Software
Engineering
School

Курсы программирования для
взрослых и детей

Делаем игры на Pygame

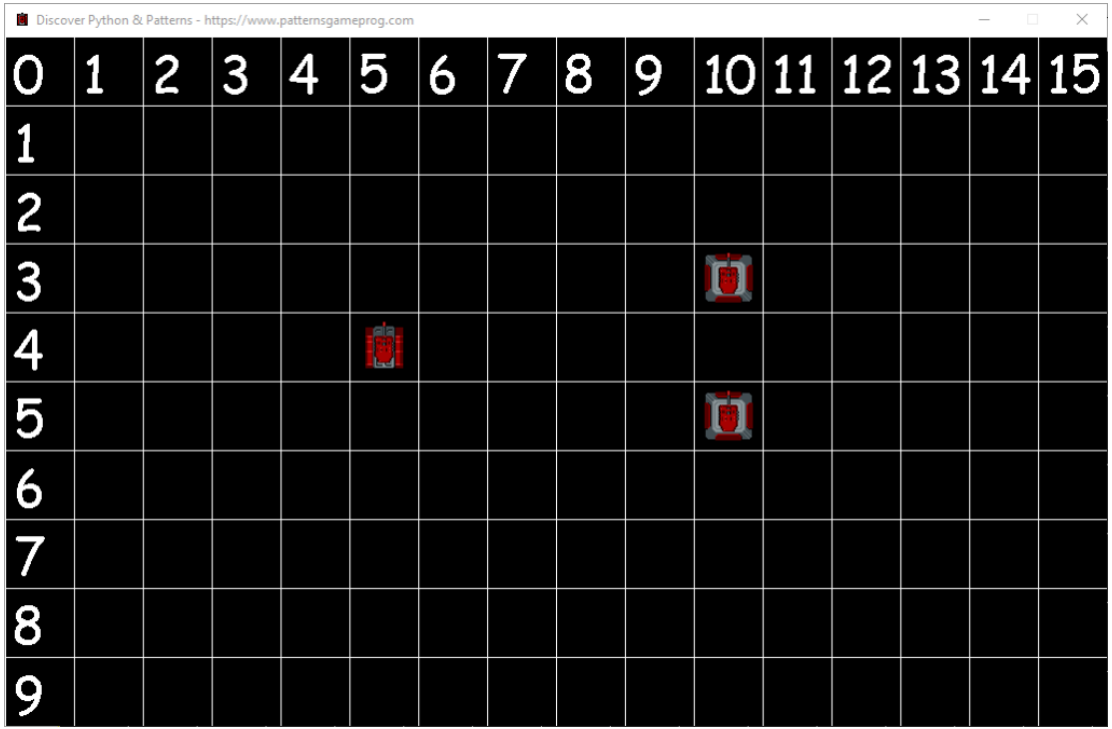
Чем мы сегодня займемся

- Узнаем про **тайловую графику**
- Качественно улучшим **организацию** нашего кода
- Нарисуем **карту игры**

Наша задача до конца модуля

Разработать 2D танковый шутер

Координатная сетка



Создаем модуль с настройками

Для удобной организации кода, вместо хранения данных о размерах поля и значении FPS внутри класса Game, создадим отдельный файл settings.py в котором будет класс Settings, содержащий информацию об игре.

Создаем модуль с настройками

```
class Settings():
    ROWS_COUNT = 5
    COLS_COUNT = 5

    MIN_ROW_INDEX = 0
    MAX_ROW_INDEX = ROWS_COUNT - 1
    MIN_COL_INDEX = 0
    MAX_COL_INDEX = COLS_COUNT - 1

    CELL_SIZE = 128
    PIXEL_HEIGHT = ROWS_COUNT * CELL_SIZE
    PIXEL_WIDTH = COLS_COUNT * CELL_SIZE

    FPS = 60
```

Координатная сетка

Создадим специальный класс `Direction` (в отдельном файле direction.py), определяющий направления движения для танка:

```
from enum import Enum


class Direction(Enum):
    UP = 0
    DOWN = 1
    LEFT = 2
    RIGHT = 3
```

Позиция в сетке

Для удобства также создадим класс CellPos, определяющий позицию игрового объекта (не только танка) на экране:

```
from direction import Direction
from settings import Settings


class CellPos():

    def __init__(self, col, row):
        self.col = col
        self.row = row
```

Позиция в сетке

Напишем метод get_neighbor, позволяющий получить соседнюю позицию для клетки при помощи направления движения (класса Direction). В случае если соседа нет, метод вернет None

```python
def get_neighbor(self, direction):
    neighbor_col = self.col
    neighbor_row = self.row

    if direction == Direction.UP:
        neighbor_row -= 1
    elif direction == Direction.DOWN:
        neighbor_row += 1
    elif direction == Direction.LEFT:
        neighbor_col -= 1
    elif direction == Direction.RIGHT:
        neighbor_col += 1

    # Соседа в заданном направлении нет
    if not CellPos._is_valid_pos(neighbor_col, neighbor_row):
        return None

    return CellPos(neighbor_col, neighbor_row)
```

Позиция в сетке

Чтобы убедится, что мы действительно получили правильные координаты для клетки, необходимо реализовать функцию проверяющую, что мы не вышли из заданного диапазона

```
@staticmethod
def _is_valid_pos(col, row):
    return (Settings.MIN_ROW_INDEX <= row <= Settings.MAX_ROW_INDEX) and \
        (Settings.MIN_COL_INDEX <= col <= Settings.MAX_COL_INDEX)
```

Позиция в сетке

В процессе игры нам понадобиться сравнивать различные позиции между собой, чтобы убедится, например, что эти позиции не заняты для движения танка, поэтому добавим магический метод __eq__:

```
def __eq__(self, other_position):
    return  (self.col == other_position.col) and (self.row == other_position.row)
```

Позиция в сетке

Наконец, чтобы иметь возможность переходить от нашей системы координат к системе координат Pygame, состоящей только из пикселей, нам понадобится функция position_to_pixel:

```
@staticmethod
def position_to_pixel(position):
    if isinstance(position, tuple):
        col = position[0]
        row = position[1]
    else:
        col = position.col
        row = position.row
    return (Settings.CELL_SIZE * col, Settings.CELL_SIZE * row)
```

Тайловая графика

Тайлы (от английского tile, "плитка") — это небольшие изображения одинаковых размеров, их которых как из фрагментов складывается участок уровня: трава, дорога, стены и так далее. Обычно тайлы плоские, но есть и исключения. Чаще всего тайлы для игры хранятся в одном файле.

Тайловая графика

Чаще всего каждый тайл в наборе тайлов имеет один и тот же размер, в нашем случае 128 на 128 пикслей. Создадим файл texture.py, в котором определим класс TileTexture для работы с тайловой графикой

```
import pygame
from pygame import Rect

from cell_pos import CellPos


class TileTexture():

    def __init__(self, texture_file: str, tile_size: int):
        self.texture = pygame.image.load(texture_file).convert_alpha()
        self.tile_size = tile_size
```

Тайловая графика

Для извлечения тайлов из тайловых карт мы реализуем метод get, позволяющий извлечь тайл по заданной позиции (в нашей системе координат) и вернуть его (возможно, повернув на какое-то кол-во градусов – зачем это нужно, увидим дальше):

```
def get(self, tile_pos=(0, 0), angle=None):
    x, y = CellPos.position_to_pixel(tile_pos)
    tile_area = Rect(x, y, self.tile_size, self.tile_size)
    tile = self.texture.subsurface(tile_area)
    if angle is not None:
        tile = pygame.transform.rotate(tile, angle)
    return tile
```

Обновленный танк

Улучшим организацию кода внутри класса танка, создав отдельный файл tank.py и написав в нем заготовку класса – теперь танк будет хранить свою позицию на поле и ссылку на само поле, где он хранится.

Обновленный танк

```python
import pygame

from direction import Direction
from texture import TileTexture
from settings import Settings


class Tank():

    def __init__(self):
        self.field = None
        self.position = None
        self.body_texture_file = 'tanks_images/blue/body.png'
        self.turret_texture_file = 'tanks_images/blue/turret2.png'
        self.body_texture = TileTexture(self.body_texture_file, Settings.CELL_SIZE)
        self.body_image = self.body_texture.get()
        self.turret_texture = TileTexture(self.turret_texture_file, Settings.CELL_SIZE)
        self.turret_image = self.turret_texture.get()
```

Обновленный танк

Чтобы не требовать от пользователя связать танк с полем немедленно при создании танка, мы сделаем сеттеры, благодаря которым танк можно будет привязать к полю уже после создания (зачем это нужно увидим далее):

```
def set_field(self, field):
    self.field = field

def set_position(self, position):
    self.position = position
```

Обновленный танк

Добавим в танк метод move, передвигающий его:

```
def move(self, direction):
    self._rotate(direction)

    if not self.field.can_move_to(self.position, direction):
        return False

    self.position = self.position.get_neighbor(direction)
    return True
```

Обновленный танк

move пытается сдвинуть танк в заданном направлении, сначала поворачивая дуло и тело танка в заданную сторону (при помощи метода _rotate), после чего спрашивая поле (которое мы ещё не реализовали), можно ли передвинуться с клетки, занимаемой танком, в заданном направлении, и если да, то мы получаем соседа клетки в направлении, при помощи метода get_neighbor.

Обновленный танк

Реализуем метод _rotate, поворачивающий танк в заданном направлении:

```
def _rotate(self, direction):
    if direction == Direction.UP:
        angle = 0
    elif direction == Direction.DOWN:
        angle = 180
    elif direction == Direction.LEFT:
        angle = 90
    elif direction == Direction.RIGHT:
        angle = -90

    self.body_image = self.body_texture.get(angle=angle)
    self.turret_image = self.turret_texture.get(angle=angle)
```

Обновленный танк

Последний метод, который мы должны реализовать у танка – render. В этот раз мы не будет рисовать танк где-либо, мы всего лишь создадим поверхность, на которой нарисуем тело танка и дуло, после чего вернем результат:

```
def render(self):
    tank = pygame.Surface((Settings.CELL_SIZE, Settings.CELL_SIZE), pygame.SRCALPHA)
    tank.blit(self.body_image, (0, 0))
    tank.blit(self.turret_image, (0, 0))
    return tank
```

Обновленный танк

move пытается сдвинуть танк в заданном направлении, сначала поворачивая дуло и тело танка в заданную сторону, после чего спрашивая поле (которое мы ещё не реализовали), можно ли передвинуться с клетки, занимаемой танком, в заданном направлении, и если да, то мы получаем соседа клетки в направлении, при помощи метода get_neighbor

Игровое поле

Игровое поле (файл – field.py) представляет из себя контейнер, в котором хранятся игровые юниты (пока что это только танк) и разнообразные объекты, принадлежащие полю (трава, стены, вражеские башни)

```
import pygame

from settings import Settings
from ground import *
from obstacle import Wall


class Field():

    def __init__(self):
        self.ground = None
        self.walls = None
        self.units = []

        self._init_field()
```

Игровое поле

Реализуем у поля метод can_move_to, позволяющий узнавать, можно при переместиться, находятся на такой-то позиции в заданном направлении:

```
def can_move_to(self, position, direction):
    neighbor = position.get_neighbor(direction)
    return (neighbor is not None) and (not self._is_occupied(neighbor))
```

Игровое поле

Посмотрим, как можно реализовать функцию _is_occupied, проверяющую, занята ли позиция каким-либо юнитом или предметом (пока что стенами) на поле:

```
def _is_occupied(self, position):
    for unit in self.units:
        if unit.position == position:
            return True
    for wall in self.walls:
        if wall.position == position:
            return True
    return False
```

Игровое поле

Чтобы установить танк на поле, нам потребуется метод put_at, помещающий игровой юнит (танк) в заданную позицию на поле

```
def put_at(self, new_unit, position):
    if self._is_occupied(position):
        return False

    new_unit.set_field(self)
    new_unit.set_position(position)
    self.units.append(new_unit)
    return True
```

Игровое поле

По аналогии с танком, мы не будет рисовать поле на главном окне напрямую – мы просто создадим поверхность, в которой будет отрисовано поле и вернем её, а что с ней делать решить уже тот, кто вызывал метод render:

```
def render(self):
    field = pygame.Surface((Settings.PIXEL_WIDTH, Settings.PIXEL_HEIGHT))

    self.render_ground(field)
    self.render_walls(field)
    self.render_units(field)

    return field
```

Игровое поле

Отрисовка стен и игровых юнитов на поле:

```
def render_walls(self, field):
    for wall in self.walls:
        wall_pixel_pos = CellPos.position_to_pixel(wall.position)
        field.blit(wall.render(), wall_pixel_pos)

def render_units(self, field):
    for unit in self.units:
        unit_pixel_pos = CellPos.position_to_pixel(unit.position)
        field.blit(unit.render(), unit_pixel_pos)
```

Игровое поле

Немного сложнее реализуется отрисовка травы и дорог на карте, для этого нам необходимо посмотреть, в какой форме они задаются на поле.

Игровое поле

Немного сложнее реализуется отрисовка травы и дорог на карте, для этого нам необходимо посмотреть, в какой форме они задаются на поле.

Игровое поле

```
def _init_field(self):
    self.ground = [
        [Grass(), Road(), Grass(), Grass(), Bush()],
        [Grass(), Road(), Grass(), Bush(), Grass()],
        [Road(angle=-90), TripleRoad(angle=180), Grass(), Grass(), Grass()],
        [Grass(), Road(), Grass(), Grass(), Grass()],
        [Bush(), Road(), Bush(), Grass(), Grass()],
    ]

    for row in range(Settings.ROWS_COUNT):
        for col in range(Settings.COLS_COUNT):
            self.ground[row][col].set_position(CellPos(col, row))

    self.walls = [
        Wall(position=CellPos(0, 3)), Wall(position=CellPos(1, 3), angle=180)
    ]
```

Игровое поле

Как видно, «земля» (ground) представляет из себя двумерный список, содержащий расположение травы и дорог. Чтобы отрисовать этот список, нам необходимо два цикла:

```
def render_ground(self, field):
    for i in range(len(self.ground)):
        for j in range(len(self.ground[i])):
            ground_obj = self.ground[i][j]
            ground_pixel_pos = CellPos.position_to_pixel(ground_obj.position)
            field.blit(ground_obj.render(), ground_pixel_pos)
```

Игровой объект

Каждый игровой объект (не являющий игровым юнитов) обладает следующими характеристиками: текстура, размеры и позиция на экране. Чтобы отрисовать поле, мы создадим класс обобщенного «игрового объекта», от которого унаследуем все общие для игровых объектов свойства.

Игровой объект

```
from texture import TileTexture
from settings import Settings


class GameObject():

    def __init__(self, texture_file, tile_pos, angle=None, position=None):
        self.texture = TileTexture(texture_file, Settings.CELL_SIZE)
        self.tile_pos = tile_pos
        self.angle = angle
        self.position = position

    def set_position(self, new_position):
        self.position = new_position

    def render(self):
        return self.texture.get(self.tile_pos, self.angle)
```

Трава и дороги

Реализуем класс Ground, являющийся базовым классом для травы и дорого различных типов:

```
from cell_pos import CellPos
from game_object import GameObject


class Ground(GameObject):

    def __init__(self, tile_pos, angle=None, position=None):
        super().__init__('tanks_images/terrain.png', tile_pos, angle, position)
```

Трава и дороги

Реализуем класс Ground, являющийся базовым классом для травы и дорого различных типов:

```
from cell_pos import CellPos
from game_object import GameObject


class Ground(GameObject):

    def __init__(self, tile_pos, angle=None, position=None):
        super().__init__('tanks_images/terrain.png', tile_pos, angle, position)
```

Трава и дороги

```
class Grass(Ground):

    def __init__(self, angle=None):
        super().__init__(tile_pos=CellPos(0, 4), angle=angle)


class Road(Ground):

    def __init__(self, angle=None):
        super().__init__(tile_pos=CellPos(3, 4), angle=angle)
```

Трава и дороги

```
class RoadCorner(Ground):

    def __init__(self, angle=None):
        super().__init__(tile_pos=CellPos(4, 4), angle=angle)


class TripleRoad(Ground):

    def __init__(self, angle=None):
        super().__init__(tile_pos=CellPos(5, 4), angle=angle)


class Bush(Ground):

    def __init__(self, angle=None):
        super().__init__(tile_pos=CellPos(2, 4), angle=angle)
```

Стены

Для создания стен нам потребуется базовый класс Obstacle, хранящий в себе адрес текстур препятствий и производный класс Wall, представляющий собой стену:

```
from cell_pos import CellPos
from game_object import GameObject


class Obstacle(GameObject):

    def __init__(self, tile_pos, angle=None, position=None):
        super().__init__('tanks_images/blue/towers_walls_blank.png', tile_pos, angle, position)


class Wall(Obstacle):

    def __init__(self, position=None, angle=None):
        super().__init__(tile_pos=CellPos(1, 4), angle=angle, position=position)
```

Собираем все вместе

```
import pygame

from settings import Settings
from direction import Direction
from cell_pos import CellPos
from tank import Tank
from field import Field


class Game():

    def __init__(self):
        pygame.init()

        self.main_window = pygame.display.set_mode((Settings.PIXEL_WIDTH, Settings.PIXEL_HEIGHT))
        self.clock = pygame.time.Clock()
        self.running = True

        self.field = Field()
        self.tank = Tank()
        self.field.put_at(self.tank, CellPos(0, 0))

        self.direction = None
```

Собираем все вместе

```
def process_input(self):
    self.direction = None

    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            self.running = False
            break
        elif event.type == pygame.KEYDOWN:
            if event.key == pygame.K_RIGHT:
                self.direction = Direction.RIGHT
            elif event.key == pygame.K_LEFT:
                self.direction = Direction.LEFT
            elif event.key == pygame.K_DOWN:
                self.direction = Direction.DOWN
            elif event.key == pygame.K_UP:
                self.direction = Direction.UP
```

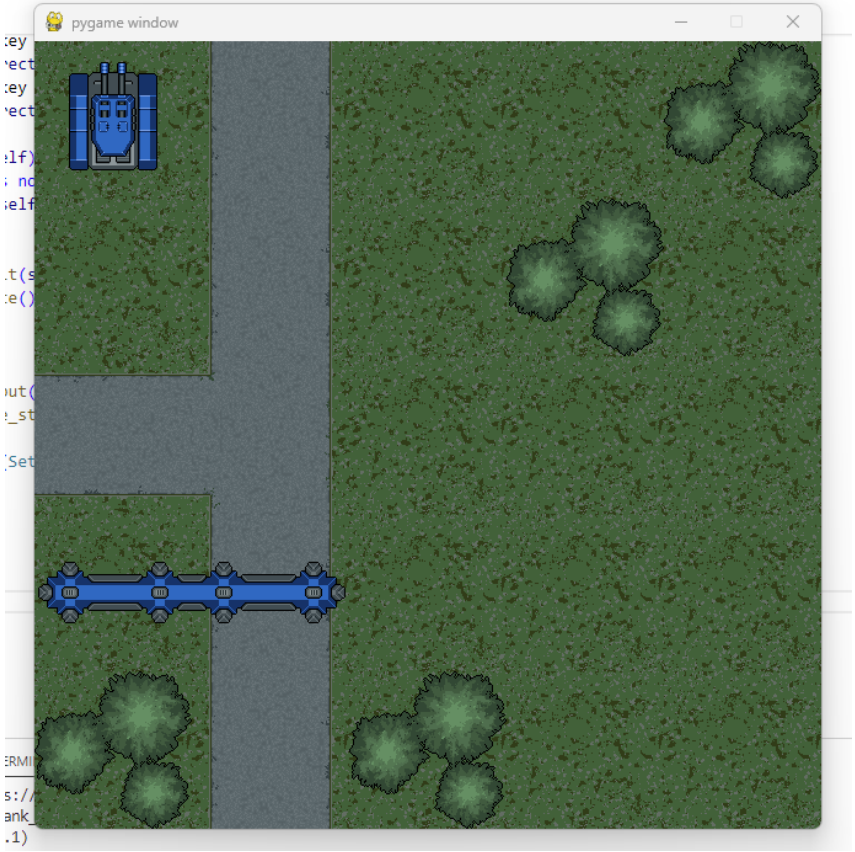
Собираем все вместе

```
def update_game_state(self):
    if self.direction is not None:
        self.tank.move(self.direction)

def render(self):
    self.main_window.blit(self.field.render(), (0, 0))
    pygame.display.update()

def game_loop(self):
    while self.running:
        self.process_input()
        self.update_game_state()
        self.render()
        self.clock.tick(Settings.FPS)

    pygame.quit()
```

Запуск!

```
game = Game()
game.game_loop()
```

Результат



Домашнее задания

Реализуйте собственную игровую обстановку с необычным расположением дорог и стен

Наш репозиторий

https://github.com/samedit66/pygame_2024/tree/main