Министерство образования и молодежной политики Свердловской области



ГАПОУ СО «Екатеринбургский колледж транспортного строительства»

Отчёт по программе «**Pacman в Python**»

Выполнил: Юнусов Руслан Александрович

Группа: ПР-32

Преподаватель: Шевцова Вера Дмитриевна

2024

**Задание:** Разработать игру Pacman в Python

**Входные и выходные данные**

**Direction – класс кнопок передвижения  
game\_renderer – класс рендеринга игры  
GameObject – класс родитель всех объектов  
Ghost – класс призраков  
Coockie – класс печенек  
PacmanGameController – класс логики игры  
Pathfinder – класс координат  
Hero – класс главного героя  
MovbleObject – класс передвижения**

**Листинг программы (если есть)**

**Main.py**def translate\_screen\_to\_maze(in\_coords, in\_size=32):

    return int(in\_coords[0] / in\_size), int(in\_coords[1] / in\_size)

def translate\_maze\_to\_screen(in\_coords, in\_size=32):

    return in\_coords[0] \* in\_size, in\_coords[1] \* in\_size

if \_\_name\_\_ == "\_\_main\_\_":

    unified\_size = 32

    pacman\_game = PacmanGameController()

    size = pacman\_game.size

    game\_renderer = GameRenderer(size[0] \* unified\_size, size[1] \* unified\_size)

    for y, row in enumerate(pacman\_game.numpy\_maze):

        for x, column in enumerate(row):

            if column == 0:

                game\_renderer.add\_wall(Wall(game\_renderer, x, y, unified\_size))

    for cookie\_space in pacman\_game.cookie\_spaces:

        translated = translate\_maze\_to\_screen(cookie\_space)

        cookie = Cookie(game\_renderer, translated[0] + unified\_size / 2, translated[1] + unified\_size / 2)

        game\_renderer.add\_cookie(cookie)

    for i, ghost\_spawn in enumerate(pacman\_game.ghost\_spawns):

        translated = translate\_maze\_to\_screen(ghost\_spawn)

        ghost = Ghost(game\_renderer, translated[0], translated[1], unified\_size, pacman\_game,

                      pacman\_game.ghost\_colors[i % 4])

        game\_renderer.add\_game\_object(ghost)

    pacman = Hero(game\_renderer, unified\_size, unified\_size, unified\_size)

    game\_renderer.add\_hero(pacman)

    game\_renderer.tick(120)

**PacmanGameController**

class PacmanGameController:

    def \_\_init\_\_(self):

        self.ascii\_maze = [

            "XXXXXXXXXXXXXXXXXXXXXXXXXXXX",

            "XP           XX            X",

            "X XXXX XXXXX XX XXXXX XXXX X",

            "X XXXX XXXXX XX XXXXX XXXX X",

            "X XXXX XXXXX XX XXXXX XXXX X",

            "X                          X",

            "X XXXX XX XXXXXXXX XX XXXX X",

            "X XXXX XX XXXXXXXX XX XXXX X",

            "X      XX    XX    XX      X",

            "XXXXXX XXXXX XX XXXXX XXXXXX",

            "XXXXXX XXXXX XX XXXXX XXXXXX",

            "XXXXXX XX          XX XXXXXX",

            "XXXXXX XX XXXXXXXX XX XXXXXX",

            "XXXXXX XX X   G  X XX XXXXXX",

            "          X G    X          ",

            "XXXXXX XX X   G  X XX XXXXXX",

            "XXXXXX XX XXXXXXXX XX XXXXXX",

            "XXXXXX XX          XX XXXXXX",

            "XXXXXX XX XXXXXXXX XX XXXXXX",

            "XXXXXX XX XXXXXXXX XX XXXXXX",

            "X            XX            X",

            "X XXXX XXXXX XX XXXXX XXXX X",

            "X XXXX XXXXX XX XXXXX XXXX X",

            "X   XX       G        XX   X",

            "XXX XX XX XXXXXXXX XX XX XXX",

            "XXX XX XX XXXXXXXX XX XX XXX",

            "X      XX    XX    XX      X",

            "X XXXXXXXXXX XX XXXXXXXXXX X",

            "X XXXXXXXXXX XX XXXXXXXXXX X",

            "X                          X",

            "XXXXXXXXXXXXXXXXXXXXXXXXXXXX",

        ]

        self.numpy\_maze = []

        self.cookie\_spaces = []

        self.reachable\_spaces = []

        self.ghost\_spawns = []

        self.ghost\_colors = [

            (255, 184, 255),

            (255, 0, 20),

            (0, 255, 255),

            (255, 184, 82)

        ]

        self.size = (0, 0)

        self.convert\_maze\_to\_numpy()

        self.p = Pathfinder(self.numpy\_maze)

    def request\_new\_random\_path(self, in\_ghost: Ghost):

        random\_space = random.choice(self.reachable\_spaces)

        current\_maze\_coord = translate\_screen\_to\_maze(in\_ghost.get\_position())

        path = self.p.get\_path(current\_maze\_coord[1], current\_maze\_coord[0], random\_space[1],

                               random\_space[0])

        test\_path = [translate\_maze\_to\_screen(item) for item in path]

        in\_ghost.set\_new\_path(test\_path)

    def convert\_maze\_to\_numpy(self):

        for x, row in enumerate(self.ascii\_maze):

            self.size = (len(row), x + 1)

            binary\_row = []

            for y, column in enumerate(row):

                if column == "G":

                    self.ghost\_spawns.append((y, x))

                if column == "X":

                    binary\_row.append(0)

                else:

                    binary\_row.append(1)

                    self.cookie\_spaces.append((y, x))

                    self.reachable\_spaces.append((y, x))

            self.numpy\_maze.append(binary\_row)

**Pathfinder.py**

class Pathfinder:

    def \_\_init\_\_(self, in\_arr):

        cost = np.array(in\_arr, dtype=np.bool\_).tolist()

        self.pf = tcod.path.AStar(cost=cost, diagonal=0)

    def get\_path(self, from\_x, from\_y, to\_x, to\_y) -> object:

        res = self.pf.get\_path(from\_x, from\_y, to\_x, to\_y)

        return [(sub[1], sub[0]) for sub in res]

**Cookie.py**

class Cookie(GameObject):

    def \_\_init\_\_(self, in\_surface, x, y):

        super().\_\_init\_\_(in\_surface, x, y, 4, (255, 255, 0), True)

**Ghost.py**

class Ghost(MovableObject):

    def \_\_init\_\_(self, in\_surface, x, y, in\_size: int, in\_game\_controller, in\_color=(255, 0, 0)):

        super().\_\_init\_\_(in\_surface, x, y, in\_size, in\_color, False)

        self.game\_controller = in\_game\_controller

    def reached\_target(self):

        if (self.x, self.y) == self.next\_target:

            self.next\_target = self.get\_next\_location()

        self.current\_direction = self.calculate\_direction\_to\_next\_target()

    def set\_new\_path(self, in\_path):

        for item in in\_path:

            self.location\_queue.append(item)

        self.next\_target = self.get\_next\_location()

    def calculate\_direction\_to\_next\_target(self) -> Direction:

        if self.next\_target is None:

            self.game\_controller.request\_new\_random\_path(self)

            return Direction.NONE

        diff\_x = self.next\_target[0] - self.x

        diff\_y = self.next\_target[1] - self.y

        if diff\_x == 0:

            return Direction.DOWN if diff\_y > 0 else Direction.UP

        if diff\_y == 0:

            return Direction.LEFT if diff\_x < 0 else Direction.RIGHT

        self.game\_controller.request\_new\_random\_path(self)

        return Direction.NONE

    def automatic\_move(self, in\_direction: Direction):

        if in\_direction == Direction.UP:

            self.set\_position(self.x, self.y - 1)

        elif in\_direction == Direction.DOWN:

            self.set\_position(self.x, self.y + 1)

        elif in\_direction == Direction.LEFT:

            self.set\_position(self.x - 1, self.y)

        elif in\_direction == Direction.RIGHT:

            self.set\_position(self.x + 1, self.y)

**Hero.py**

class Hero(MovableObject):

    def \_\_init\_\_(self, in\_surface, x, y, in\_size: int):

        super().\_\_init\_\_(in\_surface, x, y, in\_size, (255, 255, 0), False)

        self.last\_non\_colliding\_position = (0, 0)

    def tick(self):

        # TELEPORT

        if self.x < 0:

            self.x = self.\_renderer.\_width

        if self.x > self.\_renderer.\_width:

            self.x = 0

        self.last\_non\_colliding\_position = self.get\_position()

        if self.check\_collision\_in\_direction(self.direction\_buffer)[0]:

            self.automatic\_move(self.current\_direction)

        else:

            self.automatic\_move(self.direction\_buffer)

            self.current\_direction = self.direction\_buffer

        if self.collides\_with\_wall((self.x, self.y)):

            self.set\_position(self.last\_non\_colliding\_position[0], self.last\_non\_colliding\_position[1])

        self.handle\_cookie\_pickup()

    def automatic\_move(self, in\_direction: Direction):

        collision\_result = self.check\_collision\_in\_direction(in\_direction)

        desired\_position\_collides = collision\_result[0]

        if not desired\_position\_collides:

            self.last\_working\_direction = self.current\_direction

            desired\_position = collision\_result[1]

            self.set\_position(desired\_position[0], desired\_position[1])

        else:

            self.current\_direction = self.last\_working\_direction

    def handle\_cookie\_pickup(self):

        collision\_rect = pygame.Rect(self.x, self.y, self.\_size, self.\_size)

        cookies = self.\_renderer.get\_cookies()

        game\_objects = self.\_renderer.get\_game\_objects()

        for cookie in cookies:

            collides = collision\_rect.colliderect(cookie.get\_shape())

            if collides and cookie in game\_objects:

                game\_objects.remove(cookie)

    def draw(self):

        half\_size = self.\_size / 2

        pygame.draw.circle(self.\_surface, self.\_color, (self.x + half\_size, self.y + half\_size), half\_size)

**MovableObject.py**

class MovableObject(GameObject):

    def \_\_init\_\_(self, in\_surface, x, y, in\_size: int, in\_color=(255, 0, 0), is\_circle: bool = False):

        super().\_\_init\_\_(in\_surface, x, y, in\_size, in\_color, is\_circle)

        self.current\_direction = Direction.NONE

        self.direction\_buffer = Direction.NONE

        self.last\_working\_direction = Direction.NONE

        self.location\_queue = []

        self.next\_target = None

    def get\_next\_location(self):

        return None if len(self.location\_queue) == 0 else self.location\_queue.pop(0)

    def set\_direction(self, in\_direction):

        self.current\_direction = in\_direction

        self.direction\_buffer = in\_direction

    def collides\_with\_wall(self, in\_position):

        collision\_rect = pygame.Rect(in\_position[0], in\_position[1], self.\_size, self.\_size)

        collides = False

        walls = self.\_renderer.get\_walls()

        for wall in walls:

            collides = collision\_rect.colliderect(wall.get\_shape())

            if collides: break

        return collides

    def check\_collision\_in\_direction(self, in\_direction: Direction):

        desired\_position = (0, 0)

        if in\_direction == Direction.NONE: return False, desired\_position

        if in\_direction == Direction.UP:

            desired\_position = (self.x, self.y - 1)

        elif in\_direction == Direction.DOWN:

            desired\_position = (self.x, self.y + 1)

        elif in\_direction == Direction.LEFT:

            desired\_position = (self.x - 1, self.y)

        elif in\_direction == Direction.RIGHT:

            desired\_position = (self.x + 1, self.y)

        return self.collides\_with\_wall(desired\_position), desired\_position

    def automatic\_move(self, in\_direction: Direction):

        pass

    def tick(self):

        self.reached\_target()

        self.automatic\_move(self.current\_direction)

    def reached\_target(self):

        pass

**GameRenderer.py**

class GameRenderer:

    def \_\_init\_\_(self, in\_width: int, in\_height: int):

        pygame.init()

        self.\_width = in\_width

        self.\_height = in\_height

        self.\_screen = pygame.display.set\_mode((in\_width, in\_height))

        pygame.display.set\_caption('Pacman')

        self.\_clock = pygame.time.Clock()

        self.\_done = False

        self.\_game\_objects = []

        self.\_walls = []

        self.\_cookies = []

        self.\_hero: Hero = None

    def tick(self, in\_fps: int):

        black = (0, 0, 0)

        while not self.\_done:

            for game\_object in self.\_game\_objects:

                game\_object.tick()

                game\_object.draw()

            pygame.display.flip()

            self.\_clock.tick(in\_fps)

            self.\_screen.fill(black)

            self.\_handle\_events()

        print("Game over")

    def add\_game\_object(self, obj: GameObject):

        self.\_game\_objects.append(obj)

    def add\_cookie(self, obj: GameObject):

        self.\_game\_objects.append(obj)

        self.\_cookies.append(obj)

    def add\_wall(self, obj: Wall):

        self.add\_game\_object(obj)

        self.\_walls.append(obj)

    def get\_walls(self):

        return self.\_walls

    def get\_cookies(self):

        return self.\_cookies

    def get\_game\_objects(self):

        return self.\_game\_objects

    def add\_hero(self, in\_hero):

        self.add\_game\_object(in\_hero)

        self.\_hero = in\_hero

    def \_handle\_events(self):

        for event in pygame.event.get():

            if event.type == pygame.QUIT:

                self.\_done = True

        pressed = pygame.key.get\_pressed()

        if pressed[pygame.K\_UP]:

            self.\_hero.set\_direction(Direction.UP)

        elif pressed[pygame.K\_LEFT]:

            self.\_hero.set\_direction(Direction.LEFT)

        elif pressed[pygame.K\_DOWN]:

            self.\_hero.set\_direction(Direction.DOWN)

        elif pressed[pygame.K\_RIGHT]:

            self.\_hero.set\_direction(Direction.RIGHT)

**wall.py**

class Wall(GameObject):

    def \_\_init\_\_(self, in\_surface, x, y, in\_size: int, in\_color=(0, 0, 255)):

        super().\_\_init\_\_(in\_surface, x \* in\_size, y \* in\_size, in\_size, in\_color)

**GameObject.py**

class GameObject:

    def \_\_init\_\_(self, in\_surface, x, y,

                 in\_size: int, in\_color=(255, 0, 0),

                 is\_circle: bool = False):

        self.\_size = in\_size

        self.\_renderer: GameRenderer = in\_surface

        self.\_surface = in\_surface.\_screen

        self.y = y

        self.x = x

        self.\_color = in\_color

        self.\_circle = is\_circle

        self.\_shape = pygame.Rect(self.x, self.y, in\_size, in\_size)

    def draw(self):

        if self.\_circle:

            pygame.draw.circle(self.\_surface,

                               self.\_color,

                               (self.x, self.y),

                               self.\_size)

        else:

            rect\_object = pygame.Rect(self.x, self.y, self.\_size, self.\_size)

            pygame.draw.rect(self.\_surface,

                             self.\_color,

                             rect\_object,

                             border\_radius=4)

    def tick(self):

        pass

    def get\_shape(self):

        return self.\_shape

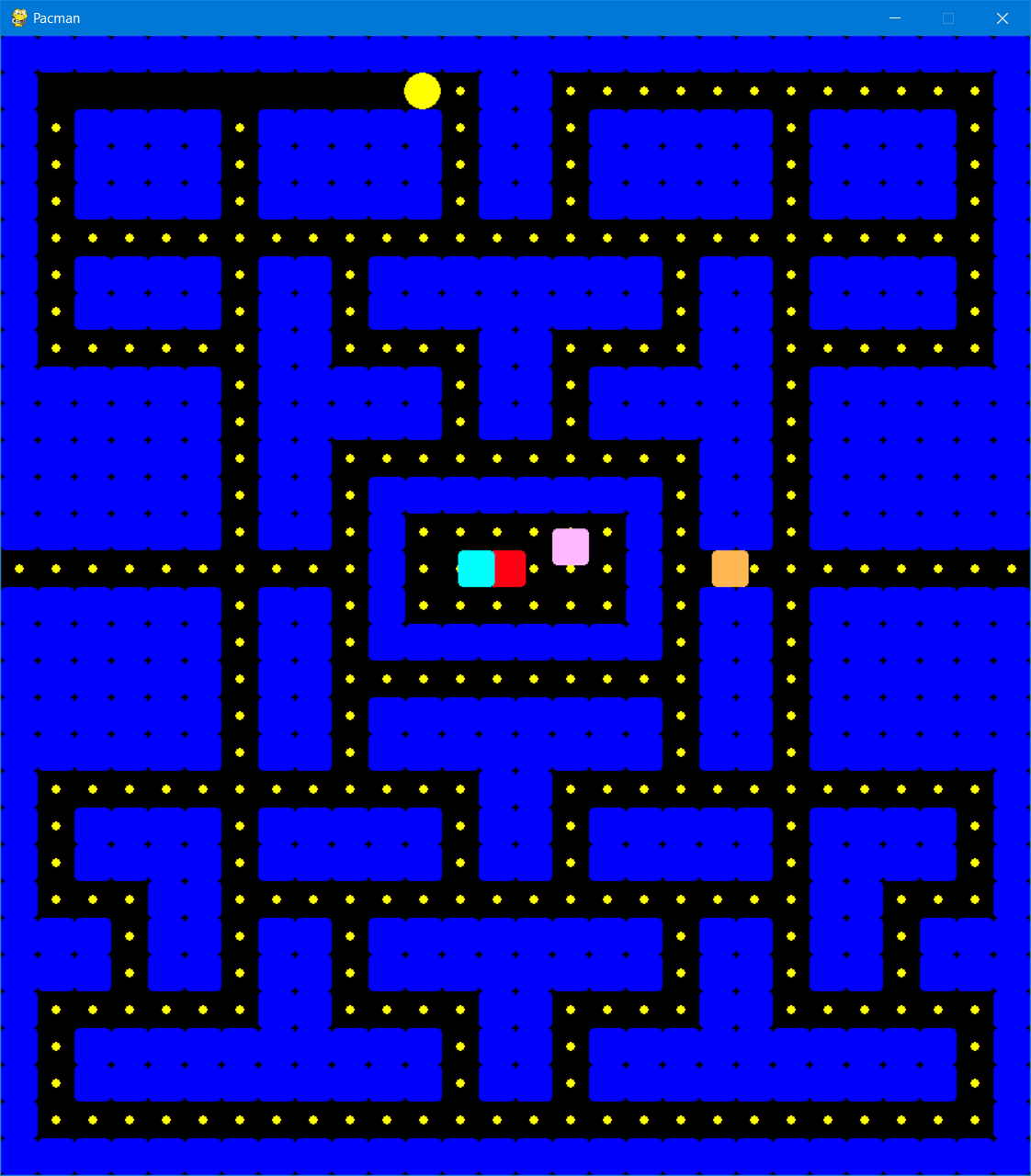
    def set\_position(self, in\_x, in\_y):

        self.x = in\_x

        self.y = in\_y

    def get\_position(self):

        return (self.x, self.y)

**Скрин-шоты**

**Вывод**

Я укрепил знания по языку программирования Python, и изучил несколько полезных для меня библиотек.