

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение

высшего профессионального образования

**«Дальневосточный федеральный университет»**

(ДВФУ)

**ШКОЛА ЕСТЕСТВЕННЫХ НАУК**

**Кафедра информатики, математического  
 и компьютерного моделирования**

**ОТЧЕТ**

о выполнении лабораторной работы «Компьютерная модель движения толпы»

по предмету «Математическое и компьютерное моделирование»

Направление подготовки

01.03.02 «Прикладная математика и информатика»

Выполнил  
студент группы Б8117-01.03.02

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Бондаревский А.С.

Проверил д.ф.-м.н.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Пермяков М. С.

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2020 г.

**г. Владивосток**

**2020**

Оглавление

[Цель 3](#_Toc31150929)

[Формализация задачи 3](#_Toc31150930)

[Описание 3](#_Toc31150931)

[Метод решения 3](#_Toc31150932)

[Анализ модели 7](#_Toc31150933)

[Вывод 8](#_Toc31150934)

# Цель

Цель данной лабораторной работы – определить положение столбов в помещении на основе статистики, при котором толпа быстрее всего выберется из помещения.

# Формализация задачи

Задача данной лабораторной работы – написать программу на языке программирования Python, которая реализует движение толпы в узком пространстве с помощью клеточного автомата, и получить статистику на основе данной программы.

# Описание

Клеточный автомат – дискретная модель, включающая решётку ячеек, каждая из которых может находиться в одном из конечного множества состояний, таких как:

* 0 – Пустое место;
* 1 – Человек;
* 2 – Стена;

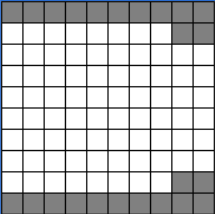
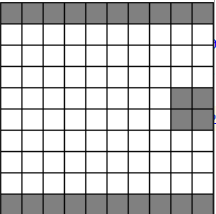
Решетка может быть любой размерности. Для работы клеточного автомата требуется задание начального состояния всех ячеек и правила перехода ячеек из одного состояния в другое.

Толпа – множество ячеек, которые находятся в состоянии 1, расположенных случайным образом в клеточном автомате.

# Метод решения

Для выполнения поставленной задачи воспользуемся средствами языка программирования Python, библиотеки numpy, pygame, pylab, random.

Для симуляции помещения на клеточном автомате определим верхнюю и нижнюю строчку ячеек, как стены. Также для симуляции столбов в помещении определим по 2 ячейки на один столб, где данные столбы будут двигаться к друг другу таким образом:

Для получения статистики прогоним 10 случайных толп на каждое положение столбов.

Правила перехода ячеек из одного состояния в другое:

1. Если ближайшая правая ячейка от ячейки с состоянием 1 находится ячейка с состоянием 0, то ячейка с состоянием 0 переходит в состояние 1, а ячейка с состоянием 1 переходит в состояние 0;
2. Если ближайшая правая ячейка от ячейки с состоянием 1 находится в состоянии 1 или 2, то:
   1. Если ближайшая верхняя ячейка от ячейки с состоянием 1 находится в состоянии 0, то ячейка с состоянием 0 переходит в состояние 1, а ячейка с состоянием 1 переходит в состояние 0;
   2. Если ближайшая нижняя ячейка от ячейки с состоянием 1 находится в состоянии 0, то ячейка с состоянием 0 переходит в состояние 1, а ячейка с состоянием 1 переходит в состояние 0;

**Код программы:**

**import** random  
**import** pylab  
**import** pygame  
**import** numpy **as** np  
  
BLACK = (0, 0, 0)  
WHITE = (255, 255, 255)  
GREY = (128, 128, 128)  
GREEN = (0, 255, 0)  
WIDTH = 16  
HEIGHT = 16  
MARGIN = 1  
count\_rect\_x = 50  
count\_rect\_y = 30  
  
  
**def** paint\_matrix(matrix, screen, time):  
 screen.fill(BLACK)  
 **for** i **in** range(count\_rect\_y):  
 **for** j **in** range(count\_rect\_x):  
 color = WHITE  
 **if** matrix[i][j] == 1:  
 color = GREEN  
 **if** matrix[i][j] == 2:  
 color = GREY  
 pygame.draw.rect(screen, color,  
 [(MARGIN + WIDTH) \* j + MARGIN, (MARGIN + HEIGHT) \* i + MARGIN, WIDTH, HEIGHT])  
 *# time.tick(10)* pygame.display.flip()  
  
  
**def** initialize\_model\_matrix():  
 matrix = np.zeros((count\_rect\_y, count\_rect\_x))  
 matrix[0, 0:count\_rect\_x] = 2  
 matrix[-1, 0:count\_rect\_x] = 2  
 **return** matrix  
  
  
**def** set\_people(matrix, count):  
 people\_count = count  
 random\_dict = {}  
 **for** i **in** range(1, count\_rect\_y - 1):  
 random\_dict[i] = []  
 **while** people\_count > 0:  
 random\_row = random.randint(1, count\_rect\_y - 2)  
 random\_col = random.randint(0, count\_rect\_x - 3)  
 **if** random\_col **not in** random\_dict[random\_row]:  
 matrix[random\_row, random\_col] = 1  
 random\_dict[random\_row].append(random\_col)  
 people\_count -= 1  
  
  
**def** set\_walls(matrix, offset):  
 matrix[offset, count\_rect\_x-2:count\_rect\_x] = 2  
 matrix[count\_rect\_y-1 - offset, count\_rect\_x-2:count\_rect\_x] = 2  
  
  
**def** check\_people(matrix):  
 **for** i **in** range(count\_rect\_y):  
 **for** j **in** range(count\_rect\_x):  
 **if** matrix[i][j] == 1:  
 **return True  
 return False  
  
  
def** move\_people(matrix, screen, time):  
 iterations\_result = 0  
 **while** check\_people(matrix):  
 iterations\_result += 1  
 **for** is\_move **in** [**False**, **True**]:  
 **for** i **in** range(count\_rect\_y):  
 **for** j **in** range(count\_rect\_x-1, -1, -1):  
 **if** is\_move:  
 **if** matrix[i][j] == 3:  
 matrix[i][j] = 0  
 **elif** matrix[i][j] **in** [4, 5, 7]:  
 matrix[i][j] = 1  
 **elif** matrix[i][j] == 1:  
 **if** j + 1 == count\_rect\_x:  
 matrix[i][j] = 0  
 **continue  
 else**:  
 **if** (j + 1 < count\_rect\_x) **and** matrix[i][j] == 1:  
 **if** matrix[i][j + 1] **in** [0, 3]:  
 matrix[i][j] = 3  
 matrix[i][j + 1] = 7  
 **elif** (matrix[i + 1][j] **in** [0, 3]) **and not** (matrix[i - 1][j] **in** [0, 3]):  
 matrix[i][j] = 3  
 matrix[i + 1][j] = 5  
 **elif not** (matrix[i + 1][j] **in** [0, 3]) **and** (matrix[i - 1][j] **in** [0, 3]):  
 matrix[i][j] = 3  
 matrix[i - 1][j] = 4  
 **elif** (matrix[i + 1][j] **in** [0, 3]) **and** (matrix[i - 1][j] **in** [0, 3]):  
 matrix[i][j] = 3  
 random\_index = random.choice([-1, 1])  
 **if** random\_index == -1:  
 matrix[i + random\_index][j] = 4  
 **else**:  
 matrix[i + random\_index][j] = 5  
 paint\_matrix(matrix, screen, time)  
 **return** iterations\_result  
  
  
offset = []  
iterations = []  
WINDOW\_SIZE = [WIDTH \* count\_rect\_x + 1 + MARGIN \* count\_rect\_x,  
 HEIGHT \* count\_rect\_y + 1 + MARGIN \* count\_rect\_y]  
screen = pygame.display.set\_mode(WINDOW\_SIZE)  
**for** j **in** range(10):  
 model\_matrix = initialize\_model\_matrix()  
 set\_people(model\_matrix, count\_rect\_x \* count\_rect\_y // 2 - count\_rect\_x \* 2 - 4)  
 **for** i **in** range(1, count\_rect\_y//2, 1):  
 time = pygame.time.Clock()  
 current\_matrix = np.copy(model\_matrix)  
 set\_walls(current\_matrix, i)  
 count = move\_people(current\_matrix, screen, time)  
 **if** j == 0:  
 iterations.append(count)  
 offset.append(i)  
 **else**:  
 iterations[i-1] = iterations[i-1] + count  
**for** i **in** range(1, count\_rect\_y//2, 1):  
 iterations[i-1] = iterations[i-1]/10  
pylab.plot(offset, iterations)  
pylab.show()

# Анализ модели

count\_rect\_x - Количество ячеек в строке

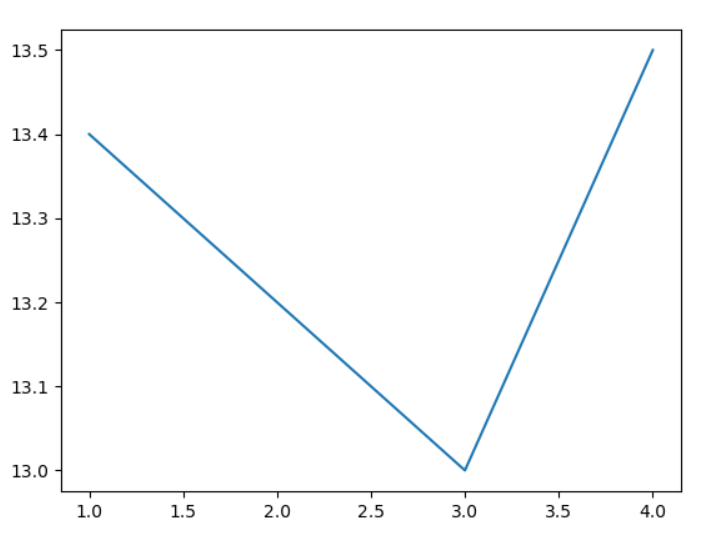
count\_rect\_y - Количество ячеек в столбце

Ось ординат – среднее кол-во итераций.

Ось абсцисс – кол-во ячеек сдвига столбов.

count\_rect\_x = 10

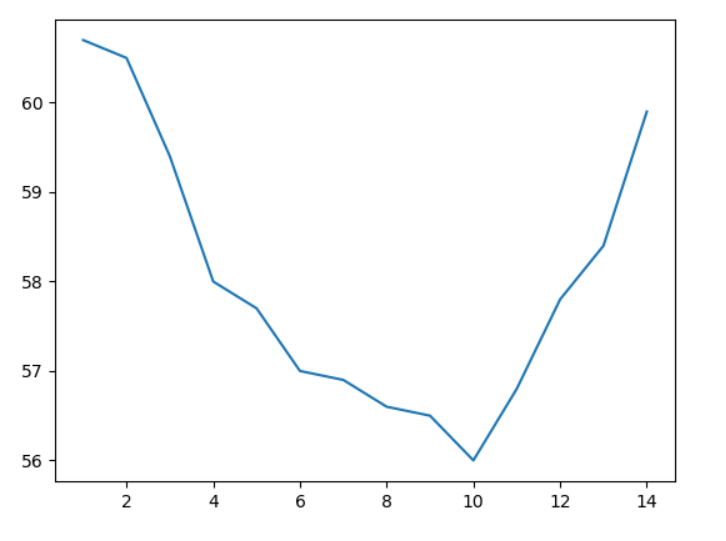
count\_rect\_y = 10



Для данного размера сетки есть всего 4 положения столбов, и лучшим, с точки зрения среднего кол-ва итерация, является 3 ячейки сдвига. Расширим сетку, для более точного графика.

count\_rect\_x = 50

count\_rect\_y = 30



Для данного размера сетки есть всего 14 положения столбов, и лучшим, с точки зрения среднего кол-ва итерация, является 10 ячеек сдвига.

Исходя из этой статистики также следует, что худшими вариантами являются краевые.

# Вывод

Таким образом, мы получили статистику на основе программы и определили лучшие положения столбов для разного размера сетки.