## Материалы к занятию

. Для более сложной физике есть специальное API Pymunk.

Для начала создадим шаблон нашего проекта

```
import arcade

SCREEN_TITLE = "PyMunk Platformer"

SCREEN_WIDTH = 800
SCREEN_HEIGHT = 600


class GameWindow(arcade.Window):

   def __init__(self, width, height, title):
       super().__init__(width, height, title)

   def setup(self):
       pass

   def on_key_press(self, key, modifiers):
       pass

   def on_key_release(self, key, modifiers):
       pass

   def on_update(self, delta_time):
       pass

   def on_draw(self):
       self.clear()


def main():
   window = GameWindow(SCREEN_WIDTH, SCREEN_HEIGHT, SCREEN_TITLE)
   window.setup()
   arcade.run()


if __name__ == "__main__":
   main()
```

Теперь определим константы, которые мы будем использовать. У нас есть спрайтовые плитки размером 128x128 пикселей. Они уменьшены до 50% ширины и 50% высоты (масштаб 0,5). Размер экрана установлен в сетку 25x15.

```
import math
```

```
from typing import Optional
import arcade

SCREEN_TITLE = "PyMunk Platformer"
SCREEN_WIDTH = 800
SCREEN_HEIGHT = 600

SPRITE_IMAGE_SIZE = 128

SPRITE_SCALING_PLAYER = 0.5
SPRITE_SCALING_TILES = 0.5

SPRITE_SIZE = int(SPRITE_IMAGE_SIZE * SPRITE_SCALING_PLAYER)

SCREEN_GRID_WIDTH = 25
SCREEN_GRID_HEIGHT = 15

SCREEN_WIDTH = SPRITE_SIZE * SCREEN_GRID_WIDTH
SCREEN_HEIGHT = SPRITE_SIZE * SCREEN_GRID_HEIGHT
```

Далее создадим переменную  для экземпляра класса

```
self.player_list: Optional[arcade.SpriteList] = None
```

В данном случае используется аннотация типов(мы подсказываем IDE, какой тип данных мы будем хранить) Можно записать и проще

```
self.player_list = None
```

и зададим зеленый цвет фона

```
arcade.set_background_color(arcade.color.AMAZON)
```

Также сразу создадим переменные для списков спрайтов и булевы значения статусов нажатия на клавиши влево и право

```
self.player_sprite: Optional[arcade.Sprite] = None

self.player_list: Optional[arcade.SpriteList] = None
self.wall_list: Optional[arcade.SpriteList] = None
self.bullet_list: Optional[arcade.SpriteList] = None
self.item_list: Optional[arcade.SpriteList] = None

self.left_pressed: bool = False
self.right_pressed: bool = False
```

В методе setup подключим необходимые текстуры и tilemap и зададим положения персонажа

```
def setup(self):
   self.player_list = arcade.SpriteList()
   self.bullet_list = arcade.SpriteList()

   map_name = ":resources:/tiled_maps/pymunk_test_map.json"

   tile_map = arcade.load_tilemap(map_name, SPRITE_SCALING_TILES)

   self.wall_list = tile_map.sprite_lists["Platforms"]
   self.item_list = tile_map.sprite_lists["Dynamic Items"]

   self.player_sprite =
arcade.Sprite(":resources:images/animated_characters/female_perso
n/femalePerson_idle.png",
                                       SPRITE_SCALING_PLAYER)

   grid_x = 1
   grid_y = 1
   self.player_sprite.center_x = SPRITE_SIZE * grid_x +
SPRITE_SIZE / 2
   self.player_sprite.center_y = SPRITE_SIZE * grid_y +
SPRITE_SIZE / 2

   self.player_list.append(self.player_sprite)
```

В метода on_draw отрисуем все необходимые элементы

```
def on_draw(self):
   self.clear()
   self.wall_list.draw()
   self.bullet_list.draw()
   self.item_list.draw()
   self.player_list.draw()
```

Теперь можно приступить к добавлению физики в нашу игру.
Добавим константы для нашей физики. Здесь мы устанавливаем:

- Константа для силы тяжести.
- Значения для «демпфирования». Демпфирование 1,0 приведет к тому, что элемент потеряет всю свою скорость, как только сила больше не применяется к нему. Демпфирование 0,5 приводит к потере 50% скорости за 1 секунду. Значение 0 является свободным падением.
- Значения трения. 0,0 — это лед, 1,0 — как резина.

- Массовый. Элемент по умолчанию равен 1. Мы делаем игрока 2, чтобы он могла легче перемещать предметы.
- Лимитами игроков являются горизонтальная и вертикальная скорость. Играть проще, если игрок ограничен постоянной скоростью. И более реалистично, потому что они не на колесах.

```
GRAVITY = 1500

DEFAULT_DAMPING = 1.0
PLAYER_DAMPING = 0.4

PLAYER_FRICTION = 1.0
WALL_FRICTION = 0.7
DYNAMIC_ITEM_FRICTION = 0.6

PLAYER_MASS = 2.0

PLAYER_MAX_HORIZONTAL_SPEED = 450
PLAYER_MAX_VERTICAL_SPEED = 1600
```

В методе init добавим переменную для нашего физического движка

```
self.physics_engine = Optional[arcade.PymunkPhysicsEngine]
```

Далее в методе setup мы создаем физический движок и добавляем спрайты. Игрок, стены и динамические предметы имеют разные свойства, поэтому они добавляются индивидуально.

```
damping = DEFAULT_DAMPING
gravity = (0, -GRAVITY)

self.physics_engine = arcade.PymunkPhysicsEngine(damping=damping,
                                                 gravity=gravity)
self.physics_engine.add_sprite(self.player_sprite,
                               friction=PLAYER_FRICTION,
                               mass=PLAYER_MASS,

moment=arcade.PymunkPhysicsEngine.MOMENT_INF,
                               collision_type="player",

max_horizontal_velocity=PLAYER_MAX_HORIZONTAL_SPEED,

max_vertical_velocity=PLAYER_MAX_VERTICAL_SPEED)

self.physics_engine.add_sprite_list(self.wall_list,
                                    friction=WALL_FRICTION,
                                    collision_type="wall",
```

```
body_type=arcade.PymunkPhysicsEngine.STATIC)
```

Вызовем метод step у нашего движка в методе on_update

```
def on_update(self, delta_time):
   self.physics_engine.step()
```

Следующим шагом мы заставим наш персонаж двигаться. Добавим переменную силы, которой мы будем сдвигать нашего персонажа

```
PLAYER_MOVE_FORCE_ON_GROUND = 8000
```

и добавим движение влево и право

```
def on_key_press(self, key, modifiers):

   if key == arcade.key.LEFT:
       self.left_pressed = True
   elif key == arcade.key.RIGHT:
       self.right_pressed = True

def on_key_release(self, key, modifiers):
   if key == arcade.key.LEFT:
       self.left_pressed = False
   elif key == arcade.key.RIGHT:
       self.right_pressed = False
```

Наконец, нужно применить правильную силу в on_update. Сила задается в кортеже с горизонтальной силой первой, а вертикальной силой второй.
Также устанавим трение, когда мы движемся к нулю, и когда мы не движемся к 1. Это важно, чтобы получить реалистичное движение.

```
def on_update(self, delta_time):
   self.physics_engine.step()
   if self.left_pressed and not self.right_pressed:
       force = (-PLAYER_MOVE_FORCE_ON_GROUND, 0)
       self.physics_engine.apply_force(self.player_sprite, force)
       self.physics_engine.set_friction(self.player_sprite, 0)
   elif self.right_pressed and not self.left_pressed:
```

```
        force = (PLAYER_MOVE_FORCE_ON_GROUND, 0)
        self.physics_engine.apply_force(self.player_sprite, force)
        self.physics_engine.set_friction(self.player_sprite, 0)
    else:
        self.physics_engine.set_friction(self.player_sprite, 1.0)
```

Отлично, движение реализовано.

Продолжим написание нашего прототипа игры

Для начала добавим прыжок

```
PLAYER_MOVE_FORCE_IN_AIR = 900
PLAYER_JUMP_IMPULSE = 1800
```

```
self.up_pressed: bool = False
self.down_pressed: bool = False
```

```
def on_key_press(self, key, modifiers):
    if key == arcade.key.LEFT:
        self.left_pressed = True
    elif key == arcade.key.RIGHT:
        self.right_pressed = True
    elif key == arcade.key.UP:
        self.up_pressed = True
        if self.physics_engine.is_on_ground(self.player_sprite):
            impulse = (0, PLAYER_JUMP_IMPULSE)
            self.physics_engine.apply_impulse(self.player_sprite,
impulse)
    elif key == arcade.key.DOWN:
        self.down_pressed = True
```

```
def on_update(self, delta_time):
    self.physics_engine.step()
    is_on_ground =
self.physics_engine.is_on_ground(self.player_sprite)
    if self.left_pressed and not self.right_pressed:
        if is_on_ground:
            force = (-PLAYER_MOVE_FORCE_ON_GROUND, 0)
        else:
            force = (-PLAYER_MOVE_FORCE_IN_AIR, 0)
        self.physics_engine.apply_force(self.player_sprite, force)
```

```
        self.physics_engine.set_friction(self.player_sprite, 0)
    elif self.right_pressed and not self.left_pressed:
        if is_on_ground:
            force = (PLAYER_MOVE_FORCE_ON_GROUND, 0)
        else:
            force = (PLAYER_MOVE_FORCE_IN_AIR, 0)
        self.physics_engine.apply_force(self.player_sprite, force)
        self.physics_engine.set_friction(self.player_sprite, 0)
    else:
        self.physics_engine.set_friction(self.player_sprite, 1.0)
```

Чтобы создать анимацию игрока, мы создаем пользовательский дочерний класс Sprite . Мы загружаем каждый кадр анимации, который нам нужен, включая его зеркальное отражение. Мы перевернем игрока лицом влево или вправо. Если игрок находится в воздухе, мы также будем переключаться между прыжком и падением.
Поскольку физический движок работает с небольшими числами с плавающей запятой, он часто имеет отклонения на небольшие величины. По этой причине у нас есть «мертвая зона». Мы не изменяем анимацию, пока она не выйдет за пределы этой зоны.
Нам также нужно контролировать, как далеко движется игрок, прежде чем мы изменим анимацию ходьбы, чтобы ноги выглядели синхронизированными с землей.

```
DEAD_ZONE = 0.1

RIGHT_FACING = 0
LEFT_FACING = 1

DISTANCE_TO_CHANGE_TEXTURE = 20
```

Создадим класс, который является дочерним для arcade.Sprite . Этот класс будет изменять анимацию игрока.
Метод загружает все текстуры __init__. Мы используем набор Toon Characters 1. Есть восемь текстур для ходьбы и текстур для состояния покоя, прыжков и падений.
Поскольку персонаж может смотреть влево или вправо, мы используем то, как обычное изображение, так и зеркальное.
Для многокадровой анимации ходьбы мы используем «одометр». Нам нужно переместить определенное количество пикселей, прежде чем менять анимацию. Если это значение слишком мало, наш персонаж двигает ногами, так, как будто он катается на коньках. Мы отслеживаем индекс нашей текущей текстуры, 0-7, так как их восемь.
Любой спрайт, перемещаемый движком Pymunk, будет иметь свой метод. Это можно использовать для изменения анимации.

```
class PlayerSprite(arcade.Sprite):
```

```
   def __init__(self):
       super().__init__()

       self.scale = SPRITE_SCALING_PLAYER

       main_path =
":resources:images/animated_characters/female_person/femalePerson
"

       self.idle_texture_pair =
arcade.load_texture_pair(f"{main_path}_idle.png")
       self.jump_texture_pair =
arcade.load_texture_pair(f"{main_path}_jump.png")
       self.fall_texture_pair =
arcade.load_texture_pair(f"{main_path}_fall.png")

       self.walk_textures = []
       for i in range(8):
           texture =
arcade.load_texture_pair(f"{main_path}_walk{i}.png")
           self.walk_textures.append(texture)

       self.texture = self.idle_texture_pair[0]

       self.hit_box = self.texture.hit_box_points

       self.character_face_direction = RIGHT_FACING

       self.cur_texture = 0

       self.x_odometer = 0

   def pymunk_moved(self, physics_engine, dx, dy, d_angle):
       if dx < -DEAD_ZONE and self.character_face_direction ==
RIGHT_FACING:
           self.character_face_direction = LEFT_FACING
       elif dx > DEAD_ZONE and self.character_face_direction ==
LEFT_FACING:
           self.character_face_direction = RIGHT_FACING

       is_on_ground = physics_engine.is_on_ground(self)

       self.x_odometer += dx

       if not is_on_ground:
           if dy > DEAD_ZONE:
               self.texture =
self.jump_texture_pair[self.character_face_direction]
               return
           elif dy < -DEAD_ZONE:
               self.texture =
self.fall_texture_pair[self.character_face_direction]
               return

       if abs(dx) <= DEAD_ZONE:
```

```
            self.texture =
self.idle_texture_pair[self.character_face_direction]
            return

        if abs(self.x_odometer) > DISTANCE_TO_CHANGE_TEXTURE:

            self.x_odometer = 0

            self.cur_texture += 1
            if self.cur_texture > 7:
                self.cur_texture = 0
            self.texture =
self.walk_textures[self.cur_texture][self.character_face_directio
n]
```

setup

```
self.player_sprite = PlayerSprite()
```

Добавим в игру возможность стрелять. Для начала определим несколько констант для использования. С какой силой стрелять пулей, масса пули и гравитация, которую нужно использовать для пули.
Если мы используем ту же гравитацию для пули, что и все остальное, она имеет тенденцию падать слишком быстро. Мы могли бы установить это значение к нулю, если бы мы хотели, чтобы она вообще не упала.

```
BULLET_MOVE_FORCE = 4500
BULLET_MASS = 0.1
BULLET_GRAVITY = 300
```

Затем мы вставим обработчик нажатия мыши, чтобы вставить код стрельбы пулей.
Нам необходимо:

- Создание спрайта маркера
- Нам нужно рассчитать угол от игрока до щелчка мыши
- Создать пулю подальше от игрока в правильном направлении, так как создание ее внутри игрока запутает физический движок.
- Добавление маркера в физический движок
- Приложите усилие к пуле, чтобы движение.

```
def on_mouse_press(self, x, y, button, modifiers):

   bullet = arcade.SpriteSolidColor(20, 5,
arcade.color.DARK_YELLOW)
```

```
    self.bullet_list.append(bullet)

    start_x = self.player_sprite.center_x
    start_y = self.player_sprite.center_y
    bullet.position = self.player_sprite.position

    dest_x = x
    dest_y = y

    x_diff = dest_x - start_x
    y_diff = dest_y - start_y
    angle = math.atan2(y_diff, x_diff)

    size = max(self.player_sprite.width,
self.player_sprite.height) / 2

    bullet.center_x += size * math.cos(angle)
    bullet.center_y += size * math.sin(angle)

    bullet.angle = math.degrees(angle)


    bullet_gravity = (0, -BULLET_GRAVITY)

    self.physics_engine.add_sprite(bullet,
                                   mass=BULLET_MASS,
                                   damping=1.0,
                                   friction=0.6,
                                   collision_type="bullet",
                                   gravity=bullet_gravity,
                                   elasticity=0.9)

    force = (BULLET_MOVE_FORCE, 0)
    self.physics_engine.apply_force(bullet, force)
```

Далее для удаления патрона и просчета коллизий создадим класс патрона

```
class BulletSprite(arcade.SpriteSolidColor):
    def pymunk_moved(self, physics_engine, dx, dy, d_angle):
        if self.center_y < -100:
            self.remove_from_sprite_lists()
```

Для удаления нашего патрона внутри метода setup, создадим два метода, которые вызываются при коллизии объектов

```
def wall_hit_handler(bullet_sprite, _wall_sprite, _arbiter,
_space, _data):
    bullet_sprite.remove_from_sprite_lists()
```

```
self.physics_engine.add_collision_handler("bullet", "wall",
post_handler=wall_hit_handler)

def item_hit_handler(bullet_sprite, item_sprite, _arbiter,
_space, _data):
   bullet_sprite.remove_from_sprite_lists()
   item_sprite.remove_from_sprite_lists()

self.physics_engine.add_collision_handler("bullet", "item",
post_handler=item_hit_handler)
```