**Аннотация**

*На последнем занятии мы рассмотрим два способа на порядок улучшить впечатление от игры (примерно в 10 раз).*

*Этот урок не содержит задач.*

**Анимация спрайтов**

В игре важно все: и удобство управления, и тщательно разработанные уровни. Но в первую очередь пользователи обращают внимание на графические эффекты и звук.

Для того чтобы картинка ожила, используют анимацию. Просто рисуют несколько изображений и последовательно их меняют. Обычно для анимации спрайтов берут листы, совмещающие в себе несколько последовательных изображений. Например, такие:



Удобно создать специальный класс «анимированный спрайт», где разрезать лист на отдельные спрайты.

class AnimatedSprite(pygame.sprite.Sprite):

def \_\_init\_\_(self, sheet, columns, rows, x, y):

super().\_\_init\_\_(all\_sprites)

self.frames = []

self.cut\_sheet(sheet, columns, rows)

self.cur\_frame = 0

self.image = self.frames[self.cur\_frame]

self.rect = self.rect.move(x, y)

def cut\_sheet(self, sheet, columns, rows):

self.rect = pygame.Rect(0, 0, sheet.get\_width() // columns,

sheet.get\_height() // rows)

for j in range(rows):

for i in range(columns):

frame\_location = (self.rect.w \* i, self.rect.h \* j)

self.frames.append(sheet.subsurface(pygame.Rect(

frame\_location, self.rect.size)))

def update(self):

self.cur\_frame = (self.cur\_frame + 1) % len(self.frames)

self.image = self.frames[self.cur\_frame]

Смены кадров удобно делать в методе update(). По времени, чтобы элементы игры оживали, или по координатам, чтобы, например, герой ходил, переставляя ноги. Реализация выше — лишь один из возможных примеров.

Смена кадра на каждой итерации цикла — это очень часто.

На рисунке показана работа спрайта, созданного командой

dragon = AnimatedSprite(load\_image("dragon\_sheet8x2.png"), 8, 2, 50, 50)



при обновлении 10 раз в секунду (clock.tick(10)).

Для того чтобы не тормозить всю игру, можно ввести в классе счетчик итераций и менять изображение, скажем, каждую пятую итерацию.

Сложнее всего — нарисовать персонажа. Для первых игр вполне можно использовать [свободно распространяемые сеты](https://yandex.ru/search/?text=free+sprite+sheets&lr=213&clid=2270455&win=413&suggest_reqid=429699388157469083249185062493782,). Обычно в них содержится несколько анимаций, например, движение игрока в разные стороны. Можно создать несколько списков с индексами изображений и переключать их в зависимости от направления движения.

**Система частиц**

Система частиц очень украшает игру. Взрывы, туман, искры — все эти эффекты делаются по одной технологии. В ее основе, как в жизни, много маленьких объектов, которые связаны со своим источником (эммитером) и иногда между собой.

Удобно оформлять частицу в виде класса, наследника спрайта. При этом можно просто проверять вылет частиц за границы экрана пересечением прямоугольника спрайта с прямоугольником экрана.

...

# для отслеживания улетевших частиц

# удобно использовать пересечение прямоугольников

screen\_rect = (0, 0, width, height)

class Particle(pygame.sprite.Sprite):

# сгенерируем частицы разного размера

fire = [load\_image("star.png")]

for scale in (5, 10, 20):

fire.append(pygame.transform.scale(fire[0], (scale, scale)))

def \_\_init\_\_(self, pos, dx, dy):

super().\_\_init\_\_(all\_sprites)

self.image = random.choice(self.fire)

self.rect = self.image.get\_rect()

# у каждой частицы своя скорость — это вектор

self.velocity = [dx, dy]

# и свои координаты

self.rect.x, self.rect.y = pos

# гравитация будет одинаковой (значение константы)

self.gravity = GRAVITY

def update(self):

# применяем гравитационный эффект:

# движение с ускорением под действием гравитации

self.velocity[1] += self.gravity

# перемещаем частицу

self.rect.x += self.velocity[0]

self.rect.y += self.velocity[1]

# убиваем, если частица ушла за экран

if not self.rect.colliderect(screen\_rect):

self.kill()

Для демонстрационного примера не будем создавать класс эмиттера, а ограничимся функцией, принимающей в себя позицию, из которой будут вылетать звезды:

def create\_particles(position):

# количество создаваемых частиц

particle\_count = 20

# возможные скорости

numbers = range(-5, 6)

for \_ in range(particle\_count):

Particle(position, random.choice(numbers), random.choice(numbers))

Эту функцию будем вызывать по щелчку мыши, передавая в нее координаты нажатия клавиши мыши:

all\_sprites = pygame.sprite.Group()

clock = pygame.time.Clock()

running = True

while running:

for event in pygame.event.get():

if event.type == pygame.QUIT:

running = False

if event.type == pygame.MOUSEBUTTONDOWN:

# создаём частицы по щелчку мыши

create\_particles(pygame.mouse.get\_pos())

all\_sprites.update()

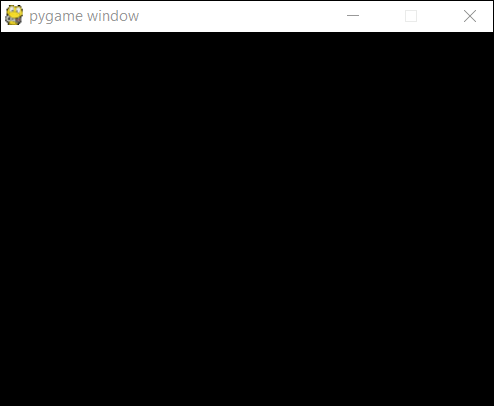
screen.fill((0, 0, 0))

all\_sprites.draw(screen)

pygame.display.flip()

clock.tick(50)

pygame.quit()



Картинку для задания можно взять тут:



Конечно, это всего лишь пример. Впереди — большое поле для экспериментов и изучения теории. Существует много сложных алгоритмов систем частиц, с которыми обязательно стоит познакомиться.

**Что же дальше?**

К этому занятию нет специальных задач. Разберитесь с приведенными в уроке примерами и продолжайте разрабатывать свою игру.

У хорошей игры должно быть звуковое оформление. Мы не обсуждали звук на занятиях, но попробуйте разобраться по документации с возможностями модуля [pygame.mixer](https://www.pygame.org/docs/ref/mixer.html) и добавьте в игру музыку и эффекты.