Министерство образования и молодежной политики Свердловской области



ГАПОУ СО «Екатеринбургский колледж транспортного строительства»

Отчёт по программе «**PacMan**»

Выполнил: Калинин Алексей

Группа: ПР-32

Преподаватель: Шевцова В.Д.

2023

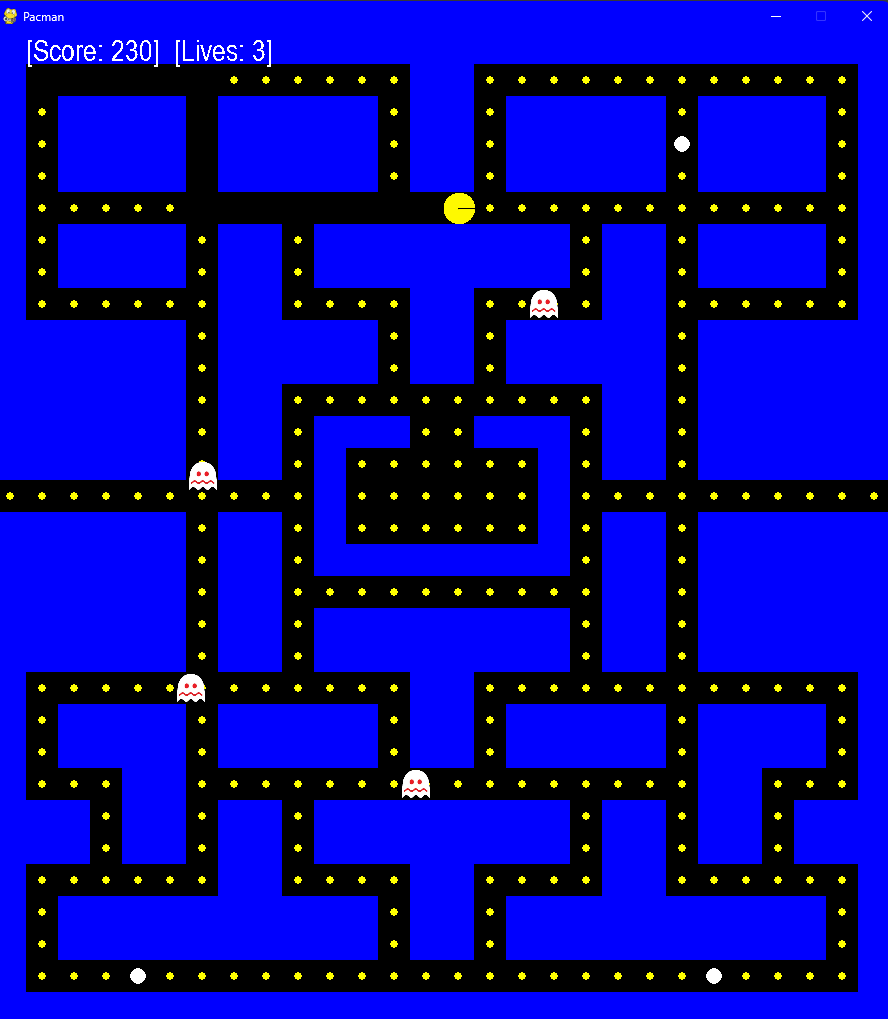
**Задание:**

Создать игру PacMan на python, реализовать стабильную работу игры

**Листинг программы**

import pygame  
import numpy as np  
import tcod  
import random  
from enum import Enum  
  
  
class Direction(Enum):  
 DOWN = -90  
 RIGHT = 0  
 UP = 90  
 LEFT = 180  
 NONE = 360  
  
  
class ScoreType(Enum):  
 COOKIE = 10  
 POWERUP = 50  
 GHOST = 400  
  
  
class GhostBehaviour(Enum):  
 CHASE = 1  
 SCATTER = 2  
  
  
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  
  
  
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=1)  
  
 def tick(self):  
 pass  
  
 def get\_shape(self):  
 return pygame.Rect(self.x, self.y, self.\_size, self.\_size)  
  
 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)  
  
  
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)  
  
  
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.\_won = False  
 self.\_game\_objects = []  
 self.\_walls = []  
 self.\_cookies = []  
 self.\_powerups = []  
 self.\_ghosts = []  
 self.\_hero: Hero = None  
 self.\_lives = 3  
 self.\_score = 0  
 self.\_score\_cookie\_pickup = 10  
 self.\_score\_ghost\_eaten = 400  
 self.\_score\_powerup\_pickup = 50  
 self.\_kokoro\_active = False # powerup, special ability  
 self.\_current\_mode = GhostBehaviour.SCATTER  
 self.\_mode\_switch\_event = pygame.USEREVENT + 1 # custom event  
 self.\_kokoro\_end\_event = pygame.USEREVENT + 2  
 self.\_pakupaku\_event = pygame.USEREVENT + 3  
 self.\_modes = [  
 (7, 20),  
 (7, 20),  
 (5, 20),  
 (5, 999999) # 'infinite' chase seconds  
 ]  
 self.\_current\_phase = 0  
  
 def tick(self, in\_fps: int):  
 black = (0, 0, 0)  
  
 self.handle\_mode\_switch()  
 pygame.time.set\_timer(self.\_pakupaku\_event, 200) # open close mouth  
 while not self.\_done:  
 for game\_object in self.\_game\_objects:  
 game\_object.tick()  
 game\_object.draw()  
  
 self.display\_text(f"[Score: {self.\_score}] [Lives: {self.\_lives}]")  
  
 if self.\_hero is None: self.display\_text("YOU DIED", (self.\_width / 2 - 256, self.\_height / 2 - 256), 100)  
 if self.get\_won(): self.display\_text("YOU WON", (self.\_width / 2 - 256, self.\_height / 2 - 256), 100)  
 pygame.display.flip()  
 self.\_clock.tick(in\_fps)  
 self.\_screen.fill(black)  
 self.\_handle\_events()  
  
 print("Game over")  
  
 def handle\_mode\_switch(self):  
 current\_phase\_timings = self.\_modes[self.\_current\_phase]  
 print(f"Current phase: {str(self.\_current\_phase)}, current\_phase\_timings: {str(current\_phase\_timings)}")  
 scatter\_timing = current\_phase\_timings[0]  
 chase\_timing = current\_phase\_timings[1]  
  
 if self.\_current\_mode == GhostBehaviour.CHASE:  
 self.\_current\_phase += 1  
 self.set\_current\_mode(GhostBehaviour.SCATTER)  
 else:  
 self.set\_current\_mode(GhostBehaviour.CHASE)  
  
 used\_timing = scatter\_timing if self.\_current\_mode == GhostBehaviour.SCATTER else chase\_timing  
 pygame.time.set\_timer(self.\_mode\_switch\_event, used\_timing \* 1000)  
  
 def start\_kokoro\_timeout(self):  
 pygame.time.set\_timer(self.\_kokoro\_end\_event, 15000) # 15s  
  
 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\_ghost(self, obj: GameObject):  
 self.\_game\_objects.append(obj)  
 self.\_ghosts.append(obj)  
  
 def add\_powerup(self, obj: GameObject):  
 self.\_game\_objects.append(obj)  
 self.\_powerups.append(obj)  
  
 def activate\_kokoro(self):  
 self.\_kokoro\_active = True  
 self.set\_current\_mode(GhostBehaviour.SCATTER)  
 self.start\_kokoro\_timeout()  
  
 def set\_won(self):  
 self.\_won = True  
  
 def get\_won(self):  
 return self.\_won  
  
 def add\_score(self, in\_score: ScoreType):  
 self.\_score += in\_score.value  
  
 def get\_hero\_position(self):  
 return self.\_hero.get\_position() if self.\_hero != None else (0, 0)  
  
 def set\_current\_mode(self, in\_mode: GhostBehaviour):  
 self.\_current\_mode = in\_mode  
  
 def get\_current\_mode(self):  
 return self.\_current\_mode  
  
 def end\_game(self):  
 if self.\_hero in self.\_game\_objects:  
 self.\_game\_objects.remove(self.\_hero)  
 self.\_hero = None  
  
 def kill\_pacman(self):  
 self.\_lives -= 1  
 self.\_hero.set\_position(32, 32)  
 self.\_hero.set\_direction(Direction.NONE)  
 if self.\_lives == 0: self.end\_game()  
  
 def display\_text(self, text, in\_position=(32, 0), in\_size=30):  
 font = pygame.font.SysFont('Arial', in\_size)  
 text\_surface = font.render(text, False, (255, 255, 255))  
 self.\_screen.blit(text\_surface, in\_position)  
  
 def is\_kokoro\_active(self):  
 return self.\_kokoro\_active  
  
 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\_ghosts(self):  
 return self.\_ghosts  
  
 def get\_powerups(self):  
 return self.\_powerups  
  
 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  
  
 if event.type == self.\_mode\_switch\_event:  
 self.handle\_mode\_switch()  
  
 if event.type == self.\_kokoro\_end\_event:  
 self.\_kokoro\_active = False  
  
 if event.type == self.\_pakupaku\_event:  
 if self.\_hero is None: break  
 self.\_hero.mouth\_open = not self.\_hero.mouth\_open  
  
 pressed = pygame.key.get\_pressed()  
 if self.\_hero is None: return  
 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)  
  
  
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  
 self.image = pygame.image.load('images/ghost.png')  
  
 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  
  
 def draw(self):  
 self.image = pygame.transform.scale(self.image, (32, 32))  
 self.\_surface.blit(self.image, self.get\_shape())  
  
  
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)  
 self.open = pygame.image.load("images/paku.png")  
 self.closed = pygame.image.load("images/man.png")  
 self.image = self.open  
 self.mouth\_open = True  
  
 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()  
 self.handle\_ghosts()  
  
 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()  
 powerups = self.\_renderer.get\_powerups()  
 game\_objects = self.\_renderer.get\_game\_objects()  
 cookie\_to\_remove = None  
 for cookie in cookies:  
 collides = collision\_rect.colliderect(cookie.get\_shape())  
 if collides and cookie in game\_objects:  
 game\_objects.remove(cookie)  
 self.\_renderer.add\_score(ScoreType.COOKIE)  
 cookie\_to\_remove = cookie  
  
 if cookie\_to\_remove is not None:  
 cookies.remove(cookie\_to\_remove)  
  
 if len(self.\_renderer.get\_cookies()) == 0:  
 self.\_renderer.set\_won()  
  
 for powerup in powerups:  
 collides = collision\_rect.colliderect(powerup.get\_shape())  
 if collides and powerup in game\_objects:  
 if not self.\_renderer.is\_kokoro\_active():  
 game\_objects.remove(powerup)  
 self.\_renderer.add\_score(ScoreType.POWERUP)  
 self.\_renderer.activate\_kokoro()  
  
 def handle\_ghosts(self):  
 collision\_rect = pygame.Rect(self.x, self.y, self.\_size, self.\_size)  
 ghosts = self.\_renderer.get\_ghosts()  
 game\_objects = self.\_renderer.get\_game\_objects()  
 for ghost in ghosts:  
 collides = collision\_rect.colliderect(ghost.get\_shape())  
 if collides and ghost in game\_objects:  
 if self.\_renderer.is\_kokoro\_active():  
 game\_objects.remove(ghost)  
 self.\_renderer.add\_score(ScoreType.GHOST)  
 else:  
 if not self.\_renderer.get\_won():  
 self.\_renderer.kill\_pacman()  
  
 def draw(self):  
 half\_size = self.\_size / 2  
 self.image = self.open if self.mouth\_open else self.closed  
 self.image = pygame.transform.rotate(self.image, self.current\_direction.value)  
 super(Hero, self).draw()  
  
  
class Ghost(MovableObject):  
 def \_\_init\_\_(self, in\_surface, x, y, in\_size: int, in\_game\_controller, sprite\_path="images/ghost\_fright.png"):  
 super().\_\_init\_\_(in\_surface, x, y, in\_size)  
 self.game\_controller = in\_game\_controller  
 self.sprite\_normal = pygame.image.load(sprite\_path)  
 self.sprite\_fright = pygame.image.load("images/ghost\_fright.png")  
  
 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:  
 if self.\_renderer.get\_current\_mode() == GhostBehaviour.CHASE and not self.\_renderer.is\_kokoro\_active():  
 self.request\_path\_to\_player(self)  
 else:  
 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  
  
 if self.\_renderer.get\_current\_mode() == GhostBehaviour.CHASE and not self.\_renderer.is\_kokoro\_active():  
 self.request\_path\_to\_player(self)  
 else:  
 self.game\_controller.request\_new\_random\_path(self)  
 return Direction.NONE  
  
 def request\_path\_to\_player(self, in\_ghost):  
 player\_position = translate\_screen\_to\_maze(in\_ghost.\_renderer.get\_hero\_position())  
 current\_maze\_coord = translate\_screen\_to\_maze(in\_ghost.get\_position())  
 path = self.game\_controller.p.get\_path(current\_maze\_coord[1], current\_maze\_coord[0], player\_position[1],  
 player\_position[0])  
  
 new\_path = [translate\_maze\_to\_screen(item) for item in path]  
 in\_ghost.set\_new\_path(new\_path)  
  
 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)  
  
 def draw(self):  
 self.image = self.sprite\_fright if self.\_renderer.is\_kokoro\_active() else self.sprite\_normal  
 super(Ghost, self).draw()  
  
  
class Cookie(GameObject):  
 def \_\_init\_\_(self, in\_surface, x, y):  
 super().\_\_init\_\_(in\_surface, x, y, 4, (255, 255, 0), True)  
  
  
class Powerup(GameObject):  
 def \_\_init\_\_(self, in\_surface, x, y):  
 super().\_\_init\_\_(in\_surface, x, y, 8, (255, 255, 255), True)  
  
  
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]  
  
  
class PacmanGameController:  
 def \_\_init\_\_(self):  
 self.ascii\_maze = [  
 "XXXXXXXXXXXXXXXXXXXXXXXXXXXX",  
 "XP XX X",  
 "X XXXX XXXXX XX XXXXX XXXX X",  
 "X XXXXOXXXXX XX XXXXXOXXXX 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 G XX XXXXXX",  
 "XXXXXX XX XXX XXX XX XXXXXX",  
 "XXXXXX XX X X XX XXXXXX",  
 " G X X ",  
 "XXXXXX XX X X XX XXXXXX",  
 "XXXXXX XX XXXXXXXX XX XXXXXX",  
 "XXXXXX XX G 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 O O X",  
 "XXXXXXXXXXXXXXXXXXXXXXXXXXXX",  
 ]  
  
 self.numpy\_maze = []  
 self.cookie\_spaces = []  
 self.powerup\_spaces = []  
 self.reachable\_spaces = []  
 self.ghost\_spawns = []  
 self.ghost\_colors = [  
 "images/ghost.png",  
 "images/ghost\_pink.png",  
 "images/ghost\_orange.png",  
 "images/ghost\_blue.png"  
 ]  
 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))  
 if column == "O":  
 self.powerup\_spaces.append((y, x))  
  
 self.numpy\_maze.append(binary\_row)  
  
  
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 powerup\_space in pacman\_game.powerup\_spaces:  
 translated = translate\_maze\_to\_screen(powerup\_space)  
 powerup = Powerup(game\_renderer, translated[0] + unified\_size / 2, translated[1] + unified\_size / 2)  
 game\_renderer.add\_powerup(powerup)  
  
 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\_ghost(ghost)  
  
 pacman = Hero(game\_renderer, unified\_size, unified\_size, unified\_size)  
 game\_renderer.add\_hero(pacman)  
 game\_renderer.set\_current\_mode(GhostBehaviour.CHASE)  
 game\_renderer.tick(120)

**Скрин-шот экранов (если есть)**

****