиции пули следует вычесть величину, хранящуюся в `self.speed_factor`, из `self.y` ❶. Затем значение `self.y` используется для изменения значения `self.rect.y` ❷. Атрибут `speed_factor` позволяет увеличить скорость пуль по ходу игры или при изменении ее поведения. Координата *x* пули после выстрела не изменяется, поэтому пуля летит вертикально по прямой линии.

Для вывода пули на экран вызывается функция `draw_bullet()`. Она заполняет часть экрана, определяемую прямоугольником пули, цветом из `self.color` ❸.

Группировка пуль

Класс `Bullet` и все необходимые настройки готовы; можно переходить к написанию кода, который будет выпускать пулю каждый раз, когда игрок нажимает клавишу «пробел». Сначала мы создадим в alien_invasion.py группу для хранения всех летящих пуль, чтобы программа могла управлять их полетом. Эта группа будет представлена экземпляром класса `pygame.sprite.Group`, который напоминает список с расширенной функциональностью, которая может быть полезна при построении игр. Мы воспользуемся группой для прорисовки пуль на экране при каждом проходе основного цикла и обновления текущей позиции каждой пули:

alien_invasion.py

```
import pygame
from pygame.sprite import Group
...

def run_game():
    ...
    # Создание корабля.
    ship = Ship(ai_settings, screen)
    # Создание группы для хранения пуль.
❶   bullets = Group()

    # Запуск основного цикла игры.
    while True:
        gf.check_events(ai_settings, screen, ship, bullets)
        ship.update()
❷       bullets.update()
        gf.update_screen(ai_settings, screen, ship, bullets)

run_game()
```

Класс `Group` импортируется из `pygame.sprite`. В точке ❶ создается экземпляр `Group` с именем `bullets`. Эта группа создается за пределами цикла `while`, чтобы новая группа пуль не создавалась при каждом проходе цикла.

ПРИМЕЧАНИЕ

Если группа будет создаваться в цикле, в результате программа создает тысячи групп, и скорость игры упадет до минимума. Если ваша игра со временем начинает заметно «тормозить», внимательно проверьте, что происходит в основном цикле while.

Объект `bullets` передается методам `check_events()` и `update_screen()`. В `check_events()` он используется при обработке клавиши «пробел», а в `update_screen()` необходимо перерисовать выводимые на экран пули.

Вызов `update()` для группы ❷ приводит к автоматическому вызову `update()` для каждого спрайта в группе. Строка `bullets.update()` вызывает `bullet.update()` для каждой пули, включенной в группу `bullets`.

Обработка выстрелов

В файле game_functions.py необходимо внести изменения в метод `check_keydown_events()`, чтобы при нажатии клавиши «пробел» происходил выстрел. Изменять `check_keyup_events()` не нужно, потому что при отпускании клавиши ничего не происходит. Также необходимо изменить `update_screen()` и вывести каждую пулю на экран перед вызовом `flip()`. Ниже представлены соответствующие изменения в game_functions.py:

game_functions.py

```
...
from bullet import Bullet
❶ def check_keydown_events(event, ai_settings, screen, ship, bullets):
    ...
❷   elif event.key == pygame.K_SPACE:
        # Создание новой пули и включение ее в группу bullets.
        new_bullet = Bullet(ai_settings, screen, ship)
        bullets.add(new_bullet)
    ...

❸ def check_events(ai_settings, screen, ship, bullets):
    """Обрабатывает нажатия клавиш и события мыши."""
    for event in pygame.event.get():
        ...
    elif event.type == pygame.KEYDOWN:
            check_keydown_events(event, ai_settings, screen, ship, bullets)
    ...

❹ def update_screen(ai_settings, screen, ship, bullets):
    ...
    # Все пули выводятся позади изображений корабля и пришельцев.
❺   for bullet in bullets.sprites():
        bullet.draw_bullet()
    ship.blitme()
    ...
```

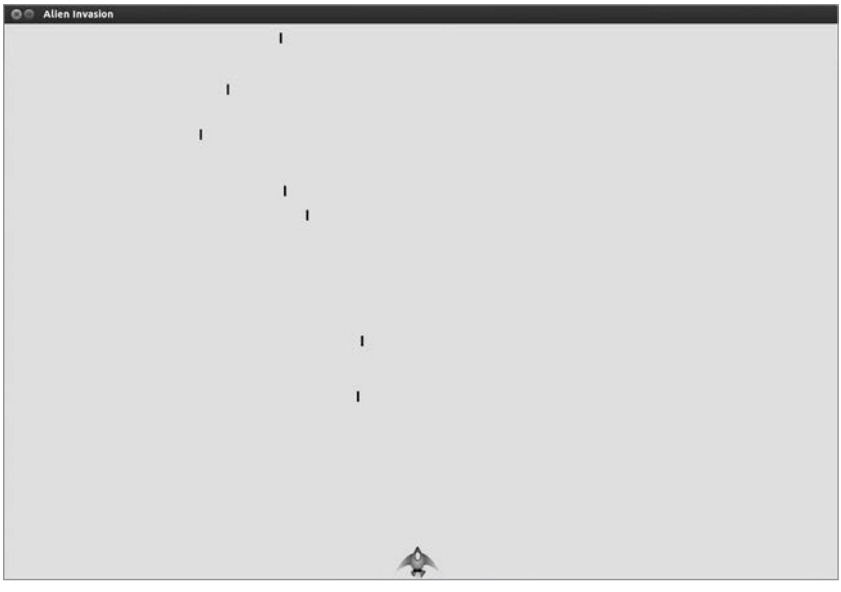

Рис. 12.3. Экран игры после серии выстрелов

Группа `bullets` передается `check_keydown_events()` ❶. Когда игрок нажимает пробел, создается новая пуля (экземпляр `Bullet` с именем `new_bullet`), которая добавляется в группу `bullets` ❷ методом `add()`; код `bullets.add(new_bullet)` сохраняет новую пулю в группе `bullets`.

Группу `bullets` необходимо добавить в число параметров в определении `check_events()` ❸, а также передать в аргументе при вызове `check_keydown_events()`.

Параметр `bullets` передается функции `update_screen()` ❹, которая рисует пули на экране. Метод `bullets.sprites()` возвращает список всех спрайтов в группе `bullets`. Чтобы нарисовать все выпущенные пули на экране, программа перебирает спрайты в `bullets` и вызывает для каждого `draw_bullet()` ❺.

Если запустить alien_invasion.py сейчас, вы сможете двигать корабль влево и вправо и выпускать сколько угодно пуль. Пули перемещаются вверх по экрану и исчезают при достижении верхнего края (рис. 12.3). Размер, цвет и скорость пуль можно изменить при помощи настроек в settings.py.

Удаление старых пуль

На данный момент пули исчезают по достижении верхнего края, но только потому, что Pygame не может нарисовать их выше края экрана. На самом деле пули продолжают существовать; их координата y продолжает уменьшаться. И это создает проблему, потому что пули продолжают потреблять память и вычислительные мощности.

От старых пуль необходимо избавиться, иначе игра замедлится из-за большого объема лишней работы. Для этого необходимо определить момент, когда атрибут `bottom` прямоугольника пули достигнет 0, — это означает, что пуля вышла за верхний край экрана:

alien_invasion.py

```
    # Запуск основного цикла игры.
    while True:
        gf.check_events(ai_settings, screen, ship, bullets)
        ship.update()
        bullets.update()

        # Удаление пуль, вышедших за край экрана.
❶       for bullet in bullets.copy():
❷           if bullet.rect.bottom <= 0:
❸               bullets.remove(bullet)
❹       print(len(bullets))

        gf.update_screen(ai_settings, screen, ship, bullets)
```

Удалять элементы из списка или группы в цикле `for` не следует, поэтому перебирать нужно копию группы. Метод `copy()` используется для создания цикла `for` ❶, в котором возможно изменять содержимое `bullets`. Программа проверяет каждую пулю и определяет, вышла ли она за верхний край экрана ❷. Если пуля пересекла границу, она удаляется из `bullets` ❸. В точке ❹ добавляется команда `print`, которая сообщает, сколько пуль сейчас существует в игре; по выведенному значению можно убедиться в том, что пули действительно были удалены.

Если код работает правильно, вы можете понаблюдать за выводом на терминале и убедиться в том, что количество пуль уменьшается до 0 после того, как очередной залп уходит за верхний край экрана. После того как вы запустите игру и убедитесь в том, что пули правильно удаляются из группы, удалите команду `print`. Если команда останется в программе, она существенно замедлит игру, потому что вывод на терминал занимает больше времени, чем отображение графики в игровом окне.

Ограничение количества пуль

Многие игры-«стрелялки» ограничивают количество пуль, одновременно находящихся на экране, чтобы у игроков появился стимул стрелять более метко. То же самое будет сделано и в игре Alien Invasion.

Сначала сохраним максимально допустимое количество пуль в settings.py:

settings.py

```
# Параметры пули
self.bullet_width = 3
self.bullet_height = 15
self.bullet_color = 60, 60, 60
self.bullets_allowed = 3
```

В любой момент времени на экране может находиться не более трех пуль. Эта настройка будет использоваться в game_functions.py для проверки количества существующих пуль перед созданием новой пули в `check_keydown_events()`:

game_functions.py

```
def check_keydown_events(event, ai_settings, screen, ship, bullets):
    ...
    elif event.key == pygame.K_SPACE:
        # Создание новой пули и включение ее в группу bullets.
        if len(bullets) < ai_settings.bullets_allowed:
            new_bullet = Bullet(ai_settings, screen, ship)
            bullets.add(new_bullet)
```

При нажатии клавиши «пробел» программа проверяет длину `bullets`. Если значение `len(bullets)` меньше трех, создается новая пуля. Но, если на экране уже находятся три активные пули, при нажатии пробела ничего не происходит. Если вы запустите игру сейчас, вы сможете выпускать пули только группами по три.

Создание функции update_bullets()

Мы хотим, чтобы главный файл программы alien_invasion.py был как можно более простым, поэтому после написания и проверки кода управления пулями этот код можно переместить в модуль `game_functions`. Мы создадим новую функцию `update_bullets()` и добавим ее в конец game_functions.py:

game_functions.py

```
def update_bullets(bullets):
    """Обновляет позиции пуль и уничтожает старые пули."""
    # Обновление позиций пуль.
    bullets.update()

    # Удаление пуль, вышедших за край экрана.
    for bullet in bullets.copy():
        if bullet.rect.bottom <= 0:
            bullets.remove(bullet)
```

Код `update_bullets()` вырезается и вставляется из alien_invasion.py; единственным необходимым параметром функции является группа `bullets`.

Цикл `while` в alien_invasion.py снова выглядит просто:

alien_invasion.py

```
    # Запуск основного цикла игры.
    while True:
❶       gf.check_events(ai_settings, screen, ship, bullets)
❷       ship.update()
❸       gf.update_bullets(bullets)
❹       gf.update_screen(ai_settings, screen, ship, bullets)
```

В результате преобразования основной цикл содержит минимум кода, чтобы можно было легко прочитать имена функций и понять, что происходит в игре. Основной цикл проверяет ввод, полученный от игрока ❶, а затем обновляет позицию корабля ❷ и всех выпущенных пуль ❸. Затем обновленные позиции игровых элементов используются для вывода нового экрана в точке ❹.

Создание функции fire_bullet()

Переместим код стрельбы в отдельную функцию, чтобы выстрел выполнялся всего одной строкой кода, а блок `elif` в `check_keydown_events()` оставался простым:

game_functions.py

```
def check_keydown_events(event, ai_settings, screen, ship, bullets):
    """Реагирует на нажатия клавиш."""
    ...
    elif event.key == pygame.K_SPACE:
        fire_bullet(ai_settings, screen, ship, bullets)

def fire_bullet(ai_settings, screen, ship, bullets):
    """Выпускает пулю, если максимум еще не достигнут."""
    # Создание новой пули и включение ее в группу bullets.
    if len(bullets) < ai_settings.bullets_allowed:
    new_bullet = Bullet(ai_settings, screen, ship)
    bullets.add(new_bullet)
```

Функция `fire_bullet()` просто содержит код, который использовался для выстрела при нажатии клавиши «пробел»; вызов `fire_bullet()` добавляется в `check_keydown_events()` при нажатии клавиши «пробел».

Запустите alien_invasion.py еще раз и убедитесь в том, что стрельба проходит без ошибок.

УПРАЖНЕНИЯ

12-5. Боковая стрельба: напишите игру, в которой корабль размещается у левого края экрана, а игрок может перемещать корабль вверх и вниз. При нажатии клавиши «пробел» корабль стреляет, и пуля двигается вправо по экрану. Проследите за тем, чтобы пули удалялись при выходе за край экрана.

Итоги

В этой главе вы научились составлять план игры, а также усвоили базовую структуру игры, написанной с использованием Pygame. Вы узнали, как задать цвет фона и как сохранить настройки в отдельном классе, чтобы они были доступны для всех частей игры. Вы научились выводить изображения на экран и управлять перемещением игровых элементов. Также вы узнали, как создавать элементы, двигающиеся самостоятельно (например, пули, летящие по экрану), и как удалять объекты, которые стали лишними. Также в этой главе рассматривалась методика регулярного рефакторинга кода для упрощения текущей разработки.

В главе 13 в игру Alien Invasion будут добавлены пришельцы. К концу главы 13 игрок сможет сбивать корабли пришельцев — конечно, если они не доберутся до него первыми!