

МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«МИРЭА – Российский технологический университет»**

**РТУ МИРЭА**

Институт искусственного интеллекта

Кафедра проблем управления

**ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ**

**По курсу «Управление интеллектуальными роботами и робототехническими системами»**

|  |  |  |
| --- | --- | --- |
| Студент группы  КРБО-03-19 | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Чепижный Е.А.  Гусельников И.А. |
|  |  |
|  |  |
|  |  |  |
| Преподаватель: | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Голубов В.В.  «\_\_» \_\_\_\_\_\_ 2023 г. |
|  |  |  |

Москва 2023

**Ход работы:**

Сначала напишем код для создания карты 400х400 пикселей с рандомной генерацией разных препятствий.

Код генерирует карту с заданным размером, указанным количеством препятствий и случайно сгенерированными координатами и размерами для каждого препятствия. Затем он рисует препятствия на поверхности дисплея с помощью функции Pygame draw.rect(). Пример сгенерированной карты представлен на рис.1. Код представлен в приложении 1

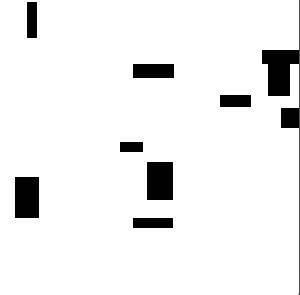


Рис.1 - Сгенерированная карта с препятствиями

Далее необходимо дискретизировать карту - разбив ее на ячейки.  
 Переменная grid\_size используется для установки размера ячеек сетки. Функция discretize\_point принимает точку в качестве входных данных и возвращает ближайшую точку на сетке.

В основном цикле сетка рисуется на поверхности дисплея, и выбранные точки дискретизируются перед проверкой, находятся ли они внутри препятствия. Дискретизированная карта представлена на Рис.2. Код для дискретизации карты представлен в приложении 2

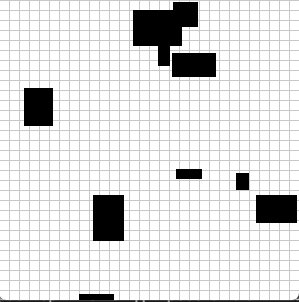
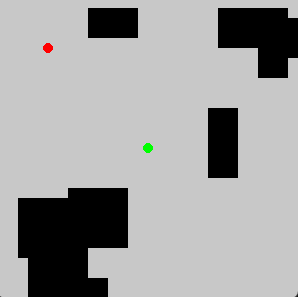


Рис.2 - Дискретизация карты

Чтобы создать пространство конфигурации для препятствий и робота, нам нужно принять во внимание форму и размер робота. Один из способов сделать это - создать новый набор препятствий, которые представляют пространство, занимаемое роботом. Это можно сделать, создав круг вокруг робота радиусом, равным половине размера робота, а затем разделив круг на набор квадратов, используя тот же размер сетки, что и препятствия. На Рис.3 представлено конфигурационное пространство, а в приложении 3 код для его реализации.

  
Рис.3 - Конфигурационное пространство

Далее необходимо реализовать алгоритм Дейкстры для нахождения кратчайшего пути между двумя точками на карте, избегая препятствий.

Код инициализирует несколько переменных, таких как ширина и высота карты, количество препятствий, размер робота и размер сетки, используемой для дискретизации пространства. Затем генерирует случайные прямоугольные препятствия и сохраняет их в списке.

Программа определяет несколько вспомогательных функций для проверки того, находится ли точка внутри препятствия, находится ли точка внутри конфигурационного пространства робота и для дискретизации точки до ближайшей точки сетки.

Функция draw\_map() используется для рисования карты и препятствий на поверхности дисплея Pygame. Сначала он очищает поверхность до белого цвета, затем рисует препятствия в виде черных прямоугольников и, наконец, рисует сетку в виде светло-серых линий. Он также рисует пространство конфигурации робота в виде серого прямоугольника для каждой ячейки сетки, которая не находится внутри препятствия или внутри пространства конфигурации робота.

Основной цикл программы предлагает пользователю ввести начальную и конечную точки робота и проверяет, находятся ли точки в пределах границ карты. Затем он рисует начальную и конечную точки на поверхности дисплея в виде красных и зеленых кругов соответственно и вычисляет кратчайший путь, используя алгоритм Дейкстры, избегая препятствий и пространства конфигурации робота. Затем кратчайший путь рисуется на поверхности дисплея в виде синей линии.

В целом, эта программа предоставляет простой пример того, как визуализировать 2D-пространство и использовать алгоритм Дейкстры для нахождения кратчайшего пути между двумя точками, избегая препятствий. На рис. 4 представлен построенный при помощи алгоритма дейкстры путь в конфигурационном пространстве, а в приложении 4 программа для реализации.

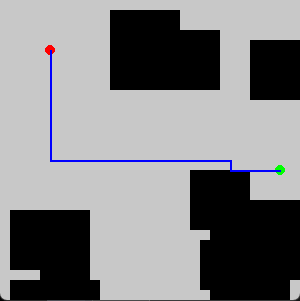


Рис. 4 - Алгоритм Дейкстры в конфигурационном пространстве

Далее используем алгоритм А\*, для нахождения пути в конфигурационном пространстве.

В начале код инициализирует pygame и определяет несколько переменных для карты, препятствий, размера и радиуса робота, а также алгоритма A\*. Затем он генерирует препятствия и определяет функции для проверки того, находится ли точка в препятствии, находится ли точка в конфигурационном пространстве робота и для дискретизации точки до ближайшей точки сетки. Он также определяет функцию для рисования карты.

Затем переходит в цикл, который предлагает пользователю ввести начальную и конечную точки. Если пользователь вводит "exit" для любой точки, цикл завершается. Если точки действительны (т.е. находятся в пределах границ карты), скрипт рисует круги в начальной и конечной точках на карте и использует алгоритм A\* для поиска пути от начальной до конечной точки. Затем путь рисуется на карте.

На рис. 5 представлен построенный при помощи алгоритма а\* путь в конфигурационном пространстве, а в приложении 5 программа для реализации.

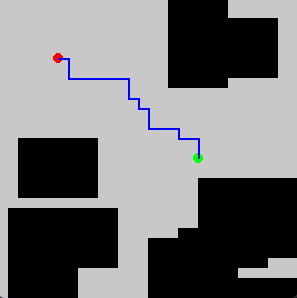


Рис. 5 - Алгоритм А\* в конфигурационном пространстве

**Приложение 1:**

import pygame

import random

pygame.init()

map\_width = 300

map\_height = 300

num\_obstacles = 10

obstacles = []

map\_color = (255, 255, 255) # white

obstacle\_color = (0, 0, 0) # black

display\_surface = pygame.display.set\_mode((map\_width, map\_height))

for i in range(num\_obstacles):

# Generate random coordinates for the obstacle

obstacle\_x = random.randint(0, map\_width)

obstacle\_y = random.randint(0, map\_height)

obstacle\_width = random.randint(10, 50)

obstacle\_height = random.randint(10, 50)

obstacles.append(pygame.Rect(obstacle\_x, obstacle\_y, obstacle\_width, obstacle\_height))

running = True

while running:

for event in pygame.event.get():

if event.type == pygame.QUIT:

running = False

display\_surface.fill(map\_color)

for obstacle in obstacles:

pygame.draw.rect(display\_surface, obstacle\_color, obstacle)

pygame.display.update()

pygame.quit()

**Приложение 2:** import pygame

import random

pygame.init()

map\_width = 300

map\_height = 300

num\_obstacles = 10

grid\_size = 10

obstacles = []

map\_color = (255, 255, 255) # white

obstacle\_color = (0, 0, 0) # black

grid\_color = (200, 200, 200) # light gray

display\_surface = pygame.display.set\_mode((map\_width, map\_height))

for i in range(num\_obstacles):

obstacle\_x = random.randint(0, map\_width)

obstacle\_y = random.randint(0, map\_height)

obstacle\_width = random.randint(10, 50)

obstacle\_height = random.randint(10, 50)

obstacles.append(pygame.Rect(obstacle\_x, obstacle\_y, obstacle\_width, obstacle\_height))

def is\_point\_in\_obstacle(point):

for obstacle in obstacles:

if obstacle.collidepoint(point):

return True

return False

def discretize\_point(point):

x = int(round(point[0] / grid\_size)) \* grid\_size

y = int(round(point[1] / grid\_size)) \* grid\_size

return (x, y)

running = True

while running:

for event in pygame.event.get():

if event.type == pygame.QUIT:

running = False

elif event.type == pygame.MOUSEBUTTONDOWN:

sample\_point = pygame.mouse.get\_pos()

discretized\_point = discretize\_point(sample\_point)

if is\_point\_in\_obstacle(discretized\_point):

print("Sampled point is inside an obstacle!")

else:

print("Sampled point is not inside an obstacle.")

print("Discretized point:", discretized\_point)

display\_surface.fill(map\_color)

for x in range(0, map\_width, grid\_size):

pygame.draw.line(display\_surface, grid\_color, (x, 0), (x, map\_height))

for y in range(0, map\_height, grid\_size):

pygame.draw.line(display\_surface, grid\_color, (0, y), (map\_width, y))

for obstacle in obstacles:

pygame.draw.rect(display\_surface, obstacle\_color, obstacle)

pygame.display.update()

pygame.quit()

**Приложение 3:**import pygame

import random

import math

pygame.init()

map\_width = 300

map\_height = 300

num\_obstacles = 10

grid\_size = 10

obstacles = []

map\_color = (255, 255, 255) # white

obstacle\_color = (0, 0, 0) # black

grid\_color = (125, 125, 125) # light gray

display\_surface = pygame.display.set\_mode((map\_width, map\_height))

robot\_size = 10

robot\_radius = robot\_size / 2

robot\_space = []

for x in range(-robot\_size, robot\_size + grid\_size, grid\_size):

for y in range(-robot\_size, robot\_size + grid\_size, grid\_size):

if math.sqrt(x \*\* 2 + y \*\* 2) <= robot\_radius:

robot\_space.append((x, y))

for i in range(num\_obstacles):

obstacle\_x = random.randint(0, map\_width)

obstacle\_y = random.randint(0, map\_height)

obstacle\_width = random.randint(10, 50)

obstacle\_height = random.randint(10, 50)

obstacles.append(pygame.Rect(obstacle\_x, obstacle\_y, obstacle\_width, obstacle\_height))

def is\_point\_in\_obstacle(point):

for obstacle in obstacles:

if obstacle.collidepoint(point):

return True

return False

def is\_point\_in\_robot\_space(point):

for x, y in robot\_space:

if is\_point\_in\_obstacle((point[0] + x, point[1] + y)):

return True

return False

def discretize\_point(point):

x = int(round(point[0] / grid\_size)) \* grid\_size

y = int(round(point[1] / grid\_size)) \* grid\_size

return (x, y)

def draw\_map():

display\_surface.fill(map\_color)

for obstacle in obstacles:

pygame.draw.rect(display\_surface, obstacle\_color, obstacle)

for x in range(0, map\_width, grid\_size):

pygame.draw.line(display\_surface, grid\_color, (x, 0), (x, map\_height))

for y in range(0, map\_height, grid\_size):

pygame.draw.line(display\_surface, grid\_color, (0, y), (map\_width, y))

config\_space\_color = (200, 200, 200)

for x in range(0, map\_width, grid\_size):

for y in range(0, map\_height, grid\_size):

point = (x, y)

if is\_point\_in\_obstacle(point):

continue

in\_config\_space = True

for dx, dy in robot\_space:

if is\_point\_in\_obstacle((x + dx, y + dy)):

in\_config\_space = False

break

if in\_config\_space:

pygame.draw.rect(display\_surface, config\_space\_color, (x, y, grid\_size, grid\_size))

pygame.display.update()

while True:

start = input("Enter the starting point of the robot (in x,y format): ")

end = input("Enter the ending point of the robot (in x,y format): ")

try:

start = tuple(map(int, start.split(",")))

end = tuple(map(int, end.split(",")))

if start[0] < 0 or start[0] >= map\_width or start[1] < 0 or start[1] >= map\_height:

raise ValueError("Starting point is outside the map")

if end[0] < 0 or end[0] >= map\_width or end[1] < 0 or end[1] >= map\_height:

raise ValueError("Ending point is outside the map")

pygame.draw.circle(display\_surface, (255, 0, 0), start, 5)

pygame.draw.circle(display\_surface, (0, 255, 0), end, 5)

config\_space\_color = (200, 200, 200)

for x in range(0, map\_width, grid\_size):

for y in range(0, map\_height, grid\_size):

point = (x, y)

if is\_point\_in\_obstacle(point):

continue

robot\_radius = 10

in\_config\_space = True

for dx in range(-robot\_radius, robot\_radius + 1):

for dy in range(-robot\_radius, robot\_radius + 1):

if is\_point\_in\_obstacle((x + dx, y + dy)):

in\_config\_space = False

break

if not in\_config\_space:

break

if in\_config\_space:

pygame.draw.rect(display\_surface, config\_space\_color, (x, y, grid\_size, grid\_size))

start\_disc = discretize\_point(start)

end\_disc = discretize\_point(end)

pygame.draw.circle(display\_surface, (255, 0, 0), start\_disc, 5)

pygame.draw.circle(display\_surface, (0, 255, 0), end\_disc, 5)

pygame.display.update()

except ValueError as e:

print(f"Error: {e}")

continue

for event in pygame.event.get():

if event.type == pygame.QUIT:

pygame.quit()

quit()

else:

continue

break

**Приложение 4:**

import pygame

import random

import math

import heapq

pygame.init()

map\_width = 300

map\_height = 300

num\_obstacles = 10

grid\_size = 10

obstacles = []

map\_color = (255, 255, 255)

obstacle\_color = (0, 0, 0)

grid\_color = (125, 125, 125)

display\_surface = pygame.display.set\_mode((map\_width, map\_height))

robot\_size = 20

robot\_radius = robot\_size

robot\_space = []

for x in range(-robot\_size, robot\_size + grid\_size, grid\_size):

for y in range(-robot\_size, robot\_size + grid\_size, grid\_size):

if math.sqrt(x \*\* 2 + y \*\* 2) <= robot\_radius:

robot\_space.append((x, y))

for i in range(num\_obstacles):

obstacle\_x = random.randint(0, map\_width)

obstacle\_y = random.randint(0, map\_height)

obstacle\_width = random.randint(10, 50)

obstacle\_height = random.randint(10, 50)

obstacles.append(pygame.Rect(obstacle\_x, obstacle\_y, obstacle\_width, obstacle\_height))

def is\_point\_in\_obstacle(point):

for obstacle in obstacles:

if obstacle.collidepoint(point):

return True

return False

def dijkstra(start, end):

dist = {start: 0}

prev = {}

unvisited = [(0, start)]

while unvisited:

current\_dist, current = heapq.heappop(unvisited)

if current == end:

path = []

while current in prev:

path.append(current)

current = prev[current]

path.append(start)

path.reverse()

return path

for dx, dy in [(0, grid\_size), (0, -grid\_size), (grid\_size, 0), (-grid\_size, 0)]:

neighbor = (current[0] + dx, current[1] + dy)

if neighbor[0] < 0 or neighbor[0] >= map\_width or neighbor[1] < 0 or neighbor[1] >= map\_height:

continue

if is\_point\_in\_robot\_space(neighbor):

continue

distance = math.sqrt(dx \*\* 2 + dy \*\* 2)

neighbor\_dist = dist[current] + distance

if neighbor not in dist or neighbor\_dist < dist[neighbor]:

dist[neighbor] = neighbor\_dist

prev[neighbor] = current

heapq.heappush(unvisited, (neighbor\_dist, neighbor))

return None

def is\_point\_in\_robot\_space(point):

for x, y in robot\_space:

if is\_point\_in\_obstacle((point[0] + x, point[1] + y)):

return True

return False

def discretize\_point(point):

x = int(round(point[0] / grid\_size)) \* grid\_size

y = int(round(point[1] / grid\_size)) \* grid\_size

return (x, y)

def draw\_map():

display\_surface.fill(map\_color)

for obstacle in obstacles:

pygame.draw.rect(display\_surface, obstacle\_color, obstacle)

for x in range(0, map\_width, grid\_size):

pygame.draw.line(display\_surface, grid\_color, (x, 0), (x, map\_height))

for y in range(0, map\_height, grid\_size):

pygame.draw.line(display\_surface, grid\_color, (0, y), (map\_width, y))

config\_space\_color = (200, 200, 200)

for x in range(0, map\_width, grid\_size):

for y in range(0, map\_height, grid\_size):

point = (x, y)

if is\_point\_in\_obstacle(point):

continue

in\_config\_space = True

for dx, dy in robot\_space:

if is\_point\_in\_obstacle((x + dx, y + dy)):

in\_config\_space = False

break

if in\_config\_space:

pygame.draw.rect(display\_surface, config\_space\_color, (x, y, grid\_size, grid\_size))

pygame.display.update()

while True:

start = input("Enter the starting point of the robot (in x,y format): ")

end = input("Enter the ending point of the robot (in x,y format): ")

if start.lower() == 'exit' or end.lower() == 'exit':

break

try:

start = tuple(map(int, start.split(",")))

end = tuple(map(int, end.split(",")))

if start[0] < 0 or start[0] >= map\_width or start[1] < 0 or start[1] >= map\_height:

raise ValueError("Starting point is outside the map")

if end[0] < 0 or end[0] >= map\_width or end[1] < 0 or end[1] >= map\_height:

raise ValueError("Ending point is outside the map")

pygame.draw.circle(display\_surface, (255, 0, 0), start, 5)

pygame.draw.circle(display\_surface, (0, 255, 0), end, 5)

config\_space\_color = (200, 200, 200)

for x in range(0, map\_width, grid\_size):

for y in range(0, map\_height, grid\_size):

point = (x, y)

if is\_point\_in\_obstacle(point):

continue

robot\_radius = 20

in\_config\_space = True

for dx in range(-robot\_radius, robot\_radius + 1):

for dy in range(-robot\_radius, robot\_radius + 1):

if is\_point\_in\_obstacle((x + dx, y + dy)):

in\_config\_space = False

break

if not in\_config\_space:

break

if in\_config\_space:

pygame.draw.rect(display\_surface, config\_space\_color, (x, y, grid\_size, grid\_size))

start\_disc = discretize\_point(start)

end\_disc = discretize\_point(end)

pygame.draw.circle(display\_surface, (255, 0, 0), start\_disc, 5)

pygame.draw.circle(display\_surface, (0, 255, 0), end\_disc, 5)

path = dijkstra(start\_disc, end\_disc)

if path is not None:

for i in range(len(path) - 1):

pygame.draw.line(display\_surface, (0, 0, 255), path[i], path[i + 1], 2)

pygame.display.update()

else:

print("No path found")

except Exception as e:

print("Error:", e)

continue

for event in pygame.event.get():

if event.type == pygame.QUIT:

pygame.quit()

quit()

else:

continue

break

**Приложение 5:**

import pygame

import random

import math

pygame.init()

map\_width = 300

map\_height = 300

num\_obstacles = 10

grid\_size = 10

obstacles = []

map\_color = (255, 255, 255)

obstacle\_color = (0, 0, 0)

grid\_color = (125, 125, 125)

display\_surface = pygame.display.set\_mode((map\_width, map\_height))

robot\_size = 20

robot\_radius = robot\_size

robot\_space = []

for x in range(-robot\_size, robot\_size + grid\_size, grid\_size):

for y in range(-robot\_size, robot\_size + grid\_size, grid\_size):

if math.sqrt(x \*\* 2 + y \*\* 2) <= robot\_radius:

robot\_space.append((x, y))

for i in range(num\_obstacles):

obstacle\_x = random.randint(0, map\_width)

obstacle\_y = random.randint(0, map\_height)

obstacle\_width = random.randint(10, 50)

obstacle\_height = random.randint(10, 50)

obstacles.append(pygame.Rect(obstacle\_x, obstacle\_y, obstacle\_width, obstacle\_height))

def is\_point\_in\_obstacle(point):

for obstacle in obstacles:

if obstacle.collidepoint(point):

return True

return False

def heuristic(a, b):

return abs(a[0] - b[0]) + abs(a[1] - b[1])

def a\_star(start, end):

open\_set = {start}

closed\_set = set()

came\_from = {}

g\_score = {start: 0}

f\_score = {start: heuristic(start, end)}

while open\_set:

current = min(open\_set, key=f\_score.get)

if current == end:

path = []

while current in came\_from:

path.append(current)

current = came\_from[current]

path.append(start)

path.reverse()

return path

open\_set.remove(current)

closed\_set.add(current)

for dx, dy in [(0, grid\_size), (0, -grid\_size), (grid\_size, 0), (-grid\_size, 0)]:

neighbor = (current[0] + dx, current[1] + dy)

if neighbor[0] < 0 or neighbor[0] >= map\_width or neighbor[1] < 0 or neighbor[1] >= map\_height:

continue

if is\_point\_in\_robot\_space(neighbor):

continue

if neighbor in closed\_set:

continue

tentative\_g\_score = g\_score[current] + math.sqrt(dx \*\* 2 + dy \*\* 2)

if neighbor not in open\_set or tentative\_g\_score < g\_score[neighbor]:

came\_from[neighbor] = current

g\_score[neighbor] = tentative\_g\_score

f\_score[neighbor] = tentative\_g\_score + heuristic(neighbor, end)

if neighbor not in open\_set:

open\_set.add(neighbor)

return None

def is\_point\_in\_robot\_space(point):

for x, y in robot\_space:

if is\_point\_in\_obstacle((point[0] + x, point[1] + y)):

return True

return False

def discretize\_point(point):

x = int(round(point[0] / grid\_size)) \* grid\_size

y = int(round(point[1] / grid\_size)) \* grid\_size

return (x, y)

def draw\_map():

display\_surface.fill(map\_color)

for obstacle in obstacles:

pygame.draw.rect(display\_surface, obstacle\_color, obstacle)

for x in range(0, map\_width, grid\_size):

pygame.draw.line(display\_surface, grid\_color, (x, 0), (x, map\_height))

for y in range(0, map\_height, grid\_size):

pygame.draw.line(display\_surface, grid\_color, (0, y), (map\_width, y))

config\_space\_color = (200, 200, 200)

for x in range(0, map\_width, grid\_size):

for y in range(0, map\_height, grid\_size):

point = (x, y)

if is\_point\_in\_obstacle(point):

continue

in\_config\_space = True

for dx, dy in robot\_space:

if is\_point\_in\_obstacle((x + dx, y + dy)):

in\_config\_space = False

break

if in\_config\_space:

pygame.draw.rect(display\_surface, config\_space\_color, (x, y, grid\_size, grid\_size))

pygame.display.update()

while True:

start = input("Enter the starting point of the robot (in x,y format): ")

end = input("Enter the ending point of the robot (in x,y format): ")

if start.lower() == 'exit' or end.lower() == 'exit':

break

try:

start = tuple(map(int, start.split(",")))

end = tuple(map(int, end.split(",")))

if start[0] < 0 or start[0] >= map\_width or start[1] < 0 or start[1] >= map\_height:

raise ValueError("Starting point is outside the map")

if end[0] < 0 or end[0] >= map\_width or end[1] < 0 or end[1] >= map\_height:

raise ValueError("Ending point is outside the map")

pygame.draw.circle(display\_surface, (255, 0, 0), start, 5)

pygame.draw.circle(display\_surface, (0, 255, 0), end, 5)

config\_space\_color = (200, 200, 200)

for x in range(0, map\_width, grid\_size):

for y in range(0, map\_height, grid\_size):

point = (x, y)

if is\_point\_in\_obstacle(point):

continue

robot\_radius = 20

in\_config\_space = True

for dx in range(-robot\_radius, robot\_radius + 1):

for dy in range(-robot\_radius, robot\_radius + 1):

if is\_point\_in\_obstacle((x + dx, y + dy)):

in\_config\_space = False

break

if not in\_config\_space:

break

if in\_config\_space:

pygame.draw.rect(display\_surface, config\_space\_color, (x, y, grid\_size, grid\_size))

start\_disc = discretize\_point(start)

end\_disc = discretize\_point(end)

pygame.draw.circle(display\_surface, (255, 0, 0), start\_disc, 5)

pygame.draw.circle(display\_surface, (0, 255, 0), end\_disc, 5)

path = a\_star(start\_disc, end\_disc)

if path is not None:

for i in range(len(path) - 1):

pygame.draw.line(display\_surface, (0, 0, 255), path[i], path[i + 1], 2)

pygame.display.update()

else:

print("No path found")

except Exception as e:

print("Error:", e)

continue

for event in pygame.event.get():

if event.type == pygame.QUIT:

pygame.quit()

quit()

else:

continue

break