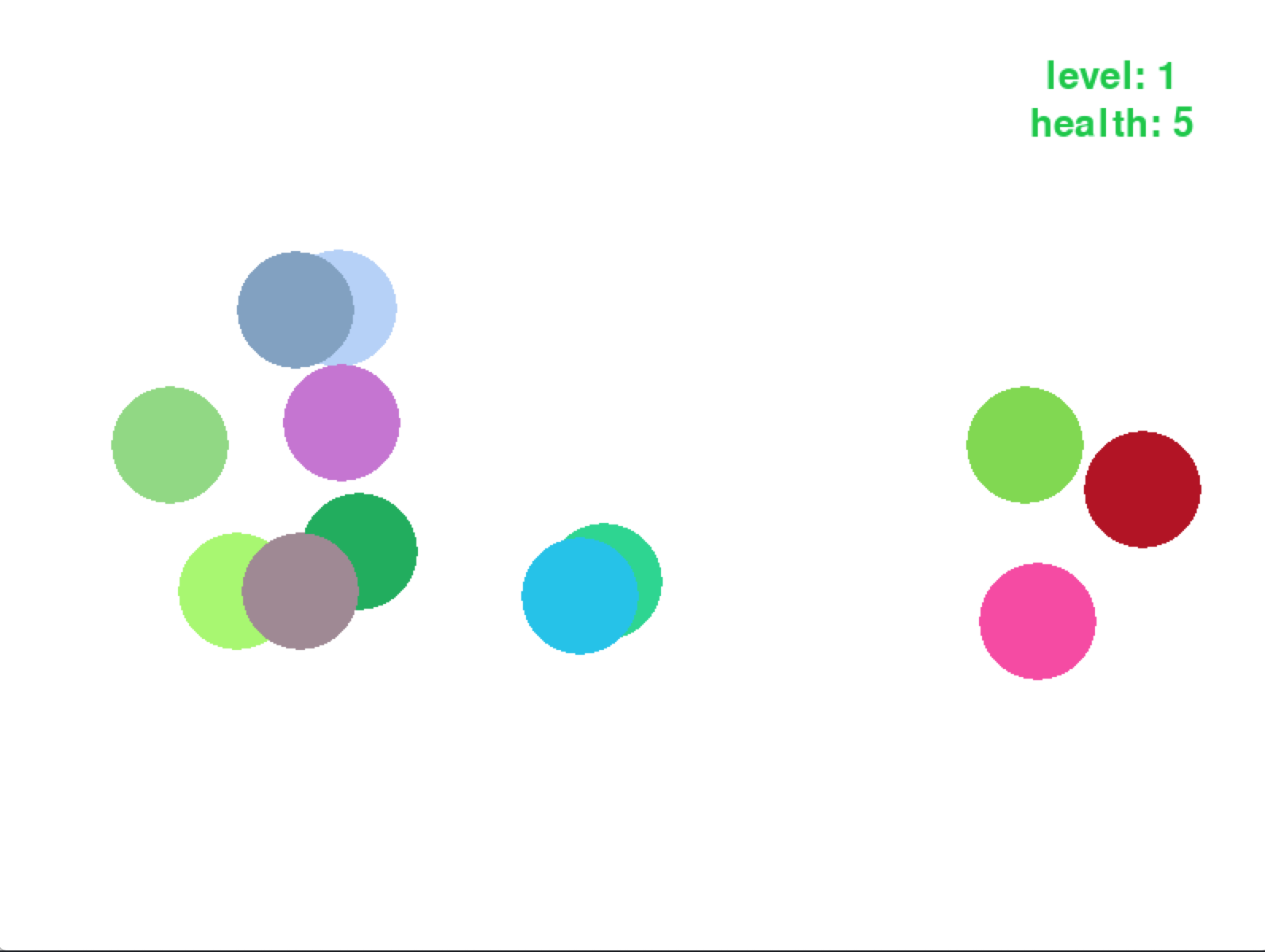
Игра 3: Конфетти



# 

# Повторение

Если вы уже уверенно создаете стартовые модули для игры, можете пропустить этот шаг. В нем будут описаны схожие действия с представленными в прошлой игре.

## Импорт, инициализация, экран и цвета

* Импорт.

Для нашей игры мы снова импортируем основную библиотеку Pygame и сразу добавим к ней модуль random:

|  |
| --- |
| import pygame import random |

* Инициализируем модуль pygame:

|  |
| --- |
| pygame.init() |

* Создадим окно игры с нужными нам параметрами:

|  |
| --- |
| SCREEN\_WIDTH = 800 SCREEN\_HEIGHT = 600  screen = pygame.display.set\_mode([SCREEN\_WIDTH, SCREEN\_HEIGHT]) |

* Добавим три переменные для хранения цветов нашей игры:

|  |
| --- |
| WHITE = (255, 255, 255) BLACK = (0, 0, 0) GREEN = (0, 200, 64) |

## 

## Главный цикл

Вновь используем переменную-переключатель цикла. Присвоим running значение True. Пока running - истина, игровой цикл продолжается. running становится ложью, как только пользователь нажимает на крестик для выхода из игры.

|  |
| --- |
| running = True  while running:  for event in pygame.event.get():  if event.type == pygame.QUIT:  running = False |

## ФПС

Добавим ограничение кадров в секунду для того, чтобы наша игра работала одинаково на компьютерах с разными процессорами. Для этого используем модуль Clock() внутри time в pygame:

|  |
| --- |
| clock = pygame.time.Clock() running = 1  while running:  for event in pygame.event.get():  if event.type == pygame.QUIT:  running = 0  clock.tick(30) |

Оставим стандартное ограничение в 30 кадров.

# Цель игры



**Конфеттѝ**(итал. *confetti*) — разноцветные, обычно бумажные кружочки мелкого размера, неотъемлемый атрибут праздников, в основном, балов, карнавалов, триумфальных шествий, но также дней рождения и свадебных торжеств. Конфетти осыпают друг друга участники празднеств или его сбрасывают сверху.

* Давайте сегодня устроим небольшой праздник и воссоздадим конфетти программным путем.
* Наша игра будет очень простой: снизу экрана будут появляться кружочки-конфетти и лететь вверх, наша цель - не дать им скрыться за верхней гранью экрана, мы будем лопать эти кружочки с помощью нажатия мыши!
* Каждый новый уровень конфетти будут становиться меньше и быстрее, а их количество - возрастать.

# Реализация

## План

Итак, сейчас перед нами стоит самая сложная задача проекта - построить логическую модель игры.

Самый первый и самый правильный вопрос, который возникает - это “А как же мы сделаем много кружочков-конфетти”? Действительно, мы умеем создавать, например, круг и три прямоугольника, но чтобы сделать их 10 или 20, или 100, нам придется 100 раз прописать параметры каждого кружочка, а это нарушает принцип программирования DRY - Don’t Repeat Yourself. Для решения этой задачи нам понадобятся классы.

### 

### Классы

Всё в Пайтоне является объектами. Это очень расплывчатое утверждение, если до этого вы не изучали программирование вообще. Это означает, что каждый объект в Пайтоне имеет метод и значение по той причине, что все объекты базируются на классе. Класс – это проект объекта. Умение управлять классами является неотъемлемой частью владения объектно-ориентированным программированием (ООП). Немного про базовые принципы ООП чуть ниже:

### 

### 

### Базовые принципы ООП

* Абстракция — отделение концепции от ее экземпляра;
* Полиморфизм — реализация задач одной и той же идеи разными способами;
* Наследование — способность объекта или класса базироваться на другом объекте или классе. Это главный механизм для повторного использования кода. Наследственное отношение классов четко определяет их иерархию;
* Инкапсуляция — размещение одного объекта или класса внутри другого для разграничения доступа к ним.

Понимание этих концептов придет со временем, сейчас мы не будем углубляться в их интерпретацию, но после написания этой игры у вас уже появится понимание о некоторых пунктах.

## Пишем код

### Функция put\_text()

|  |
| --- |
| font = pygame.font.Font(None, 36)   def put\_text(message, x, y):  text = font.render(message, True, GREEN)  place = text.get\_rect(center=(x, y))  screen.blit(text, place) |

Все то же самое, как и в прошлый раз, только на этот раз будем отображать цвет зеленого цвета. В функцию подаем сообщение message и координаты x,y. До функции создадим объект font, необходимый для определения шрифта.

### 

### Функция victory()

|  |
| --- |
| def victory():  screen.fill(WHITE)  put\_text("VICTORY!", SCREEN\_WIDTH // 2, SCREEN\_HEIGHT // 2)  pygame.display.flip()  while True:  for ev in pygame.event.get():  if ev.type == pygame.QUIT:  pygame.quit()  exit() |

После объявления победы заполним весь экран белым цветом и отобразим соответствующее сообщение по центру. Запустим новый цикл с возможностью выхода из игры по нажатию на крестик. В вашем праве изменять функции на ваш вкус. Например, было бы удобно предлагать пользователю продолжать игру после того, как она закончилась, подумайте, как это можно реализовать.

### Функция game\_over()

|  |
| --- |
| def game\_over():  screen.fill(WHITE)  put\_text("Game over!", SCREEN\_WIDTH // 2, SCREEN\_HEIGHT // 2)  pygame.display.flip()  while True:  for ev in pygame.event.get():  if ev.type == pygame.QUIT:  pygame.quit()  exit() |

Идентичная функция, которая отвечает за конец игры при проигрыше. Опять же, на ваш вкус изменение функции - подумайте, как, например, объединить две функции: для победы и для проигрыша.

### 

### Дополнительные переменные

|  |
| --- |
| level = 1 lives = 5 ready = False |

* level - отвечает за показатель текущего уровня. Начнем, конечно же, с первого.
* lives - отвечает за счетчик оставшихся жизней. Предлагаю сделать 5 жизней на всю игру.
* ready - определяет готов ли игрок начать следующий раунд. Изначально False (ложь) - изменять будем по нажатию на пробел.

### Класс Bubble

Bubble - это один наш кружок. Класс - это проект объекта. Набор стандартных инструкций и доступных методов для нашего будущего объекта. Чтобы создать этот самый объект нужно будет создать экземпляр класса, в данном случае экземпляр класса будет выглядеть так - Bubble().

Создание класса будет сопровождаться специальным слово Class, в скобках к классу можно указать то, от кого наследуется текущий класс (кто является родителем). В нашем случае нам необходимо наследовать параметры класса Sprite внутри модуля sprite библиотеки pygame.

#### Sprite

**Спрайт** (англ. *sprite* — «фея, эльф») — графический объект в компьютерной графике. В разработке видеоигр очень удобно работать с объектами как с спрайтами, в pygame для этого реализованы очень много важнейших функций.

Чтобы задать инструкцию классу о том, что дальше мы будем работать с ним как со спрайтом, нам понадобится унаследовать стандартный класс, сделать это можно следующим образом:

|  |
| --- |
| class Bubble(pygame.sprite.Sprite): |

#### 

#### 

#### Инициализация

Для инициализации класса всегда используется метод init, это системный метод, поэтому он выделяется двумя нижними подчеркиваниями с каждой стороны.

|  |
| --- |
| def \_\_init\_\_(self):  super().\_\_init\_\_()  self.color = (random.randrange(255), random.randrange(255), random.randrange(255))  self.size = 80 - level \* 5  self.surf = pygame.Surface((self.size, self.size))  self.rect = self.surf.get\_rect()  self.rect.x = random.randint(100, 600)  self.rect.y = random.randint(SCREEN\_HEIGHT + 40, SCREEN\_HEIGHT + 300)  self.vel\_x = random.randint(-7, 7)  self.vel\_y = random.randrange(int(-level \* 1.5), int(-level / 2)) |

Первая строчка вызывает инициализацию у родительского класса - Sprite. Это необходимо для настройки параметров объекта типа sprite. super() обращается к методам родительского класса.

Дальше следует набор параметров. self - это имя нашего объекта.

* self.color - устанавливает параметр color нашему объекту.
* self.size - установим размер нашему кружку. Размер будет уменьшаться на 5 с каждым уровнем.
* self.surf - создает объект поверхности с заданными параметрами размера.
* self.rect - создадим прямоугольную область, исходя из нашей поверхности, для этого используем метод get\_rect() у нашего surf.
* self.rect.x - после определения прямоугольной области мы можем задать координаты ее левого верхнего угла, эти координаты будут у кружка в тот момент, когда он появится - именно тогда вызовется метод init(). Используем метод randint() модуля random, интервал укажем от 100 до 600.
* self.rect.y - укажем высоту, на которой появится кружок. Также используем randint: пусть конфетти появляются ниже нижней грани экрана с интервалом от 40 до 300 пикселей от нее.
* self.vel\_x - зададим скорость нашего кружка по горизонтальной оси, обозначим интервал от -7 до 7.
* self.vel\_y - зададим скорость нашего кружка по вертикальной оси, пусть она будет увеличиваться в зависимости от текущего уровня. Интервал, который предложен в примере кажется сбалансированным. Опять же, на ваше усмотрение поменять какие-либо параметры на ваш вкус.

#### Метод отрисовки

Внутри нашего класса Bubble обозначим метод draw. Метод - это та же функция, только обозначенная внутри класса. Вызывая метод, мы будем передавать туда экземпляр текущего класса, объект self.

|  |
| --- |
| def draw(self):  pygame.draw.circle(screen, self.color,  (self.rect.x + self.size//2, self.rect.y + self.size//2), self.size//2) |

Задача этого метода - отрисовка нашего спрайта в виде круга. Передаем в метод circle() модуля draw в pygame все необходимые параметры:

* screen - поверхность, на которой отображаем наш кружок.
* self.color - цвет объекта.
* следующий аргумент - это кортеж из координат: для того, чтобы круг отобразился в центре нашего спрайта (напомню, что спрайт - это прямоугольная область) сместим на половину размера (стороны) центр круга по оси ‘x’ и по оси ‘y’.
* радиус - это последний аргумент, он будет равен половине стороны нашей прямоугольной (квадратной) области.

#### 

#### Метод move()

Самый важный метод в нашем классе - нужно двигать наши кружочки, и нужно делать это правильно. Добавим еще немного функционала:

* Переменная lives обозначена вне класса Bubble, поэтому мы должны указать, что эта переменная внутри метода move будет изменяться глобально. Для этого напишем первую строчку global lives.
* mouse\_pos - переменная, которая хранит в себе кортеж из координат, на которых сейчас находится курсор мыши, для передачи этих координат используем метод get\_pos() модуля mouse.
* mouse\_clicked - кортеж из нажатых клавиш мыши, вариантов у нас немного, нас интересует только нажатие левой кнопки мыши. С помощью клика левой кнопки мыши, игрок будет лопать наше конфетти. Массив возвращается с помощью метода get\_pressed() модуля mouse.

|  |
| --- |
| def move(self):  global lives  mouse\_pos = pygame.mouse.get\_pos()  mouse\_clicked = pygame.mouse.get\_pressed()  if b.rect.collidepoint(\*mouse\_pos) and mouse\_clicked[0]:  b.kill()  if self.rect.left < 0:  self.vel\_x = abs(self.vel\_x)  if self.rect.right > SCREEN\_WIDTH:  self.vel\_x = -abs(self.vel\_x)  self.rect.move\_ip(self.vel\_x, self.vel\_y)  if self.rect.bottom < 0:  lives -= 1  if lives < 0:  game\_over()  self.kill() |

### Функция generate\_bubbles()

Добавим функцию для создания кружков относительно текущего уровня:

|  |
| --- |
| all\_bubbles = pygame.sprite.Group()   def generate\_bubbles():  for \_ in range(10 + level \* 2):  all\_bubbles.add(Bubble()) |

### Логика игры

Все элементы нашей игры уже добавлены. Давайте разберемся с логикой того, как мы их будем вызывать внутри нашего главного цикла:

1. В самом начале будем проверять, получилось ли у игрока лопнуть все шарики, если да, то меняем ready на False, level увеличиваем на 1 и проверяем, достиг ли игрок победного условия.
2. Ловим ивенты, и по нажатию на крестик закрываем игру, а по нажатию на пробел при условии, что ready = False, генерируем шарики.
3. Отрисовка:
   1. Заполняем экран белым цветом.
   2. Рисуем все шарики из списка all\_bubbles с помощью метода draw(), а после в этом же цикле подвинем их с помощью метода move(), так мы в следующей итерации цикла отрисуем их уже на новой позиции.
4. Вывод текста, для этого будем использовать функцию put\_text():
   1. Если игрок не готов, попросим нажать на пробел для того, чтобы начать уровень.
   2. В правом верхнем углу выведем текущее значение уровня.
   3. В правом верхнем углу чуть ниже выведем текущее значение оставшихся очков здоровья.
5. Вызове flip() и ограничим количество кадров в секунду.

### 

### Главный цикл

Конечный вид нашего цикла примет вид:

|  |
| --- |
| while running:  if ready and len(all\_bubbles) == 0:  ready = 0  level += 1  if level == 7:  victory()  for event in pygame.event.get():  if event.type == pygame.QUIT:  running = False  if event.type == pygame.KEYDOWN:  if event.key == pygame.K\_SPACE:  if not ready:  generate\_bubbles()  ready = True  screen.fill(WHITE)  for b in all\_bubbles:  b.draw()  b.move()  if not ready:  put\_text("Press space to start next level", SCREEN\_WIDTH//2, SCREEN\_HEIGHT//2)  put\_text(("level: " + str(level)), SCREEN\_WIDTH - 100, 50)  put\_text(("health: " + str(lives)), SCREEN\_WIDTH - 100, 80)  pygame.display.flip()  clock.tick(30) pygame.quit() exit() |

### 

# Заключение

Сегодня мы рассмотрели основные концепции для разработки игр - спрайты и классы. Это очень важно, и для каждой следующей игры мы будем использовать оба из наших новых умений.

# 

# Ссылка на проект

Для того, чтобы увидеть мой вариант решения, переходите по прикрепленной ссылке:

<https://drive.google.com/file/d/1-T2u995bYNpUT5KO3zoFAjDP7ovBMCc4/view?usp=sharing>