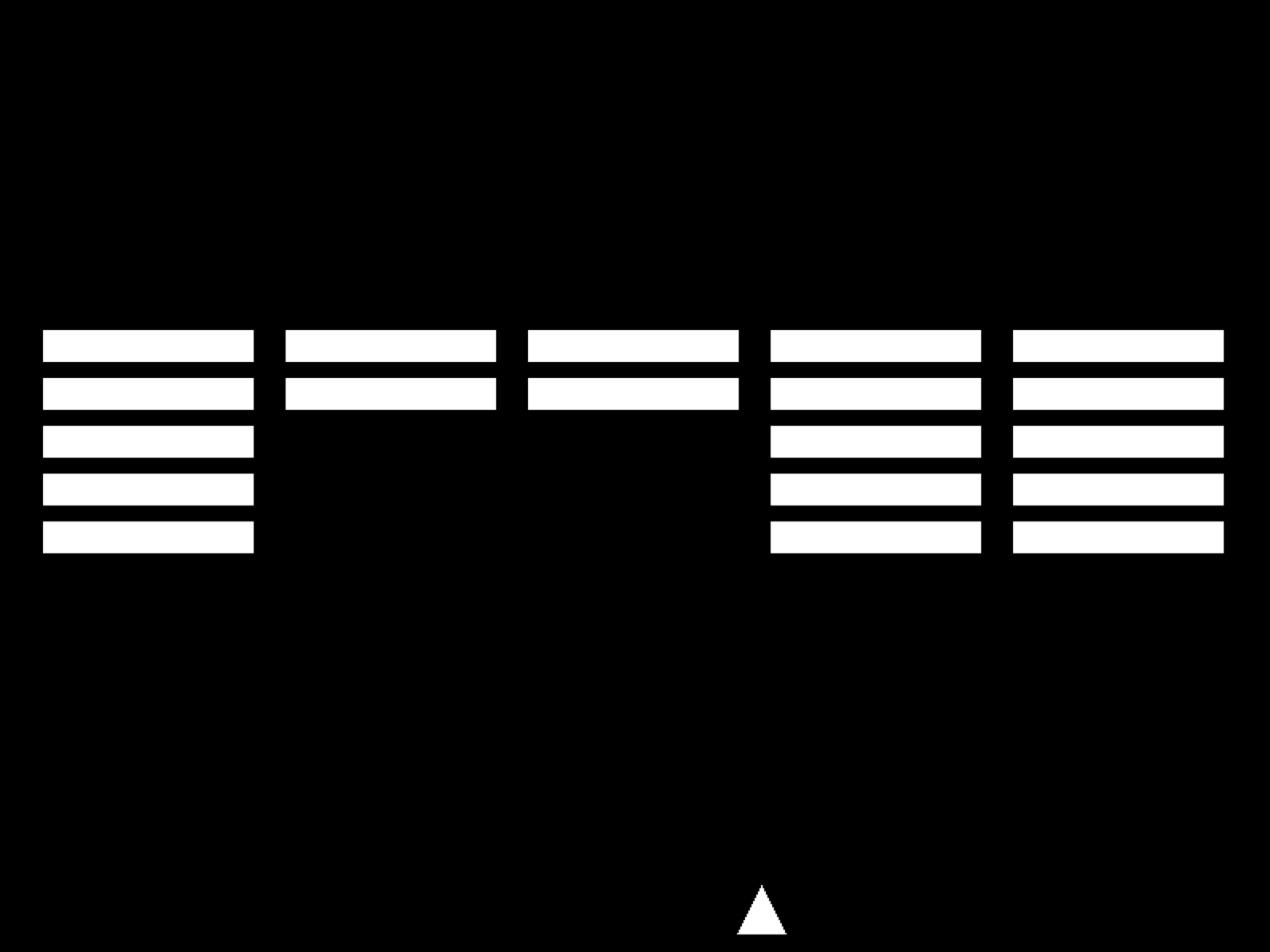
Игра 4: Звездные сражения



# 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:

|  |
| --- |
| import pygame |

* 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 две переменные для хранения цветов нашей игры:

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

## 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 = 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 белым цветом по черному экрану:

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

## Функция victory()

Как и в прошлый раз, будем использовать функцию для уведомления о победе:

|  |
| --- |
| def victory():  screen.fill(BLACK)  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() |

Вызывать эту функцию мы будем при условии победы.

## Функция game\_over()

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

|  |
| --- |
| def game\_over():  screen.fill(BLACK)  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() |

Вызывать эту функцию мы будем, когда игрок проиграет.

# Цель игры



Если вы смотрели “Звездный путь”, то знаете, что героям в космосе приходится совсем нелегко. С помощью одного маленького кораблика им приходится спасать вселенную от злодеев, которые хотят уничтожить ее.

Сегодня мы создадим абстрактную игру близкую к космической тематике: мы управляем маленьким космическим кораблем, цель которого - уничтожить надвигающиеся вражеские корабли, но берегитесь: с каждым уровнем их становится все больше.

*Да пребудет с тобой сила!*

# План игры

Давайте составим логическую модель нашей будущей игры. Для этого нам надо подумать о том, какие объекты будут взаимодействовать между собой внутри игры.

## Объекты

1. Космический корабль - это то, чем наш игрок будет управлять.
2. Вражеские корабли - надвигающаяся на игрока угроза. Их количество будет увеличиваться с каждым уровнем.
3. Пульки космического корабля - чтобы уничтожить врагов, нам нужно хорошее космическое оружие - на нажатие клавиши будем выпускать пульки из корабля игрока.

## Процесс игры

Как будет происходить сам процесс игры:

* Корабль игрока уже на позиции и готов защищать галактику от угрозы.
* Вверху экрана появляются враги и начинают приближаться к игроку.
* Игрок уничтожает врагов с помощью бластера и не дает приблизиться к нижней части экрана.

# Реализация

## Класс Player

Для создания класса игрока, мы также наследуем параметры класса Sprite из pygame. Сам спрайт - всегд прямоугольная область на экране, так его и инициализируем:

### Инициализация

|  |
| --- |
| class Player(pygame.sprite.Sprite):  def \_\_init\_\_(self):  super().\_\_init\_\_()  self.surf = pygame.Surface((30, 30))  self.rect = self.surf.get\_rect(center=(SCREEN\_WIDTH//2,SCREEN\_HEIGHT-30)) |

Пусть область будет квадратом размером 30 на 30 пикселей. Создадим нашего игрока по центру экрана по ширине, близко к нижней границе, для этого мы используем метод get\_rect, который возвращает прямоугольную область, с ней мы и будем работать дальше.

### Отрисовка

|  |
| --- |
| def draw(self):  pygame.draw.polygon(screen, WHITE, [[self.rect.x+15, self.rect.y],  [self.rect.x, self.rect.y+30],  [self.rect.x+30, self.rect.y+30]]) |

Так как наш космический корабль будет треугольным, то мы будем использовать метод polygon в модуле draw в pygame. Метод принимает точки вершин многоугольника, у нашего корабля их 3:

* Напомню, что координаты x и y прямоугольника - это координаты его левого верхнего угла.
* Верхняя точка треугольника будет удалена на половину ширины квадрата - 15 пикселей и расположена на высоте y.
* Левая нижняя точка треугольника расположена прямо под точкой левого верхнего угла - добавим высоту квадрата к координате y.
* Правая нижняя точка треугольника расположена на расстоянии ширины квадрата от левой нижней точки - добавим к координате x ширину квадрата.

### Управление

Для управления нашим кораблем мы будем использовать клавиши направо и налево. Отслеживать нажатия клавиш правильнее в главном цикле игры, добавим эту строчку:

|  |
| --- |
| pressed\_keys = pygame.key.get\_pressed() |

Метод get\_pressed() модуля key возвращает список всех зажатых на данный момент клавиш, добавьте эту строчку в главный цикл после цикла с проверкой event.

В методе нашего класса Player мы используем созданный нами список:

|  |
| --- |
| def update(self):  if pressed\_keys[pygame.K\_LEFT] and self.rect.left > 0:  self.rect.move\_ip(-5, 0)  if pressed\_keys[pygame.K\_RIGHT] and self.rect.right < SCREEN\_WIDTH:  self.rect.move\_ip(5, 0) |

* Проверяем, содержится ли в списке клавиша “налево”, и не находится ли наш игрок уже у левого края экрана, тогда двигаем его левее на 5 пикселей с помощью метода move\_ip(), который обращается к нашему объекту rect.
* Проверяем, содержится ли в списке клавиша “направо”, и не находится ли наш игрок у правого края экрана, тогда двигаем его правее на 5 пикселей.

Класс игрока готов, но пока мы не создали экземпляр класса, работа с ним не начнется.

Предлагаю сначала разобраться с остальными объектами в игре.

## Класс Bullet

Здесь все проще, создадим класс по аналогии:

1. Инициализируем прямоугольную область размером 2 на 6. При создании объекта пульки мы будем передавать координаты игрока - так мы сможем создавать пульки над игроком, с этим нам поможет флаг center в методе get\_rect().
2. В методе update() будем удалять пульку, если она вылетела за верхнюю границу экрана с помощью метода kill() (напомню, что этот метод унаследован от класса Sprite, метод удаляет текущий экземпляр класса из всех групп, которые инициализированы как группы спрайтов). Иначе - меняем ее координату на 10, это будет ее скоростью.
3. Метод draw() отрисует пульку белым цветом на экране screen.

|  |
| --- |
| class Bullet(pygame.sprite.Sprite):  def \_\_init\_\_(self, x, y):  super().\_\_init\_\_()  self.surf = pygame.Surface((2, 6))  self.rect = self.surf.get\_rect(center=(x, y))   def update(self):  if self.rect.bottom < 0:  self.kill()  self.rect.y -= 10   def draw(self):  pygame.draw.rect(screen, WHITE, self.rect) |

## 

## Группы спрайтов

Перед тем, как продолжить, создадим группы спрайтов в коде над главным циклом:

|  |
| --- |
| all\_sprites = pygame.sprite.Group()  enemies = pygame.sprite.Group() bullets = pygame.sprite.Group()  player = Player() all\_sprites.add(player) |

Всего у нас будет 3 группы:

1. all\_sprites - все существующие спрайты. Эту группу мы будем использовать для отображения всех спрайтов на экране и их движения.
2. enemies - вражеские корабли. В эту группу мы будем заносить экземпляры классов врагов, использовать эту группу будем для того, чтобы отделить врагов от всех остальных спрайтов, это очень важный момент.
3. bullets - пульки космического корабля. Очень важная группа, ее мы будем использовать для проверки столкновений с группой врагов.

В последних двух строчках мы создали экземпляр класса Player и поместили его в группу all\_sprites.

Добавим в главный цикл отображение и движение всех спрайтов после строчки с созданием списка зажатых клавиш:

|  |
| --- |
| all\_sprites.update()  screen.fill(BLACK) for entity in all\_sprites:  entity.draw() |

1. Во всех классах наших объектов мы переопределяем метод update(), в нашем случае отвечает за движение объектов, вызываем update() у всех спрайтов, которые состоят в группе all\_sprites первой строчкой в этом блоке.
2. Заполняем экран черным цветом для дальнейшей прорисовки на экране по-новому.
3. Проходим циклом по группе all\_sprites и у каждой сущности (entity) вызываем метод draw().

## 

## Создаем объекты пулек

|  |
| --- |
| for event in pygame.event.get():  if event.type == pygame.QUIT:  running = False  if event.type == pygame.KEYDOWN:  if event.key == pygame.K\_SPACE:  bullet = Bullet(player.rect.x + 15, player.rect.y)  all\_sprites.add(bullet)  bullets.add(bullet)  if event.key == pygame.K\_RETURN:  if not ready:  ready = True |

В цикле для отслеживания event, добавим проверку на нажатие на клавишу пробел. Если нажали на пробел, создаем объект пульки, передавая в него координату верхней точки треугольника, откуда и будут вылетать пульки. Добавляем этот объект в группу all\_sprites и bullets.

На нажатие клавиши Enter мы будем запускать уровень, за это отвечают последние 3 строчки в блоке, добавим сопутствующие строчки в конец главного цикла.

## Готовность игрока

Перед отображением всех изменений на экране с помощью flip() и ограничением кадров в секунду, будем выводить сообщение на экране с инструкцией о том, как начать следующий уровень:

|  |
| --- |
| if not ready:  put\_text("Press enter to start next level", SCREEN\_WIDTH//2,SCREEN\_HEIGHT//2) pygame.display.flip() clock.tick(30) |

## Класс Enemy

Враги - все также объекты-спрайты, которые имеют в себе инструкции инициализации, обновления и прорисовки:

1. Инициализация. Будем передавать врагу его ширину и координаты его центра, это необходимо для создания сетки из кораблей врагов, которая будет увеличиваться с каждым уровнем.
2. Обновление. В этом методе мы будем проверять, не столкнулся ли враг с какой-либо пулькой из группы bullets. Также все вражеские корабли будут медленно лететь вниз экрана.
3. Прорисовка. Вызываем метод для прорисовки прямоугольного вражеского корабля.

|  |
| --- |
| class Enemy(pygame.sprite.Sprite):  def \_\_init\_\_(self, x, y, width):  super().\_\_init\_\_()  self.surf = pygame.Surface((width, 20))  self.rect = self.surf.get\_rect(center=(x, y))   def update(self):  if self.rect.bottom > SCREEN\_HEIGHT:  game\_over()  bul = pygame.sprite.spritecollideany(self, bullets)  if bul:  self.kill()  bul.kill()   def draw(self):  pygame.draw.rect(screen, WHITE, self.rect) |

Если какой-либо из кораблей пересек нижнюю границу экрана, игра закончена, мы вызываем функцию game\_over().

Для отслеживания столкновений вражеского корабля с любой пулькой, будем использовать метод spritecollideany(), в него мы передаем экземпляр класса объекта, у которого мы проверяем столкновений с группой других спрайтов - это второй передаваемый аргумент. Метод возвращает объект спрайта, с которым столкнулся враг, если столкновение произошло, иначе - None.

Если враг столкнулся с пулькой, удалим оба объекта с помощью метода kill().

## Переменные

Перед главным циклом обозначим:

* Стартовый уровень level.
* ready для проверки готовности игрока.
* running для непрерывности цикла.

|  |
| --- |
| level = 1 ready = False running = True |

## 

## 

## Функция gen\_enemies()

В этой функции мы будем создавать вражеские корабли в линию. Задача стоит таким образом, чтобы каждый новый уровень количество врагов увеличивалось, а размер уменьшался. Предлагаю такое решение:

|  |
| --- |
| def gen\_enemies():  line\_amount = 1 + level \* 2  enemy\_w = (SCREEN\_WIDTH - 40) // line\_amount - 20  y = -30 \* level \* 2  for \_ in range(level \* 2 + 1):  x = 30 + enemy\_w // 2  for \_ in range(line\_amount):  enemy = Enemy(x, y, enemy\_w)  all\_sprites.add(enemy)  enemies.add(enemy)  x += enemy\_w + 20  y += 30 |

* line\_amount отвечает за то, сколько врагов будет в одной линии, зависит от текущего уровня.
* enemy\_w - это ширина одного вражеского корабля. Зависит от ширины экрана и количества врагов в линии.
* y - это координата первого созданного корабля, создаем корабли мы сверху вниз, удалим первую линию наверх относительно текущего уровня.
* Создадим в цикле линии врагов, количество которых будет равно level \* 2 + 1, все эти значения вы можете менять на свое усмотрение, если вам кажется более сбалансированным другое решение.
* x - это координата центра вражеского корабля. для первого удалим ее на 30 пикселей от края экрана.
* В цикле создадим заданное количество кораблей в линии. Добавим объект enemy в группы all\_sprites и enemies.
* Следующий кораблик удалим на 20 пикселей от текущего.
* После создания линии кораблей, спустимся вниз на 30 пикселей.

Так мы будем создавать сетку кораблей, размеры и количество которых будет меняться с каждым уровнем. Получилась не самая большая функция, но она выполняет важнейшую работу в создании уровней.

Эту функцию необходимо вызывать в начале уровня, когда игрок заявляет о своей готовности с помощью нажатия на клавишу Enter:

|  |
| --- |
| if event.key == pygame.K\_RETURN:  if not ready:  gen\_enemies()  ready = True |

## Следующий уровень

В самое начало нашего цикла добавим условие, что если количество врагов и пулек стало 0, и игрок активировал готовность до этого, тогда уровень пройден.

Если это был уровень 5, объявим победу.

Иначе, сбросим значение переменной ready и увеличим значение level.

Теперь игроку снова предложат нажать enter для начала уровня.

|  |
| --- |
| while running:  if len(enemies) == 0 and len(bullets) == 0 and ready:  if level == 5:  victory()  ready = False  level += 1 |

# Заключение

Сегодня мы более подробно изучили классы и взаимодействие групп спрайтов игроков. Получилась довольно сложная абстрактная игра с использованием простейших геометрических фигур. Отличная работа и увидимся в следующих уроках!

# Ссылка на проект

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

<https://drive.google.com/file/d/1PZjC0xt1GBY7OOWTalBKUCRzos73toR0/view?usp=sharing>