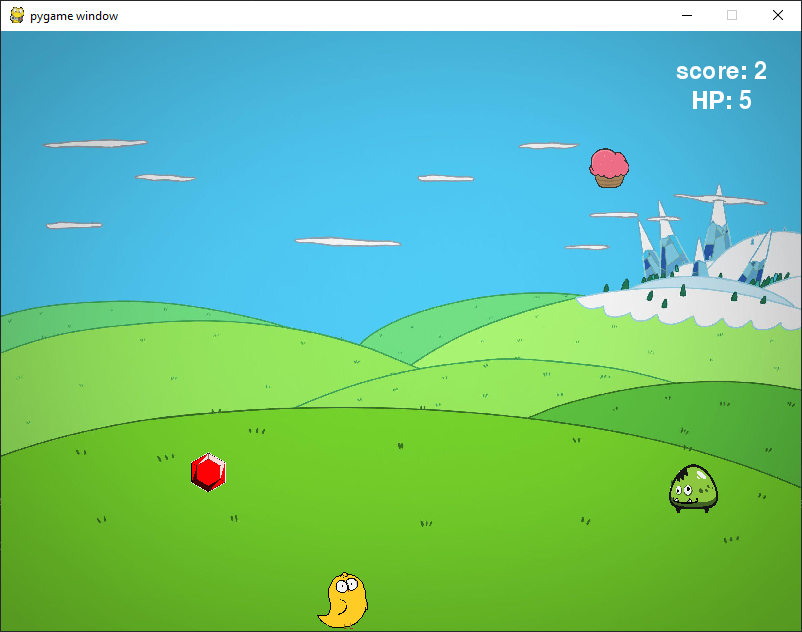
Игра 7: Птенец



# Повторение

Приступим к нашей любимой части написания программы и вспомним все, что уже делали в предыдущих уроках.

## Импорт и инициализация

* Импортируем библиотеки pygame и random:

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

* Инициализируем модуль pygame:

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

## Экран

Как и в прошлые разы создадим объект экрана с заданными параметрами ширины и высоты, в этот раз мы используем стандартный размер экрана:

|  |
| --- |
| SCREEN\_WIDTH = 800 SCREEN\_HEIGHT = 600  screen = pygame.display.set\_mode((SCREEN\_WIDTH, SCREEN\_HEIGHT)) pygame.display.set\_caption('Cheeper') |

В последней строчке мы установили название окна.

## Цвета

Обозначим два цвета в формате RGB, которые будем использовать в этой игре:

|  |
| --- |
| WHITE = (255, 255, 255)  BLACK = (0, 0, 0) |

## 

## 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 = True  while running:  for event in pygame.event.get():  if event.type == pygame.QUIT:  running = False  clock.tick(60) |

Оставим стандартное ограничение в 60 кадров.

## Функция put\_text()

Используем нашу функцию put\_text() из прошлых игр, на этот раз будем писать белым цветом. Добавим возможность передавать цвет текста в функцию, по умолчанию установим белый:

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

## 

## Функция game\_over()

Снова используем функцию game\_over для уведомления о проигрыше игрока:

|  |
| --- |
| def game\_over():  screen.blit(bg, [0, 0])  put\_text('Game over!', SCREEN\_WIDTH // 2, SCREEN\_HEIGHT // 2 + 120)  put\_text('Final score: ' + str(score), SCREEN\_WIDTH//2, SCREEN\_HEIGHT//2+160)  pygame.display.flip()  while True:  for ev in pygame.event.get():  if ev.type == pygame.QUIT:  pygame.quit()  exit() |

В этот раз вместо заливки всего экрана цветом используем функцию blit(), которая позволит поместить фоновое изображение в наше пространство экрана с левым верхним углом в заданных координатах.

Вызываем функцию в тот момент, когда игрок проиграет.

# Цель игры



В этой игре мы будем управлять маленьким птенчиком. Его целью будет уклоняться от падающих с неба монстров и собирать полезные предметы и еду. Птенец очень озабочен долей пойманных предметов, поэтому не допустит потери больше, чем 5. Поможем ему с такой непростой задачей с помощью клавиш перемещения влево и вправо.

# План игры

Вновь не будем упускать этот важнейший этап разработки игры.

## Загрузка изображений

В этот раз мы будем использовать относительно много графики. В первую очередь, следует разобраться с тем, как правильно загрузить все изображения в программу.

Все необходимые файлы можно найти по ссылке [изображения](https://drive.google.com/drive/folders/14Y3DxKarJW7y9jNEcTvoHwAZcIp2dBqt?usp=sharing) - поместите их в корень игры.

## Объекты

А точнее будет сказать группы объектов, классы.

В этой игре нас снова ждет 3 основных группы:

* Птенец - главный персонаж нашей игры, которым мы будем управлять.
* Монстры - враги, от которых мы будем уворачиваться.
* Полезные предметы - их наш птенец будет собирать, чтобы накопить победные очки.

## Процесс игры

Думаю, вы уже представляете то, как будет проходить процесс игры. На всякий случай уточним:

* Клавишами управления будем управлять нашим птенцом.
* Сверху экрана падают монстры, от которых нужно уворачиваться.
* Также сверху экрана падают предметы, которые нужно ловить для получения победных очков.
* Если птенец упустит 5 предметов или наткнется на монстра, то игра закончена.

# Реализация

## Загрузка

### Фоновое изображение

Здесь все очень просто: загружаем изображение с помощью знакомых нам методов и подгоняем его под размер экрана, попутно сохраняя объект в переменную. Получатся такие две простые строчки:

|  |
| --- |
| bg = pygame.image.load("background.jpg").convert() bg = pygame.transform.scale(bg, (SCREEN\_WIDTH, SCREEN\_HEIGHT)) |

### Изображения объектов

В этом случае уже придется слегка подумать. Главное - понять то, что мы хотим получить в итоге. Итак, мы хотим две независимых друг от друга группы объектов загруженных изображений, которые готовы к использованию непосредственно внутри игры.

Для начала создадим эти две группы:

|  |
| --- |
| enemy\_images = [] item\_images = [] |

Списки подойдут.

Если заглянуть в папку с изображениями, можно увидеть, что изображений монстров всего два, с их парсингом (достаточно важно понимать, что означает этот термин) проблем не должно возникнуть, но с предметами нужно поступить немного иначе.

Решим эти две задачи таким способом:

1. Создадим список имен предметов. Имена идентичны названиям соответствующих файлов.
2. В цикле пройдемся по каждому файлу и отформатируем его в подходящий размер и формат, после чего добавим полученный объект в список предметов.
3. Используем идентичный цикл для формирования объектов изображений монстров.

Получим:

|  |
| --- |
| item\_images\_names = ('apple', 'cake', 'cherry', 'pear', 'ruby')  for name in item\_images\_names:  img = pygame.image.load(name + '.png').convert()  if name == 'ruby' or name == 'cake' or name == 'apple':  img.set\_colorkey(BLACK, pygame.RLEACCEL)  else:  img.set\_colorkey(WHITE, pygame.RLEACCEL)  img = pygame.transform.scale(img, (40, 40))  item\_images.append(img)  for i in [1, 2]:  img = pygame.image.load('monster' + str(i) + '.png').convert()  if i == 1:  img.set\_colorkey(WHITE, pygame.RLEACCEL)  img = pygame.transform.scale(img, (50, 50))  else:  img.set\_colorkey(BLACK, pygame.RLEACCEL)  img = pygame.transform.scale(img, (60, 60))  enemy\_images.append(img) |

Сейчас в списках item\_images и enemy\_images лежат объекты, которые мы можем использовать в коде.

## 

## Класс Player

Этот класс будет отвечать за логику главного объекта - птенца.

### Инициализация

|  |
| --- |
| class Player(pygame.sprite.Sprite):  def \_\_init\_\_(self):  super().\_\_init\_\_()  self.image = pygame.image.load("bird.png").convert()  self.image.set\_colorkey(WHITE, pygame.RLEACCEL)  self.rect = self.image.get\_rect()  self.image\_left = self.image  self.image\_right = pygame.transform.flip(self.image, 1, 0)  self.rect.y = SCREEN\_HEIGHT - self.rect.height  self.rect.x = SCREEN\_WIDTH // 2  self.speed = 6 |

Разберемся построчно:

* Уже привычное нам наследование от класса Sprite для инициализации объекта как спрайта в pygame.
* После инициализации загружаем изображение и скрываем его фон.
* Создаем прямоугольную область с помощью метода get\_rect().
* Далее создадим два объекта изображений: image\_left - птенец смотрит влево и image\_right - птенец смотрит вправо. Чтобы создать отзеркаленное изображение используем метод flip().
* Назначим начальные координаты y и x нашей прямоугольной области.
* Обозначим скорость персонажа.

### 

### Метод draw()

Метод нужен для отрисовки персонажа на экране игры.

|  |
| --- |
| def draw(self):  screen.blit(self.image, self.rect) |

### 

### Метод update()

Этот метод нужен для обновления положения объекта в пространстве игры.

|  |
| --- |
| def update(self):  if keys[pygame.K\_SPACE]:  self.speed = 20  else:  self.speed = 6  if keys[pygame.K\_LEFT]:  self.rect.move\_ip(-self.speed, 0)  self.image = self.image\_left  if keys[pygame.K\_RIGHT]:  self.rect.move\_ip(self.speed, 0)  self.image = self.image\_right  if self.rect.left < 0:  self.rect.left = 0  if self.rect.right > SCREEN\_WIDTH:  self.rect.right = SCREEN\_WIDTH |

* В первой строчке добавим способность нашему птенчику:

При нажатии пробела он будет ускоряться в указанном направлении.

* Далее мы проверяем в какую сторону движется птенец и двигаем его в соответствующее направление с указанной скоростью, при этом инициализируем правильное изображение.
* Последние два условия отвечают за то, чтобы птенчик не вылетал за пределы краев экрана.

На этом основной класс Player завершен.

## Класс Item

Этот класс необходим для реализации взаимодействия с полезными объектами, которые птенец будет собирать.

### Инициализация

|  |
| --- |
| class Item(pygame.sprite.Sprite):  def \_\_init\_\_(self):  super().\_\_init\_\_()  self.image = random.choice(item\_images)  self.rect = self.image.get\_rect()  self.speed = random.randrange(3, 7)  self.rect.x = random.randrange(0, SCREEN\_WIDTH - self.rect.width)  self.rect.y = random.randrange(-100, 0) |

* Идентично прошлому классу инициализируем класс Item как Sprite из Pygame.
* Выберем изображение случайно из ранее созданного списка item\_images.
* Получим прямоугольную область для работы со спрайтом, исходя из текущего изображения.
* Выберем рандомную скорость в диапазоне от 3 до 7.
* Случайно выберем стартовые координаты предмета.

### 

### Метод draw()

Необходимый метод для того, чтобы объект работал в качестве спрайта.

|  |
| --- |
| def draw(self):  screen.blit(self.image, self.rect) |

### Метод update()

Аналогичный прошлому классу метод:

|  |
| --- |
| def update(self):  global hp  self.rect.y += self.speed  if self.rect.top > SCREEN\_HEIGHT:  self.kill()  hp -= 1 |

* Перемещение предмета по оси y со скоростью speed.
* Возьмем переменную hp из глобальной зоны видимости, будем отнимать значение, если предмет пропал за нижней гранью экрана.

## 

## Класс Enemy

Этот класс отвечает за монстров в игре.

### Инициализация

|  |
| --- |
| class Enemy(pygame.sprite.Sprite):  def \_\_init\_\_(self):  super().\_\_init\_\_()  self.image = random.choice(enemy\_images)  self.rect = self.image.get\_rect()  self.speed = random.randrange(4, 8)  self.rect.y = random.randrange(-100, 0)  self.rect.x = random.randrange(0, SCREEN\_WIDTH - self.rect.width) |

* Инициализируем объект как Sprite в pygame.
* Выбираем случайное изображение из пула возможных.
* Устанавливаем прямоугольную область вокруг изображения.
* Выбираем случайную скорость из диапазона (4, 8).
* Выбираем случайное положение в пространстве.

### Метод draw()

В этом классе действуем по стандартной схеме:

|  |
| --- |
| def draw(self):  screen.blit(self.image, self.rect) |

### Метод update()

|  |
| --- |
| def update(self):  self.rect.y += self.speed  if self.rect.top > SCREEN\_HEIGHT:  self.kill() |

На этом классы в нашей игре завершены.

## Переменные и группы

* Перед игрой необходимо подготовить стартовые объекты:
* Объект clock, отвечающий за фпс.
* Счетчик победных очков.
* Показатель оставшихся жизней.
* Переменная running, отвечающая за главный цикл.
* Экземпляр класса Player.
* Подготовим три группы спрайтов:
  + all\_sprites - все спрайты в игре, группа нужна для отрисовки всех разом, сразу добавим в нее экземпляр класса Player.
  + items - группа для полезных предметов.
  + enemies - группа для вражеских монстров.

|  |
| --- |
| clock = pygame.time.Clock() score = 0 hp = 5  running = True  player = Player()  all\_sprites = pygame.sprite.Group() all\_sprites.add(player) items = pygame.sprite.Group() enemies = pygame.sprite.Group() |

## Ивенты

Подготовим ивенты для создания монстров и предметов:

|  |
| --- |
| add\_item = pygame.USEREVENT + 1 pygame.time.set\_timer(add\_item, 2000)  add\_enemy = pygame.USEREVENT + 2 pygame.time.set\_timer(add\_enemy, 7000) |

Будем создавать предметы каждые 2 секунды, а монстров - каждые 7.

## 

## Главный цикл

В главном цикле осталось запустить механизмы друг за другом и добавить условия конца игры.

### Захват ивентов

В самом начале цикла оформим цикл, который проходит по списку всех активных ивентов:

|  |
| --- |
| while running:  for event in pygame.event.get():  if event.type == pygame.QUIT:  running = False  if event.type == add\_item and len(all\_sprites) < 6:  item = Item()  all\_sprites.add(item)  items.add(item)  if event.type == add\_enemy and len(all\_sprites) < 6:  enemy = Enemy()  all\_sprites.add(enemy)  enemies.add(enemy) |

Перечислим наши 3 условия:

* При нажатии на крестик, выходим из игры, как и в прошлых проектах.
* Добавим ограничение - 6 спрайтов на экране.
* Если пришел счетчик добавить предмет и ограничение позволяет добавить предмет, создаем экземпляр класса Item, добавляем его в соответствующие группы спрайтов.
* Повторяем аналогичные действия и для монстров.

### Захват нажатий клавиш

Тут все просто, нам понадобится уже знакомая строчка:

|  |
| --- |
| keys = pygame.key.get\_pressed() |

### 

### Изменения на экране

Эту часть кода мы посвятим перемещениям всех спрайтов на экране.

|  |
| --- |
| screen.blit(bg, [0, 0]) all\_sprites.update() all\_sprites.draw(screen) |

Как мы уже знаем, для того, чтобы отразить все изменения на экране, нужно заново отрисовать фон (blit(), передаем изображение и координаты) и нарисовать все спрайты в новых координатах. Для этого нам понадобятся методы update и draw.

### Условие поражения и ПО

Добавим два условия, при которых вызываем функция game\_over().

И 1 условие для подсчета победных очков.

|  |
| --- |
| if hp == 0:  game\_over() if pygame.sprite.spritecollide(player, enemies, True):  game\_over() if pygame.sprite.spritecollide(player, items, True):  score += 1 |

* Если показатель здоровья оказался на нуле (пропущенные предметы достигли числа 5), то выводим сообщение о проигрыше на экран.
* Если игрок столкнулся с монстром (вспомним, что такое коллизия и вызываем метод spritecollide(), в который передаем спрайт, группу спрайтов для проверки и флаг на удаление спрайта из переданной группы, True - удаляет последний), игра проиграна.
* Если игрок столкнулся с предметом, удаляем предмет из игры и добавляем 1 к счетчику победных очков.

### 

### Score и HP

Отобразим в правом верхнем углу наши показатели победных очков и жизней:

|  |
| --- |
| put\_text(("score: " + str(score)), SCREEN\_WIDTH - 80, 40) put\_text(("HP: " + str(hp)), SCREEN\_WIDTH - 80, 70) |

### Конец цикла

|  |
| --- |
| pygame.display.flip()  clock.tick(60) |

В конце цикла отобразим все изменения на экране с помощью метода flip() и запустим ограничение FPS.

За циклом выйдем из pygame для правильного завершения программы:

|  |
| --- |
| pygame.quit() |

# Заключение

В нашем сегодняшнем проекте мы подробнее познакомились с анимацией спрайтов и взаимодействием между собой. Повторили и закрепили знания в работе с классами и ООП. Познакомились с новыми методами модуля pygame. Получилась очень простая, но симпатичная игра, которую вы можете дополнять на ваше усмотрение. Попробуйте использовать другие изображения, их вы сможете найти на <https://opengameart.org/>  
Удачи!

# Ссылка на проект

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

[Игра 7: "Птенец"](https://drive.google.com/file/d/1dt5_oyV8q_wbyCAu7_zl3n6xr58nX5yh/view?usp=sharing)