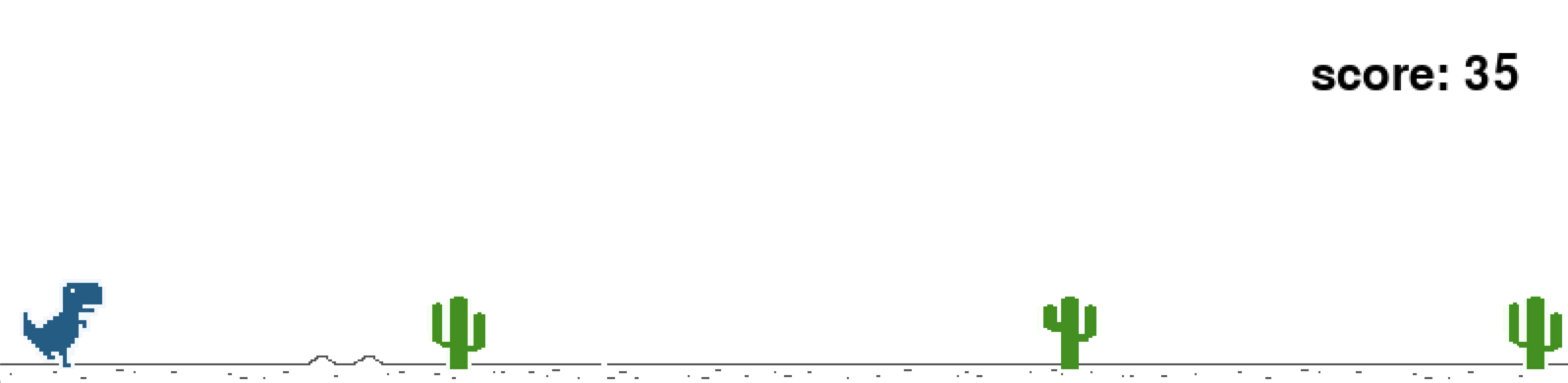
Игра 6: Дино



# Повторение

Как обычно, наши первые строчки в игре будут повторять предыдущие проекты - это правильно в программировании использовать уже имеющиеся наработки и изменять их от проекта к проекту. В этом модуле мы рассмотрим уже знакомые нам конструкции, некоторые из них немного модифицируем.

## 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 нестандартный размер экрана:

|  |
| --- |
| SCREEN\_WIDTH = 800 SCREEN\_HEIGHT = 200  screen = pygame.display.set\_mode((SCREEN\_WIDTH, SCREEN\_HEIGHT)) pygame.display.set\_caption('Dino run!') |

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(30) |

Оставим стандартное ограничение в 30 кадров.

## 

## Функция put\_text()

Используем нашу функцию put\_text() из прошлых игр, на этот раз будем писать черным цветом по белому экрану:

|  |
| --- |
| font = pygame.font.Font(None, 36)   def put\_text(message, x, y):  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.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() |

Вызываем функцию в тот момент, когда игрок проиграет.

# Цель игры



В этой игре мы будем управлять динозавриком. Его цель - добежать как можно дальше, не задев при этом колючие кактусы. Поможем перепрыгивать препятствия с помощью нажатия на пробел.

# План игры

Как обычно, придержимся рабочей стратегии планировать свои действия, прежде чем их совершать.

## Объекты

В нашей игре будет три объекта:

* Динозавр - персонаж игрока, им мы будем управлять.
* Кактус - колючие препятствия, которые нам нужно будет избегать.
* Земля - графический объект, который понадобится нам для иллюзии бега динозавра.

Для всех этих объектов нам понадобятся изображения. Все файлы можно найти в папке по ссылке: [изображения](https://drive.google.com/drive/folders/1oadyGLzkTP15Q2oDK1mV57WQNrUi6z7F?usp=sharing).

Загрузите файлы и оставьте их в корне игры, ниже мы рассмотрим работу с этими изображениями.

## Процесс игры

Процесс этой игры не будет отличаться особой сложностью:

* Игрок управляет динозавриком с помощью кнопки пробел.
* Справа экрана выезжают кактусы, которые игрок должен перепрыгнуть.
* Если динозавр наткнется на кактус, игра закончена.
* Чем дальше пробежит динозаврик, тем больше очков заработает игрок.

# Реализация

## Класс Dino

Это наш главный класс. Наследуем параметры класса Sprite и загрузим изображение, после чего создадим прямоугольную область вокруг него.

### Инициализация

|  |
| --- |
| class Dino(pygame.sprite.Sprite):  def \_\_init\_\_(self):  super(Dino, self).\_\_init\_\_()  self.image = pygame.image.load('dino.png').convert()  self.image = pygame.transform.scale(self.image, (44, 47))  self.rect = self.image.get\_rect() |

* Как мы помним, загрузка изображения происходит с помощью метода image.load, в который мы подаем файл нужного изображения. После чего используем метод convert() для улучшения отображения.
* Установим размер динозавра 44 на 47 пикселей с помощью метода transform.scale().
* Последней строчкой устанавливаем прямоугольную область динозаврика.

Для анимации мы добавим еще два изображения бегущего динозаврика, для этого также инициализируем два объекта и создадим массив images со всеми изображениями.

|  |
| --- |
| self.image1 = pygame.image.load('dino1.png').convert() self.image1 = pygame.transform.scale(self.image1, (44, 47)) self.image2 = pygame.image.load('dino2.png').convert() self.image2 = pygame.transform.scale(self.image2, (44, 47)) self.images = [self.image, self.image1, self.image2] |

Установим начальное положение динозаврика:

|  |
| --- |
| self.rect.x = 10 self.height = SCREEN\_HEIGHT - 10 - self.rect.height self.rect.y = self.height |

* Отступим 10 пикселей от левого края экрана, там будет расположена левая граница прямоугольной области динозаврика.
* Отступим 10 пикселей от нижнего края экрана и вычтем из этого значения высоту прямоугольной области динозаврика, это нужно для того, чтобы левый верхний угол встал на нужную нам точку. Сохраним эту координату в переменную height.
* Присвоим значение height координате y нашей прямоугольной области rect.

Как мы помним, наш динозаврик умеет прыгать. Для этого мы инициализируем дополнительные переменные, которые будем использовать дальше в коде:

|  |
| --- |
| self.isJumping = False self.jumpSpeed = 13 self.run\_counter = 0 |

Первые две переменные определяют состояние прыжка.

Третья переменная нужна будет для правильной смены изображений при беге динозаврика - будем менять положение ног динозаврика каждые 10 итераций.

На этом инициализация закончена.

### Метод draw()

Все очень просто: отобразим наше изображение динозаврика в его прямоугольной области с помощью метода blit().

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

### 

### Метод update()

Наш последний в классе метод по порядку, но не последний по значимости.

В этом методе мы рассмотрим единственное перемещение персонажа - прыжок, будем отлавливать нажатие клавиши пробел с помощью массива keys, который создадим позже в главном цикле.

|  |
| --- |
| if keys[pygame.K\_SPACE] and self.rect.y == self.height:  self.isJumping = True if self.isJumping:  self.rect.y -= self.jumpSpeed  self.jumpSpeed -= 1  self.image = self.images[0]  self.run\_counter = 0 |

Для имитации прыжка мы будем каждую итерацию отнимать из текущей координаты скорость прыжка, после чего уменьшим эту скорость. Так к самой верхней точке, динозаврик замедлится и немного зависнет в воздухе. При прыжке будем использовать первую картинку из списка images. В прыжке сбросим счетчик, отвечающий за бег.

Иначе, если мы не в прыжке, будем менять изображение динозаврика каждые 10 итераций с помощью счетчика. Будем обнулять переменную каждые 20 итераций.

|  |
| --- |
| else:  if self.run\_counter < 10:  self.image = self.images[1]  else:  self.image = self.images[2]  if self.run\_counter > 20:  self.run\_counter = 0  self.run\_counter += 1 |

В конце добавим сброс состояния прыжка. Наша скорость становится отрицательной и динозаврик летит вниз. Как только он достиг своей стартовой высоты, сбросим основные параметры:

|  |
| --- |
| if self.rect.y > self.height:  self.rect.y = self.height  self.isJumping = False  self.jumpSpeed = 13 |

На этом класс Dino завершен.

## Класс Cactus

По аналогии с классом Dino, инициализируем кактус как спрайт. После чего инициализируем схожим способом.

### Инициализация

|  |
| --- |
| class Cactus(pygame.sprite.Sprite):  def \_\_init\_\_(self):  super(Cactus, self).\_\_init\_\_()  self.image1 = pygame.image.load('cactus1.png').convert()  self.image2 = pygame.image.load('cactus2.png').convert()  self.image3 = pygame.image.load('cactus3.png').convert()  self.image1 = pygame.transform.scale(self.image1, (30, 40))  self.image2 = pygame.transform.scale(self.image2, (30, 40))  self.image3 = pygame.transform.scale(self.image3, (30, 40))  self.image = random.choice([self.image1, self.image2, self.image3])  self.rect = self.image.get\_rect()  self.rect.bottom = SCREEN\_HEIGHT - 10  self.rect.x = SCREEN\_WIDTH + self.rect.width + random.randrange(10, 150)  self.speed = 7 |

* Создадим три объекта с изображениями, использовав соответствующие файлы.
* Изменим их размер на 30х40 пикселей каждый.
* После чего для каждого экземпляра класса Cactus, выберем один из трех кактусов с помощью метода random.choice().
* Создадим прямоугольную область вокруг выбранного изображения.
* Отступим 10 пикселей от нижней границы экрана.
* Координату x установим за правой границей экрана на расстоянии ширина объекта + случайное расстояние из диапазона (10, 150), так кактусы будут появляться на разном друг от друга расстоянии.
* Установим стандартную скорость равную 7.

### 

### Метод draw()

Отображаем изображения в прямоугольной области объекта.

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

### Метод update()

* Движение кактусов зависит от скорости.
* Удаляем кактус из всех групп спрайтов, если кактус вышел за левую границу экрана.

|  |
| --- |
| def update(self):  self.rect.x -= self.speed   if self.rect.right < 0:  self.kill() |

## Класс Ground

Класс отвечает за отображение движущейся земли. Для этого нам понадобится создать два одинаковых объекта земли и склеить их между собой. Так получится бесконечная движущаяся дорожка.

### Инициализация

Инициализируем два одинаковых объекта и установим скорость:

|  |
| --- |
| class Ground:  def \_\_init\_\_(self):  self.image = pygame.image.load('ground.png').convert()  self.image.set\_colorkey((0, 0, 0), pygame.RLEACCEL)  self.rect = self.image.get\_rect()  self.rect.bottom = SCREEN\_HEIGHT  self.image1 = pygame.image.load('ground.png').convert()  self.image1.set\_colorkey((0, 0, 0), pygame.RLEACCEL)  self.rect1 = self.image1.get\_rect()  self.rect1.bottom = SCREEN\_HEIGHT  self.rect1.left = self.rect.right  self.speed = 7 |

* Вспомним, что метод set\_colorkey() отвечает за прозрачность изображения. Изображение с землей необходимо обработать, используя черный цвет.
* Прикрепим левую часть второго объекта к правой части первого объекта. По сути, мы склеили две одинаковых полоски земли. Теперь, когда правая часть выйдет из правой границы экрана, за ней последует второй объект - будет казаться, что земля бесконечная, но по факту использовано две одинаковых картинки.
* Установим скорость 7, также как у кактусов.

### 

### Метод draw()

Отобразим оба изображения в их прямоугольных областях.

|  |
| --- |
| def draw(self):  screen.blit(self.image, self.rect)  screen.blit(self.image1, self.rect1) |

### Метод update()

|  |
| --- |
| def update(self):  self.rect.x -= self.speed  self.rect1.x -= self.speed   if self.rect.right < 0:  self.rect.left = self.rect1.right   if self.rect1.right < 0:  self.rect1.left = self.rect.right |

* Двигаем обе части нашей склеенной земли со скоростью speed.
* Если правая часть первого объекта вышла за левую границу экрана, приклеим левую часть этого объекта к правой части второго объекта.
* Аналогично для второго объекта.

В этом методе заключается вся магия бесконечной беговой дорожки, по которой будет бежать наш динозаврик.

На этом класс Ground закончен.

Мы создали все классы и теперь готовы приступать к реализации основной логики игры.

## Дополнительные переменные

Добавим к нашим двум переменным еще две, отвечающие за счетчик счета и за паузу (в этой игре добавим возможность пользователю устанавливать паузу).

|  |
| --- |
| clock = pygame.time.Clock() running = True score = 0 pause = False |

## Экземпляры классов и группы спрайтов

|  |
| --- |
| ground = Ground() dino = Dino()  enemies = pygame.sprite.Group() all\_sprites = pygame.sprite.Group() all\_sprites.add(dino) |

Создадим два экземпляра класса: для дино и для земли.

Создадим группу спрайтов enemies и all\_sprites. Добавим во вторую группу экземпляр дино.

## Создание ивента

Создадим ивент для появления кактусов. Обозначим ограничение в полторы секунды.

|  |
| --- |
| add\_cactus = pygame.USEREVENT + 1 pygame.time.set\_timer(add\_cactus, 1500) |

## 

## Главный цикл

Вся логика будет реализована в главном цикле игры.

### Ивенты

|  |
| --- |
| while running:  for event in pygame.event.get():  if event.type == pygame.QUIT:  running = False  if event.type == pygame.KEYDOWN:  if event.key == pygame.K\_ESCAPE:  if not pause:  pause = True  else:  pause = False  if event.type == add\_cactus and len(enemies) < 3:  cactus = Cactus()  all\_sprites.add(cactus)  enemies.add(cactus) |

* Отлавливаем нажатие на крестик для выхода из игры.
* При нажатии на эскейп, ставим паузу, если она не поставлена или отжимаем паузу, если она поставлена.
* Если активируется ивент add\_cactus и число созданных кактусов меньше 3 (это сделано для дополнительного баланса), создадим новый экземпляр класса Cactus, добавим его в две наши группы спрайтов.

### Нажатые клавиши

Создадим массив keys, содержащий в себе все нажатые на данный момент клавиши.

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

### 

### Движение игры

Если пауза не поставлена, обновляем экран известными нам методами:

|  |
| --- |
| if not pause:  screen.fill(WHITE)  score += 1  ground.update()  ground.draw()  all\_sprites.update()  all\_sprites.draw(screen) |

* Заполняем экран фоновым цветом.  
  Добавляем к счетчику очков.
* Обновляем и отображаем землю.
* Обновляем и отображаем все спрайты с помощью группы all\_sprites.

Если пауза поставлена, выводим соответствующее сообщение на экран:

|  |
| --- |
| else:  put\_text("pause", SCREEN\_WIDTH // 2, SCREEN\_HEIGHT // 2) |

Также всегда выводим счетчик очков на экран. Разделим значение на 5, чтобы очки не копились слишком быстро:

|  |
| --- |
| put\_text(("score: " + str(score // 5)), SCREEN\_WIDTH - 80, 40) |

### Проверка коллизии

Если динозаврик столкнулся с любым кактусом, игра закончена, и мы вызываем функцию game\_over():

|  |
| --- |
| if pygame.sprite.spritecollideany(dino, enemies):  game\_over() |

### 

### Конец главного цикла

В конце цикла не забудьте отобразить все изменения на экране и добавить ограничение по кадрам в секунду. За циклом выйдем из модуля pygame и закроем программу.

|  |
| --- |
| pygame.display.flip()  clock.tick(30) pygame.quit() exit() |

# Заключение

Сегодня мы более подробно рассмотрели работу с изображениями и смогли реализовать анимацию с помощью движения или поочередной смены файлов, это очень эффективный подход, который применяется постоянно в разработке игр. Повторили основные моменты создания игр, классы и группы спрайтов, также как и взаимодействия между ними. Впервые реализовали паузу в игре, при хорошей структуре кода, сделать это получается очень просто.

Получилась классическая игра подобная той, в которую можно поиграть при выключенном интернете в браузере Chrome. Отличная работа!

# Ссылка на проект

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

[Игра 6: "Дино"](https://drive.google.com/file/d/1S32c33Ss8Wi8OozwFe12h5VgFwDVbsWP/view?usp=sharing)