

Задания финала Иннополис Опен 2025-2026. 7-8 класс

Задание 1

Стартовая позиция фигуры $(-8;3)$, робот должен пронести её по траектории, указанной далее, не поднимая более чем на 25 мм от поверхности стола. Траектория: $(-8;3)$ - $(-5;3)$ - $(-5;5)$ - $(5;5)$ - $(5;3)$ - $(8;3)$. По сторонам на клетках, закрашенных красным, стоит забор, который нельзя сбивать. Ставить фигуру по пути, ронять, поднимать выше высоты ограждений – запрещается.

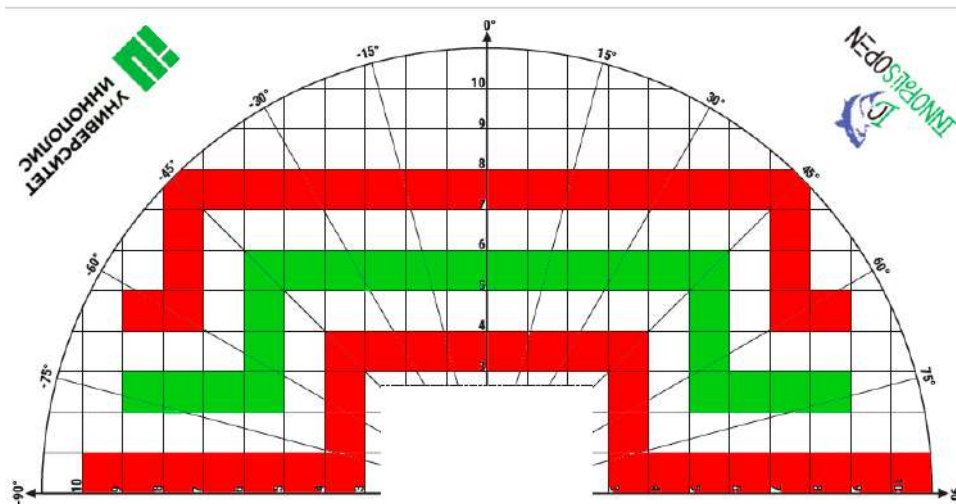


Рис. 1: Пример расстановки на игровом поле

Баллы за задание:

Фигура взята со своего места и удерживается манипулятором	5 баллов
успешно (без нарушений) пройдена область отрицательных координат по оси X	10 баллов
успешно (без нарушений) пройдена область положительных координат по оси X	10 баллов
Фигура поставлена в клетку $(8;3)$, опирается на поле всем своим основанием и не выходит за границы клетки	15 баллов
Фигура поставлена в клетку $(8;3)$, опирается на поле всем своим основанием и частично выходит за границы клетки	5 баллов

Максимальный балл: 40

Разбор решения — Задание 1

Идея решения

Основная задача робота — аккуратно взять фигуру, пронести её по заданной траектории и поставить в конечную точку, не нарушая ограничений.

Решение строится по следующему плану:

1. Переместить манипулятор в безопасную стартовую позицию
2. Подъехать к фигуре и захватить её с помощью захвата (gripper)
3. Поднять фигуру на небольшую высоту (не более 25 мм от стола)
4. Последовательно пройти все точки траектории
5. В конце аккуратно опустить фигуру и отпустить её
6. Вернуть манипулятор в исходное положение

Все перемещения выполняются с помощью команды `go_xyz(x, y, z)`, которая задаёт координаты в пространстве.

Разбор сложных мест

1. Ограничение по высоте

В условии сказано, что нельзя поднимать фигуру выше 25 мм от поверхности стола. Это значит:

- нельзя делать большие подъёмы вверх
- все перемещения выполняются примерно на одной высоте (например, $z \approx 30$)

Если поднять выше — задание считается выполненным с нарушением.

2. Точное следование траектории

Робот должен пройти строго по точкам:

$$(-8; 3) \rightarrow (-5; 3) \rightarrow (-5; 5) \rightarrow (5; 5) \rightarrow (5; 3) \rightarrow (8; 3)$$

В программе следование по точкам реализовано последовательными вызовами:

- каждая точка — отдельная команда `go_xyz`
- между командами добавлены задержки `sleep`, чтобы движение успевало завершиться

Если пропустить точку — можно задеть «забор».

3. Захват фигуры

Захват выполняется в два этапа:

1. Подъезд сверху (безопасно)

2. Опускание вниз

3. Закрытие захвата

Это важно, чтобы:

- не столкнуться с фигурой сбоку
- не сдвинуть её

—

4. Скорость движения

В коде используется параметр скорости:

- быстрое движение — когда нет препятствий
- медленное движение — рядом с фигурой или забором

Это уменьшает риск ошибок.

—

Полный код решения

```
from roboarm import RoboARM
import time
```

```
arm, port = RoboARM.auto_connect(baudrate=115200, tool_type='gripper',ack_timeout=2.5,
print('Connected RoboARM on', port))
```

```
arm.go_xyz(150, 0, 150, v=100)
time.sleep(2)
arm.gripper_set(61, 30)
time.sleep(3)
```

```
arm.go_xyz(58, -216, 150, v=30)
arm.go_xyz(58, -216, 50, v=30)
time.sleep(2)
arm.go_xyz(58, -216, 20, v=30)
arm.gripper_set(61, 5)
time.sleep(2)
arm.go_xyz(58, -216, 30, v=30) # первая точка
```

```
time.sleep(2)
arm.go_xyz(58, -144, 30, v=30) # вторая точка
time.sleep(2)
arm.go_xyz(94, -144, 30, v=30) # вторая точка
```

```
time.sleep(2)
arm.go_xyz(146, -144, 30, v=30) # третья точка
time.sleep(2)
arm.gripper_set(38, 10)
```

```
time.sleep(2)
arm.go_xyz(146, -9, 30, v=30) # четвертая точка
time.sleep(2)
arm.gripper_set(-7, 10)

time.sleep(2)
arm.go_xyz(140, 76, 30, v=30) # пятая точка
time.sleep(2)
arm.gripper_set(-37, 10)

time.sleep(2)
arm.go_xyz(137, 139, 30, v=30) # шестая точка
time.sleep(2)
arm.gripper_set(35, 10)

time.sleep(2)
arm.go_xyz(55, 140, 30, v=30) # седьмая точка
time.sleep(2)
arm.gripper_set(-80, 10)

time.sleep(2)
arm.go_xyz(60, 184, 30, v=30) # финиш

arm.go_xyz(65, 184, 21, v=30)
time.sleep(2)
arm.gripper_set(-80, 30)
time.sleep(2)

time.sleep(2)
arm.go_xyz(65, 184, 80, v=30)
time.sleep(2)
arm.go_xyz(150, 0, 150, v=100)

arm.close()
```

Задание 2

В красной зоне (1; 10) установлена башня из деталей лего высотой от 1 до 8 модулей лего. Необходимо измерить высоту этой башни и переместить вправо фигуры с клетки (-7;5) на количество клеток равное высоте башни, выраженной в количестве модулей лего умноженному на 2. Пример перемещения показан на рисунке ниже (синим где фигура стояла исходно, зеленым куда её надо переместить, красным цветом обозначено положение башни высотой в 6 модулей).

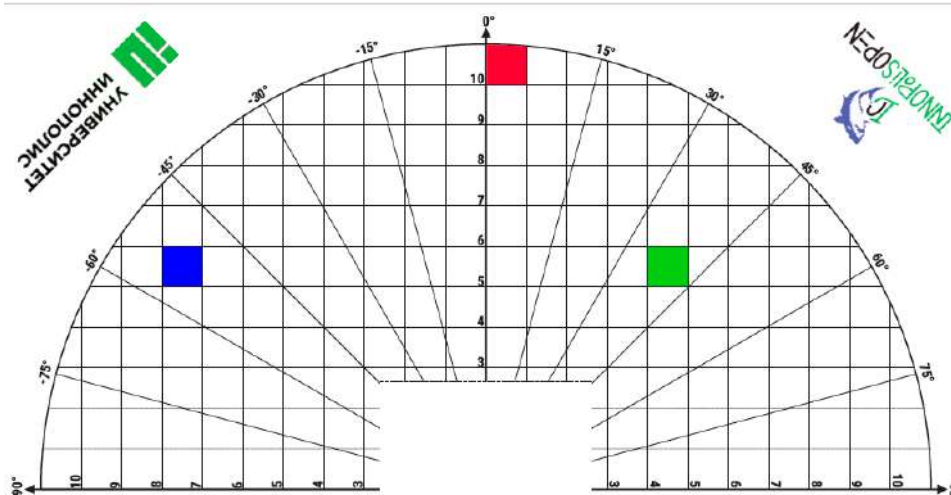


Рис. 2: Пример расстановки на игровом поле

Баллы за задание:

Башня осталась стоять на своем месте и не имеет повреждений	5 баллов
На светодиодную матрицу изображено число соответствующее высоте башни	10 баллов
Цвет надписи на матрице соответствует цвету вершины башни (красный, зеленый, синий, желтый)	10 баллов
Фигура поставлена в клетку с правильным адресом, опирается на поле всем своим основанием и не выходит за границы клетки	25 баллов
Фигура поставлена в правильную клетку, опирается на поле всем своим основанием и частично выходит за границы клетки	15 баллов

Максимальный балл: 50

Разбор решения — Задание 2

Идея решения

Задача делится на три части: определить высоту башни, определить её цвет и затем переместить фигуру на нужное расстояние. Высота измеряется датчиком расстояния, цвет — камерой, а перемещение считается по формуле из условия.

Общий принцип простой: «измерили → посчитали → переместили».

Определение высоты башни

Высота определяется с помощью датчика расстояния. Робот измеряет расстояние до башни и по этому значению определяет количество LEGO-модулей.

В коде это реализовано через последовательность условий:

```
if d < 96:
    h_tower = 8
elif d < 110:
    h_tower = 7
...
```

Идея здесь такая: чем ближе датчик к башне, тем она выше. Поэтому диапазоны расстояний заранее сопоставлены с высотой.

Важно понимать, что это не формула, а калибровка — значения подбираются экспериментально.

Определение цвета

Камера берёт изображение и анализирует небольшой квадрат в центре кадра. Далее считается средний цвет:

```
avg = np.mean(roi, axis=(0,1))
r, g, b = avg.astype(int)
```

После этого цвет упрощается до одного из заранее заданных (красный, жёлтый, синий и т.д.) через условия.

Это нужно, потому что реальные цвета «шумные», и напрямую их использовать нельзя.

Вывод на матрицу

После определения высоты и цвета робот выводит число:

```
arm.led_write(str(h_tower), r, g, b, 120)
```

Здесь: число — это высота башни, цвет — цвет её вершины.

Вычисление точки перемещения

По условию нужно сдвинуть фигуру вправо на $2 \times$ высоту.

В программе это сделано так:

```
y_target = -185 + h_tower * 2 * step
```

Где:

- `h_tower` — высота башни
- `step` — размер одной клетки

Таким образом робот автоматически вычисляет нужную позицию.

Перемещение фигуры

Движение реализовано в функции `go_to_target()`. Сначала робот поднимается, затем едет в нужную точку и опускает фигуру.

Обрати внимание на строки:

```
arm.go_xyz(135, y_target, 85, v=60)
arm.go_xyz(135, y_target, 20, v=60)
```

Сначала движение сверху (безопасно), затем опускание — это стандартный и правильный способ работы манипулятора.

Ключевые моменты

Главная сложность в этой задаче — это не движение, а работа с датчиками.

Во-первых, датчик расстояния требует калибровки: одинаковая высота может давать немного разные значения. Во-вторых, камера не всегда точно определяет цвет, поэтому используется усреднение и упрощение цвета.

Полный код решения

```
import cv2
import numpy as np

from colorsys import hls_to_rgb
from roboarm import RoboARM
import time

arm, port = RoboARM.auto_connect(baudrate=115200, tool_type='gripper', ack_timeout=2.5,
print('Connected RoboARM on', port)

def check_tower():
    global d
    global h_tower
    # инициализация датчика расстояния
    st, ok = arm.vl53l0_init(i2c_id=0, addr=0x29, timing_budget_us=100000, vcsel_pre=1
    print('VL53L0 init:', st, ok)

    time.sleep(3)
    # чтения расстояния
    st, d = arm.vl53l0_read()
    print('VL53L0 read:', st, d, 'mm')

    if d < 96:
        h_tower = 8
    elif d < 110:
        h_tower = 7
```

```
elif d < 126:
    h_tower = 6
elif d < 138:
    h_tower = 5
elif d < 141:
    h_tower = 4
elif d < 180:
    h_tower = 3
else:
    h_tower = 2

print(h_tower)

def go_to_target():
    step = 24.4
    y_target = -185 + h_tower * 2 * step
    rotate = 120 - h_tower * 12
    time.sleep(2)
    arm.go_xyz(135, y_target, 85, v=60)
    time.sleep(2)
    arm.gripper_set(rotate, 5)
    time.sleep(2)
    arm.go_xyz(135, y_target, 20, v=60)
    time.sleep(2)
    arm.gripper_set(rotate, 30)
    time.sleep(2)
    arm.go_xyz(135, y_target, 85, v=60)
    time.sleep(2)

def check_camera(cap):
    ret, frame = cap.read()
    if not ret:
        print('Ошибка чтения кадра')
        return None

    h, w, _ = frame.shape

    size = 100
    x1 = w//2 - size//2
    y1 = h//2 - size//2
    x2 = x1 + size
    y2 = y1 + size

    frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    roi = frame_rgb[y1:y2, x1:x2]

    avg = np.mean(roi, axis=(0,1))
```

```
r, g, b = avg.astype(int)

if r < 35 and g < 110 and b < 95:
    r, g, b = 101, 179, 68
elif r < 35 and g < 110 and b < 95:
    r, g, b = 44, 29, 207 # синий
elif r > 150 and g > 120 and b < 95:
    r, g, b = 255, 217, 0 # желтый
else:
    r, g, b = 255, 0, 0 # красный

print(f'{{r}}, {{g}}, {{b}}')

cv2.rectangle(frame, (x1,y1), (x2,y2), (0,255,0), 2)

cv2.putText(frame, f'R,G,B: {{r}},{{g}},{{b}}',
            (10,30),
            cv2.FONT_HERSHEY_SIMPLEX,
            0.8,
            (0,255,0),
            2)

cv2.imshow('USB Camera', frame)
cv2.waitKey(1)

return r, g, b

def led_number():
    arm.led_paint(0, 0, 0, 0)
    arm.led_write(str(h_tower), r, g, b, 120)
    time.sleep(3)

arm.gripper_set(78, 28)
arm.angles_set(127, 28, 2, v=60)
cap = cv2.VideoCapture(0)
r, g, b = check_camera(cap)
cap.release()
cv2.destroyAllWindows()

arm.angles_set(145, 16, -3, v=60)
time.sleep(2)

check_tower()
led_number()

time.sleep(2)
```

```
arm.go_xyz(130, -185, 85, v=60)
time.sleep(2)
arm.gripper_set(43, 28)
time.sleep(3)
arm.go_xyz(130, -194, 20, v=60)
time.sleep(3)
arm.gripper_set(43, 2)
time.sleep(3)
arm.go_xyz(142, -185, 85, v=60)
time.sleep(2)
```

```
go_to_target()
```

```
arm.close()
```

Задание 3

В указанной области на полигоне расположено несколько цветных фигур (от 1 до 5). Возможные формы фигур: треугольник, прямоугольник, ромб, круг. Возможные цвета фигур: желтый, красный, синий, зеленый, черный. Задача робота определить форму и цвет фигуры, имеющей наибольшую площадь и однозначно её изобразить на светодиодной матрице. За одну попытку робот должен распознать и определить 3 фигуры, на трех разных листах. Разработчик сам придумывает систему передачи робота информация о необходимости работы на следующей фигурой (не просто пауза в X секунд, а ожидание специального события «обработай следующую фигуру»). Для примера на рисунке ниже необходимо изобразить желтый круг.

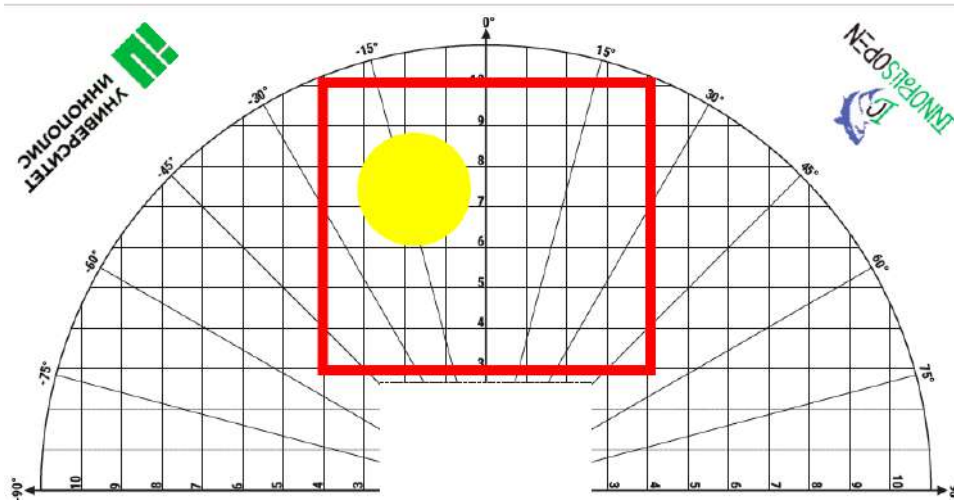


Рис. 3: Пример расстановки на игровом поле

Баллы за задание:

Правильно определена первая фигура	30 баллов
Правильно определена вторая фигура	20 баллов
Правильно определена третья фигура	10 баллов

Максимальный балл: 60

—

Разбор решения — Задание 3

Идея решения

Задача сводится к распознаванию фигуры по изображению с камеры. Робот получает кадр, находит на нём фигуру, определяет её форму и цвет, а затем отображает её на светодиодной матрице.

Основной принцип: «найти контур → определить форму → определить цвет → отрисовать».

—

Поиск фигуры на изображении

Сначала изображение упрощается: сглаживается и переводится в чёрно-белый вид. Это делается так:

```
gray = cv.cvtColor(roi, cv.COLOR_BGR2GRAY)
_, mask = cv.threshold(gray, 110, 255, cv.THRESH_BINARY_INV)
```

После этого остаётся только силуэт фигуры, что сильно упрощает дальнейшую обработку.

Далее находятся контуры:

```
contours, _ = cv.findContours(mask, cv.RETR_EXTERNAL, cv.CHAIN_APPROX_SIMPLE)
```

Из всех контуров выбирается самый большой — предполагается, что это и есть нужная фигура.

—

Определение цвета

Цвет определяется по пикселю в центре фигуры:

```
blue_led = roi[center_y, center_x][0]
green_led = roi[center_y, center_x][1]
red_led = roi[center_y, center_x][2]
```

Это простой и быстрый способ: центр всегда принадлежит самой фигуре, а не фону для выпуклых фигур, именно такие и были все фигуры на тестовых изображениях.

—

Определение формы

Форма определяется через количество вершин у контура. Контур упрощается до многоугольника:

```
approx = cv.approxPolyDP(largest_contour, epsilon, True)
f = len(approx)
```

Дальше логика очень простая: если 3 вершины — треугольник, если больше 10 — круг, если 4 — нужно дополнительное различие.

Различие прямоугольника и ромба

Для четырёхугольников используется следующий приём: сравнение площадей внутри контура и прямоугольника описанного вокруг фигуры (с учетом его поворота)

```
ratio = contourArea / area_of_rectangle
```

В коде:

```
ratio = round(cv.contourArea(largest_contour) / cv.contourArea(box), 2)
```

Если фигура сильно отличается от прямоугольника ($\text{ratio} < 0.9$), считается, что это ромб. Иначе — прямоугольник.

Это один из самых сложных и важных моментов решения.

Отрисовка на матрице

После определения фигуры вызывается соответствующая команда:

```
arm.led_draw('circle', (3,4,2), r, g, b, 100)
```

или, например, для треугольника:

```
arm.led_draw('polygon', [(1,6),(3,1),(6,6)], r, g, b, 100)
```

Таким образом робот не просто определяет фигуру, а «рисует» её заново.

Ключевые моменты

Главная сложность — обработка изображения. Нужно правильно выбрать порог бинаризации (110), иначе фигура либо исчезнет, либо сольётся с фоном.

Второй важный момент — аппроксимация контура. Если параметр `epsilon` выбран неправильно, фигура может определиться неверно (например, круг станет многоугольником).

Полный код решения

```
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
from testing.roboarm import RoboARM

# Инициализация робоманипулятора
arm = RoboARM(
    port='COM38',
    baudrate=115200,
    tool_type='gripper',
    ack_timeout=2.5,
    resp_timeout=10,
)

# Подключение к камере
cap = cv.VideoCapture(2)

# Переменная для хранения распознанной фигуры
figure = None

while True:
    # Считываем кадр с камеры
    ret, frame = cap.read()

    # Если кадр не получен, пропускаем итерацию
    if not ret:
        continue

    # Сглаживаем изображение для уменьшения шума
    frame = cv.GaussianBlur(frame, (5, 5), 0)

    # Выделяем область интереса (ROI)
    roi = frame[145:455, 155:520]

    # Переводим ROI в оттенки серого
    gray = cv.cvtColor(roi, cv.COLOR_BGR2GRAY)

    # Бинаризация изображения:
    # все пиксели темнее порога становятся белыми, остальные - черными
    _, mask = cv.threshold(gray, 110, 255, cv.THRESH_BINARY_INV)

    # Ищем внешние контуры на бинарном изображении
    contours, _ = cv.findContours(mask, cv.RETR_EXTERNAL, cv.CHAIN_APPROX_SIMPLE)

    # Если контуры найдены
    if contours:
        # Сортируем контуры по площади в порядке убывания
```

```
contours_sorted = sorted(contours, key=cv.contourArea, reverse=True)

# Берем самый большой контур
largest_contour = contours_sorted[0]

# Получаем ограничивающий прямоугольник
x, y, w, h = cv.boundingRect(largest_contour)

# Определяем центральную точку фигуры
center_x = x + w // 2
center_y = y + h // 2

# Считываем цвет пикселя в центре фигуры (BGR)
blue_led = roi[center_y, center_x][0]
green_led = roi[center_y, center_x][1]
red_led = roi[center_y, center_x][2]

# Отмечаем центр фигуры на изображении
cv.rectangle(
    roi,
    (center_x, center_y),
    (center_x + 1, center_y + 1),
    (0, 255),
    3,
)

# Пытаемся взять область фигуры с небольшим отступом
try:
    roi2 = cv.cvtColor(
        roi[y - 2:y + h + 2, x - 2:x + w + 2],
        cv.COLOR_BGR2GRAY,
    )
except Exception:
    # Если вышли за границы изображения - берем область без отступа
    roi2 = cv.cvtColor(roi[y:y + h, x:x + w], cv.COLOR_BGR2GRAY)

# Поиск углов
corners = cv.goodFeaturesToTrack(roi2, 10, 0.03, 15)

if corners is not None:
    corners = np.intp(corners)

# Аппроксимация контура многоугольником
epsilon = 0.01 * cv.arcLength(largest_contour, True)
approx = cv.approxPolyDP(largest_contour, epsilon, True)

# Количество вершин аппроксимированной фигуры
f = len(approx)

# Определяем тип фигуры по числу вершин
```

```
if f == 3:
    figure = "triangle"

elif f > 10:
    figure = "circle"

elif f == 4:
    # Для четырехугольников дополнительно различаем прямоугольник и ромб
    rect = cv.minAreaRect(largest_contour)
    box = cv.boxPoints(rect)
    box = np.intp(box)

    # Сравниваем площадь исходного контура с площадью минимального прямоугольника
    ratio = round(cv.contourArea(largest_contour) / cv.contourArea(box), 2)

    # Если отношение меньше 0.9 - считаем, что это ромб
    if ratio < 0.9:
        figure = "romb"
    else:
        figure = "rectangle"

    # Показываем ROI с отмеченной фигурой
    cv.imshow("cam", roi)

key = cv.waitKey(1) & 0xFF

# По нажатию q - выводим фигуру на LED-матрицу
if key == ord('q'):
    # Сначала очищаем подсветку
    arm.led_paint(0, 0, 0, 0)
    print(figure)
    if figure == "triangle":
        arm.led_draw(
            "polygon",
            [(1, 6), (3, 1), (6, 6)],
            red_led,
            green_led,
            blue_led,
            100,
        )

    elif figure == "circle":
        arm.led_draw(
            "circle",
            (3, 4, 2),
            red_led,
            green_led,
            blue_led,
            100,
        )
```

```
elif figure == 'romb':
    arm.led_draw(
        'polygon',
        [(3, 1), (5, 3), (3, 5), (1, 3)],
        red_led,
        green_led,
        blue_led,
        100,
    )

elif figure == 'rectangle':
    arm.led_draw(
        'rect',
        (2, 2, 5, 4),
        red_led,
        green_led,
        blue_led,
        100,
    )

# По нажатию w - завершаем программу
if key == ord('w'):
    break

# Освобождаем ресурсы
cap.release()
cv.destroyAllWindows()
```

Задание 4

Работа робота происходит в обозначенной зоне поля. Три фигуры случайным образом располагаются в левой (отрицательной) области рабочей зоны. Каждую из фигур необходимо переместить на позицию, симметричную по диагонали относительно точки (0;6). Пример показан на рисунке ниже (синим где фигуры стояли исходно, зеленым куда их надо переместить).

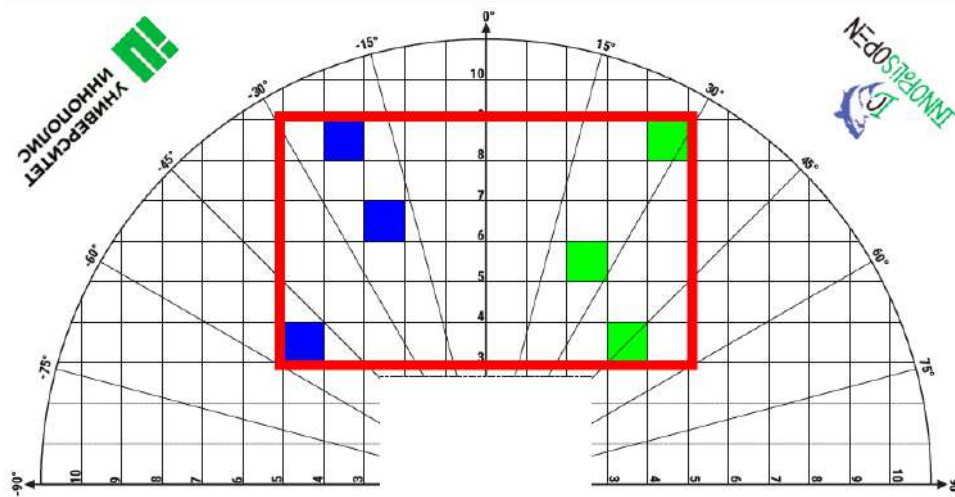


Рис. 4: Пример расстановки на игровом поле

Также робот должен обеспечить индикацию состояния поля до перемещений и после на светодиодной матрице. Красная рамка и внутри нее исходные позиции фигур синим, пауза не менее 2х секунд, карта с зеленым обозначением куда перемещаются фигуры.

	За каждую	Итого
Фигура поставлена в клетку с правильным адресом, опирается на поле всем своим основанием и не выходит за границы клетки	40 баллов	120 баллов
Фигура поставлена в правильную клетку, опирается на поле всем своим основанием и частично выходит за границы клетки	30	90
Правильно изображено поле до перемещений		25 баллов
Правильно изображено поле после перемещений		15 баллов

Максимальный балл: 160 —

Разбор решения — Задание 4

Идея решения

Задача состоит из трёх частей: найти фигуры на поле, вычислить их новые координаты (симметрию) и переместить их, а также отобразить поле до и после.

Общий принцип: «распознали клетки → посчитали новые координаты → перенесли».

Поиск фигур на поле

Поле разбивается на сетку (6 строк и 5 столбцов), и каждая клетка анализируется отдельно. Для каждой клетки считается средний цвет:

```
mean_color = cell.mean(axis=(0, 1))
```

Далее вычисляется, насколько цвет клетки отличается от среднего цвета всего поля. Идея простая: пустые клетки похожи друг на друга, а клетки с фигурами сильно отличаются.

Разница считается так:

```
diff = abs(c[2][0] - bg[0]) + abs(c[2][1] - bg[1]) + abs(c[2][2] - bg[2])
```

После этого выбираются 3 клетки с наибольшим отличием — это и есть позиции фигур.

—

Отображение поля

Перед перемещением робот показывает расположение фигур на матрице. Сначала рисуется рамка, затем клетки:

```
draw_cell(x, y, 0, 0, 255)
```

Синий цвет означает исходное положение, зелёный — конечное. Это помогает визуально проверить правильность работы.

—

Вычисление симметрии

Самая важная часть — правильно найти новую позицию.

Используется простая формула:

```
new_x = 6 - x
```

```
new_y = 7 - y
```

Это отражение относительно точки (0;6). То есть координаты «переворачиваются» относительно центра.

—

Перемещение фигур

Перемещение разбито на две функции: взять фигуру и поставить.

Например, при захвате координаты переводятся в реальные координаты робота:

```
x_shift = 210 - y * 25
```

```
y_shift = -113 + x * 25
```

Это перевод из «клеток» в миллиметры.

Далее робот: подъезжает сверху → опускается → захватывает → поднимает.

Аналогично работает функция постановки фигуры.

—

Ключевые моменты

Главная сложность — это сразу несколько задач в одной программе.

Во-первых, нужно правильно найти фигуры на изображении. Ошибка в выборе области или размера клетки приведёт к неправильным координатам.

Во-вторых, важно не перепутать систему координат: в изображении и у робота оси могут быть направлены по-разному.

Полный код решения

```
from roboarm import RoboARM
import time
import cv2 as cv
import numpy as np

cap = cv.VideoCapture(2)
arm, port = RoboARM.auto_connect(baudrate=115200, tool_type='gripper', ack_timeout=2.5,
print('Connected RoboARM on', port)

def find_figures(image):
    # Координаты прямоугольной области, где находится сетка
    x, y, w, h = 100, 165, (556 - 100) // 2, (440 - 165)

    # Количество столбцов и строк в сетке
    cols = 5
    rows = 6

    # Размер одной клетки
    cell_w = w // cols
    cell_h = h // rows

    # Список для хранения данных по каждой клетке
    cells = []

    # Проходим по всем клеткам сетки
    for row in range(rows):
        for col in range(cols):
            # Координаты текущей клетки
            x1 = x + col * cell_w
            y1 = y + row * cell_h
            x2 = x + (col + 1) * cell_w
            y2 = y + (row + 1) * cell_h

            # Вырезаем клетку из изображения
            cell = image[y1:y2, x1:x2]

            # Считаем средний цвет клетки
            mean_color = cell.mean(axis=(0, 1))
```

```
# Сохраняем номер клетки и её средний цвет
cells.append([col + 1, row + 1, mean_color])

# Находим средний цвет всего поля
bg = np.array([0.0, 0.0, 0.0])
for c in cells:
    bg += c[2]
bg = bg / len(cells)

# Для каждой клетки считаем отличие от среднего цвета поля
for c in cells:
    diff = abs(c[2][0] - bg[0]) + abs(c[2][1] - bg[1]) + abs(c[2][2] - bg[2])
    c.append(diff)

# Сортируем клетки по убыванию отличия
cells.sort(key=lambda c: c[3], reverse=True)

# Берём 3 самые отличающиеся клетки
result = []
for c in cells[:3]:
    result.append((c[0], c[1]))

# Возвращаем координаты клеток с фигурами
return result

# функция отрисовки одного пикселя на матрице
def draw_cell(x, y, r, g, b, bright=32):
    mx = x + 1
    my = y + 1
    arm.led_draw('rect', (mx, my, mx, my), r, g, b, bright)

# отрисовка рамки
def draw_board():
    arm.led_paint(0, 0, 0, 0)
    # красная рамка
    arm.led_paint(255, 0, 0, 32)
    arm.led_draw('rect', (1, 1, 6, 6), 0, 0, 0, 32)
    # правый край
    for y in range(1, 7):
        arm.led_draw('rect', (6, y, 6, y), 255, 0, 0, 32)

# матрица с расположением фигур до перемещения
def before_move(a, b, c):
    draw_board()
    # исходные позиции синим
    draw_cell(a[0] - 1, a[1] - 1, 0, 0, 255)
    draw_cell(b[0] - 1, b[1] - 1, 0, 0, 255)
    draw_cell(c[0] - 1, c[1] - 1, 0, 0, 255)
    time.sleep(2)
```

матрица с расположением фигур после перемещения

```
def after_move(a, b, c):  
    draw_board()  
    # конечные позиции зелёным  
    draw_cell(a[0] - 1, a[1] - 1, 0, 255, 0)  
    draw_cell(b[0] - 1, b[1] - 1, 0, 255, 0)  
    draw_cell(c[0] - 1, c[1] - 1, 0, 255, 0)  
    time.sleep(2)
```

функция забора фигуры с исходного места

```
def grab(x, y):  
    x -= 1  
    y -= 1  
    arm.go_xyz(150, 0, 150, v=50)  
    arm.gripper_set(0, 36)  
    x_shift = 210 - y * 25  
    y_shift = -113 + x * 25  
    time.sleep(2)  
    arm.go_xyz(x_shift, y_shift, 100, v=40)  
    arm.go_xyz(x_shift, y_shift, 50, v=40)  
    gr = 30 + y * 3 - x * 5  
    print(x_shift, y_shift, gr)  
    print()  
    time.sleep(2)  
    arm.gripper_set(gr, 36)  
    arm.gripper_set(gr, 36)  
    time.sleep(2)  
    arm.go_xyz(x_shift, y_shift, 20, v=40)  
    time.sleep(2)  
    arm.gripper_set(gr, 13)  
    time.sleep(3)  
    arm.go_xyz(x_shift, y_shift, 70, v=40)  
    time.sleep(2)  
    arm.go_xyz(150, 0, 150, v=50)  
    arm.gripper_set(0, 13)
```

функция постановки фигуры в новое место

```
def drop(x, y):  
    x -= 1  
    y -= 1  
    x_shift = 210 - y * 25.5  
    y_shift = 10 + x * 25.5  
    time.sleep(2)  
    arm.go_xyz(x_shift, y_shift, 70, v=40)  
    gr = -5 + y * 1 - x * 11.5
```

```
print(x_shift, y_shift, gr)
print()
time.sleep(2)
arm.gripper_set(gr, 13)
arm.gripper_set(gr, 13)
time.sleep(2)
arm.go_xyz(x_shift, y_shift, 20, v=40)
time.sleep(2)
arm.gripper_set(gr, 36)
arm.gripper_set(gr, 36)
time.sleep(3)
arm.go_xyz(x_shift, y_shift, 70, v=40)
time.sleep(2)
arm.go_xyz(150, 0, 150, v=50)
arm.gripper_set(0, 36)

# определяем новое расположение фигур
def calc_target(x, y):
    new_x = 6 - x
    new_y = 7 - y
    return new_x, new_y

# функция перемещения фигур с исходных координат в целевые
def move(x, y):
    new_x, new_y = calc_target(x, y)
    grab(x, y)
    drop(new_x, new_y)
    return new_x, new_y

# делаем снимок
arm.gripper_set(88, 0)
arm.go_xyz(150, 0, 175, v=50)
time.sleep(5)

ret, frame = cap.read()
coords = find_figures(frame)
print(coords)

# выводим на матрицу расположения фигур до перемещения
before_move(coords[0], coords[1], coords[2])

move(coords[0][0], coords[0][1])
move(coords[1][0], coords[1][1])
move(coords[2][0], coords[2][1])

# выводим матрицу расположения фигур после перемещения
after_move(calc_target(coords[0][0], coords[0][1]), calc_target(coords[1][0], coords[1][1]), calc_target(coords[2][0], coords[2][1]))

arm.close()
```

Задание 5

Роботу необходимо произвести сортировку фигур различной высоты и расставить их по возрастанию высоты слева направо. Самая высокая фигура имеет номер 3, средняя 2, низкая 1. Исходно фигуры стоят на клетках, указанных синим цветом. Во время сортировки разрешается переставлять их в зеленую зону. В зеленой зоне не может одновременно находиться более одной фигуры.

Робот может самостоятельно узнать высоты всех фигур или разрешается ввести ему расстановку в строку терминала программы, после её старта.

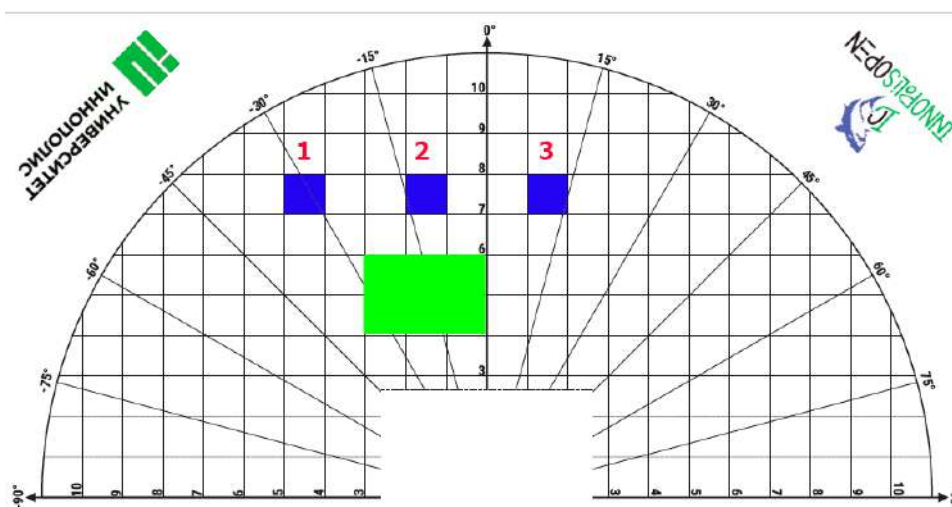


Рис. 5: Пример расстановки на игровом поле

Если фигура была расположена на своем целевом месте (например, самая низкая на позиции 1) и остальные фигуры не были поставлены правильно роботом в ходе попытки, то баллы за эту фигуры НЕ НАЧИСЛЯЮТСЯ!

	За каждую	Итого
Фигура расположена на месте соответствующем ее высоте (разрешается частичное размещение, но большая часть фигуры должна быть расположена на нужной клетке)	40 баллов	120 баллов
Робот работал без ввода информации о расстановки в консоль	-	40 баллов

Максимальный балл: 160

Разбор решения — Задание 5

Идея решения

Задача — отсортировать три фигуры по высоте. Это классическая задача сортировки, но с ограничением: робот может перемещать только одну фигуру за раз и есть дополнительная «временная» позиция.

По сути мы сортируем массив из трёх чисел, где: 1 — низкая фигура, 2 — средняя, 3 — высокая.

Сначала робот определяет высоты фигур, затем приводит их к правильному порядку.

—

Определение высоты

Высота определяется датчиком расстояния. В коде используется простое разбиение по диапазонам:

```
if d > 107:
    h_tower = 1
elif d > 96:
    h_tower = 2
else:
    h_tower = 3
```

Таким образом формируется массив, например:

```
arr = [2, 1, 3]
```

Это означает: первая фигура — средняя, вторая — низкая, третья — высокая.

—

Идея сортировки

Так как элементов всего три, не нужно использовать сложные алгоритмы. Проще разобрать все возможные случаи и для каждого прописать действия.

Например:

```
if arr == [2, 1, 3]:
    swap(arr, 0, 1, temp)
```

То есть если первые два элемента перепутаны — просто поменять их местами.

—

Обмен элементов (swap)

Важный элемент решения — как поменять местами две фигуры, если робот может держать только одну.

Используется временная позиция («temp»). Обмен делается в три шага:

```
move_tower(i, temp, size_i)
move_tower(j, i, size_j)
move_tower(temp, j, size_i)
```

Сначала убираем первую фигуру во временную зону, затем ставим вторую на её место, потом возвращаем первую.

Это классический приём, похожий на обмен переменных через временную переменную.

Сложный случай — цикл

Если все элементы стоят «по кругу» (например [3,1,2]), простой обмен не поможет. Тогда используется циклическая перестановка:

```
cycle(arr, temp)
```

Здесь фигуры «сдвигаются» по кругу, и каждая попадает на своё место.

Перемещение роботом

Функция `move_tower(from, to, size)` объединяет захват и установку.

Внутри используются координаты для каждой позиции. Например:

```
if from_pos == 0:  
    take_012(size, -116, 38)
```

Это перевод из номера позиции в реальные координаты робота.

Отдельно реализованы функции для временной зоны (“empty”).

Что здесь самое сложное

Главная идея — понять, что это задача не про робота, а про алгоритмы.

Нужно: правильно определить текущий порядок, выбрать нужный случай, и выполнить минимальное число перемещений.

Также важно аккуратно работать с временной позицией — если перепутать шаги, фигуры могут потеряться или оказаться не там.

Полный код решения

```
from roboarm import RoboARM  
import time
```

```
arm, port = RoboARM.auto_connect(baudrate=115200, tool_type='gripper', ack_timeout=2.5,  
print('Connected RoboARM on', port))
```

```
def check_tower():  
    global h_tower  
    # инициализация датчика расстояния  
    st, ok = arm.vl53l0_init(i2c_id=0, addr=0x29, timing_budget_us=100000, vcSEL_pre=1
```

```
print("VL53L0 init:", st, ok)

# чтения расстояния
st, d = arm.vl53l0_read()
print("VL53L0 read:", st, d, 'mm')

# определение высоты башни
if d > 107:
    h_tower = 1
elif d > 96:
    h_tower = 2
else:
    h_tower = 3

print(h_tower)
return h_tower

def take_012(size, y, grip):
    arm.go_xyz(150, 0, 150, v=50) # go to initial position
    arm.go_xyz(184, y, 80, v=50)
    arm.gripper_set(grip, 38)
    time.sleep(3)
    if size == 2:
        arm.go_xyz(184, y, 20, v=30)
    elif size == 3:
        arm.go_xyz(184, y, 28, v=30)
    else:
        arm.go_xyz(184, y, 10, v=30)

    time.sleep(3)
    arm.gripper_set(grip, 14)
    time.sleep(3)
    arm.go_xyz(184, y, 100, v=50)
    arm.go_xyz(150, 0, 150, v=50) # go to initial position
    arm.gripper_set(-2, 14)

def drop_012(size, y, grip):
    arm.go_xyz(150, 0, 150, v=50) # go to initial position
    arm.go_xyz(184, y, 80, v=50)
    arm.gripper_set(grip, 14)

    if size == 2:
        arm.go_xyz(184, y, 20, v=30)
    elif size == 3:
        arm.go_xyz(184, y, 28, v=30)
    else:
        arm.go_xyz(184, y, 10, v=30)

    arm.gripper_set(grip, 38)
```

```
arm.go_xyz(184, y, 100, v=50)
arm.go_xyz(150, 0, 150, v=50) # go to initial position
arm.gripper_set(-2, 38)

def take_empty(size):
    arm.go_xyz(150, 0, 150, v=50) # go to initial position
    arm.go_xyz(132, 0, 80, v=50)
    arm.gripper_set(-1, 38)

    if size == 2:
        arm.go_xyz(132, 0, 20, v=30)
    elif size == 3:
        arm.go_xyz(132, 0, 28, v=30)
    else:
        arm.go_xyz(132, 0, 10, v=30)

    arm.gripper_set(-1, 14)
    arm.go_xyz(132, 0, 100, v=50)
    arm.go_xyz(150, 0, 150, v=50) # go to initial position
    arm.gripper_set(-2, 14)

def drop_empty(size):
    arm.go_xyz(150, 0, 150, v=50) # go to initial position
    arm.go_xyz(132, 0, 80, v=50)
    arm.gripper_set(-1, 14)

    if size == 2:
        arm.go_xyz(132, 0, 20, v=30)
    elif size == 3:
        arm.go_xyz(132, 0, 28, v=30)
    else:
        arm.go_xyz(132, 0, 10, v=30)

    arm.gripper_set(-1, 38)
    arm.go_xyz(132, 0, 100, v=50)
    arm.go_xyz(150, 0, 150, v=50) # go to initial position
    arm.gripper_set(-2, 38)

def move_tower(from_pos, to_pos, size):
    #take
    if from_pos == 0:
        take_012(size, -116, 38)
        print('take from 0, size ', size)
    elif from_pos == 1:
```

```
        take_012(size, -41, 15)
        print('take from 1, size ', size)
    elif from_pos == 2:
        take_012(size, 30, -16)
        print('take from 2, size ', size)
    else:
        take_empty(size)

# drop
if to_pos == 0:
    drop_012(size, -116, 38)
    print('drop in 0, size ', size)
elif to_pos == 1:
    drop_012(size, -41, 15)
    print('drop in 1, size ', size)
elif to_pos == 2:
    drop_012(size, 30, -16)
    print('drop in 2, size ', size)
else:
    drop_empty(size)

def swap(arr, i, j, temp):
    size_i = arr[i]
    size_j = arr[j]

    move_tower(i, temp, size_i)
    move_tower(j, i, size_j)
    move_tower(temp, j, size_i)

    arr[i], arr[j] = arr[j], arr[i]

def cycle(arr, temp):
    target = [1, 2, 3]
    dest = [target.index(arr[i]) for i in range(3)]

    a = next(i for i in range(3) if dest[i] != i)
    b = dest[a]
    c = dest[b]

    size_a = arr[a]
    size_b = arr[b]
    size_c = arr[c]

    move_tower(c, temp, size_c)
    move_tower(b, c, size_b)
    move_tower(a, b, size_a)
```

```
move_tower(temp, a, size_c)

arr[a], arr[b], arr[c] = size_c, size_a, size_b

def sort_arr(arr, temp):
    print("start sort:", arr)

    if arr == [2, 1, 3]:
        swap(arr, 0, 1, temp)
        return

    if arr == [1, 3, 2]:
        swap(arr, 1, 2, temp)
        return

    if arr == [3, 2, 1]:
        swap(arr, 0, 2, temp)
        return

    if arr == [3, 1, 2] or arr == [2, 3, 1]:
        cycle(arr, temp)
        return

# массив для хранения изначальной позиций башен
arr = []

arm.go_xyz(150, 0, 150, v=50)
# едем к 1 башне
arm.go_xyz(153, -99, 100, v=50)
arm.gripper_set(35, 30)
arm.go_xyz(153, -99, 37, v=30)
# определяем размер башни
arr.append(check_tower())
arm.go_xyz(153, -99, 100, v=50)

# едем ко 2 башне
arm.go_xyz(153, -27, 60, v=50)
arm.gripper_set(6, 30)
arm.go_xyz(153, -27, 37, v=30)
# определяем размер башни
arr.append(check_tower())
arm.go_xyz(153, -27, 100, v=50)

# определяем высоту третьей башни
arr.append(6 - sum(arr))

# сортировка башен
sort_arr(arr, 14)
```

Задание 6

Робот должен построить башню из пешек. Исходно пешки расположены на одной линии в соседних клетках (от 2 до 10). Участник сам выбирает необходимое количество фигур для попытки и точку строительства башни. Чем выше построена башня, тем больше можно заработать баллов.

Фигур в башне	баллы
2	5
3	10
4	20
5	30
6	40
7	55
8	70
9	85
10	100

Разбор решения — Задание 6

Идея решения

Задача — построить башню из пешек, последовательно перекладывая их одну на другую.

Основная идея решения: каждая новая пешка кладётся чуть выше предыдущей. Для этого используется переменная, которая увеличивает высоту подъёма с каждой итерацией.

Проще говоря: «взяли пешку → положили → увеличили высоту → повторили».

Управление высотой башни

Высота башни задаётся через переменную `delta`. Она увеличивается с каждой новой пешкой:

```
delta = 20 * (i - 1)
```

И используется в координатах:

```
arm.go_xyz(230, 0, 60 + delta, v=50)
```

Это означает: чем выше башня, тем выше нужно поднимать следующую пешку.

Захват пешки

Захват реализован в функции `grab(delta, i)`. Она учитывает номер пешки, чтобы правильно подстроить положение гриппера:

```
y_shift = i * 25 * 2
```

Также меняется угол захвата:

```
if i < 4:
    gr = 52 - i * 5 * 2
else:
    gr = 0 - (i - 4) * 17.5
```

Это нужно, потому что пешки расположены под разными углами.

—

Построение башни

Основной цикл выглядит так:

```
for i in range(1, 8):
    ...
    arm.go_xyz(230, 0, 12 + delta, v=30)
    arm.gripper_set(0, 30)
```

Сначала робот опускает пешку на нужную высоту, затем отпускает её. После этого вызывается `grab()` для следующей пешки.

Таким образом башня постепенно растёт.

—

На что обратить внимание?

Главная сложность — учёт высоты башни. Если не увеличивать координату *Z*, робот будет сталкиваться с уже поставленными пешками.

Второй важный момент — корректный захват. Для разных позиций используются разные углы захвата, иначе пешки могут выпадать.

Также важно соблюдать порядок действий: сначала подняться, потом ехать, потом опускаться. Нарушение этого порядка часто приводит к ошибкам.

—

Завершение программы

После построения башни робот отъезжает вверх и выводит надпись на матрице:

```
arm.led_write('B', 255, 0, 0, 120)
arm.led_write('C', 255, 0, 0, 120)
arm.led_write('E', 255, 0, 0, 120)
```

Это визуальный сигнал, что программа завершена.

—

Полный код решения

```
from roboarm import RoboARM
import time

arm, port = RoboARM.auto_connect(baudrate=115200, tool_type='gripper', ack_timeout=2.5,
print("Connected RoboARM on", port)

def grab(delta, i):
    # вычисляем сдвиг по Y
    y_shift = i * 25 * 2

    # вычисляем поворот гриппера
    if i < 4:
        gr = 52 - i * 5 * 2
    else:
        gr = 0 - (i - 4) * 17.5
    time.sleep(2)

    # поднимаем на безопасное расстояние от текущей пешки над башней
    arm.go_xyz(230, 0, 60 + delta, v=50)

    # поднимаем на безопасное расстояние в home position
    arm.go_xyz(150, 0, 60 + delta, v=30)

    # едем к следующей пешке с учетом delta и сдвига по y
    arm.go_xyz(154, -194 + y_shift, 60 + delta, v=50)
    arm.go_xyz(154, -194 + y_shift, 50, v=50)
    # раскрываем гриппер и поворачиваем на угол gr
    arm.gripper_set(gr, 36)
    time.sleep(2)
    # опускаемся к пешке
    arm.go_xyz(154, -194 + y_shift, 12, v=50)
    time.sleep(3)
    # закрываем гриппер и поворачиваем на угол gr
    arm.gripper_set(gr, 4)
    time.sleep(3)

    # поднимаем на безопасное расстояние с учетом сдвига по y
    arm.go_xyz(154, -194 + y_shift, 50, v=50)
    # поднимаем на безопасное расстояние с учетом сдвига по y и delta
    arm.go_xyz(154, -194 + y_shift, 60 + delta, v=50)
    # перемещаемся к башне на безопасном расстоянии с учетом delta
    arm.go_xyz(230, 0, 60 + delta, v=50)
    time.sleep(2)
    # поворачиваем гриппер
    arm.gripper_set(0, 5)
    print("grab ", i + 1)
```

```
arm.led_paint(0, 0, 0, 0)
arm.go_xyz(230, 0, 150, v=50)
arm.gripper_set(0, 30)

# берем первую пешку
grab(20, 0)

# перемещаем оставшиеся пешки
for i in range(1, 8):
    delta = 20 * (i - 1)
    print(i + 1, delta + 60)

    time.sleep(3)
    arm.go_xyz(230, 0, 60 + delta, v=30)
    arm.go_xyz(230, 0, 12 + delta, v=30)
    time.sleep(3)

    arm.gripper_set(0, 30)
    print('drop', i)

    grab(delta, i)

# перемещаемся на безопасное расстояние от башни
arm.go_xyz(230, 0, 195, v=30)
arm.gripper_set(0, 30)

# выводим на экран 'ВСЕ'
for _ in range(3):
    arm.led_paint(0, 0, 0, 0)
    arm.led_write('B', 255, 0, 0, 120)
    time.sleep(1)
    arm.led_paint(0, 0, 0, 0)
    arm.led_write('C', 255, 0, 0, 120)
    time.sleep(1)
    arm.led_paint(0, 0, 0, 0)
    arm.led_write('E', 255, 0, 0, 120)
    time.sleep(1)

arm.close()
```

Заключительный этап

Старшая возрастная группа

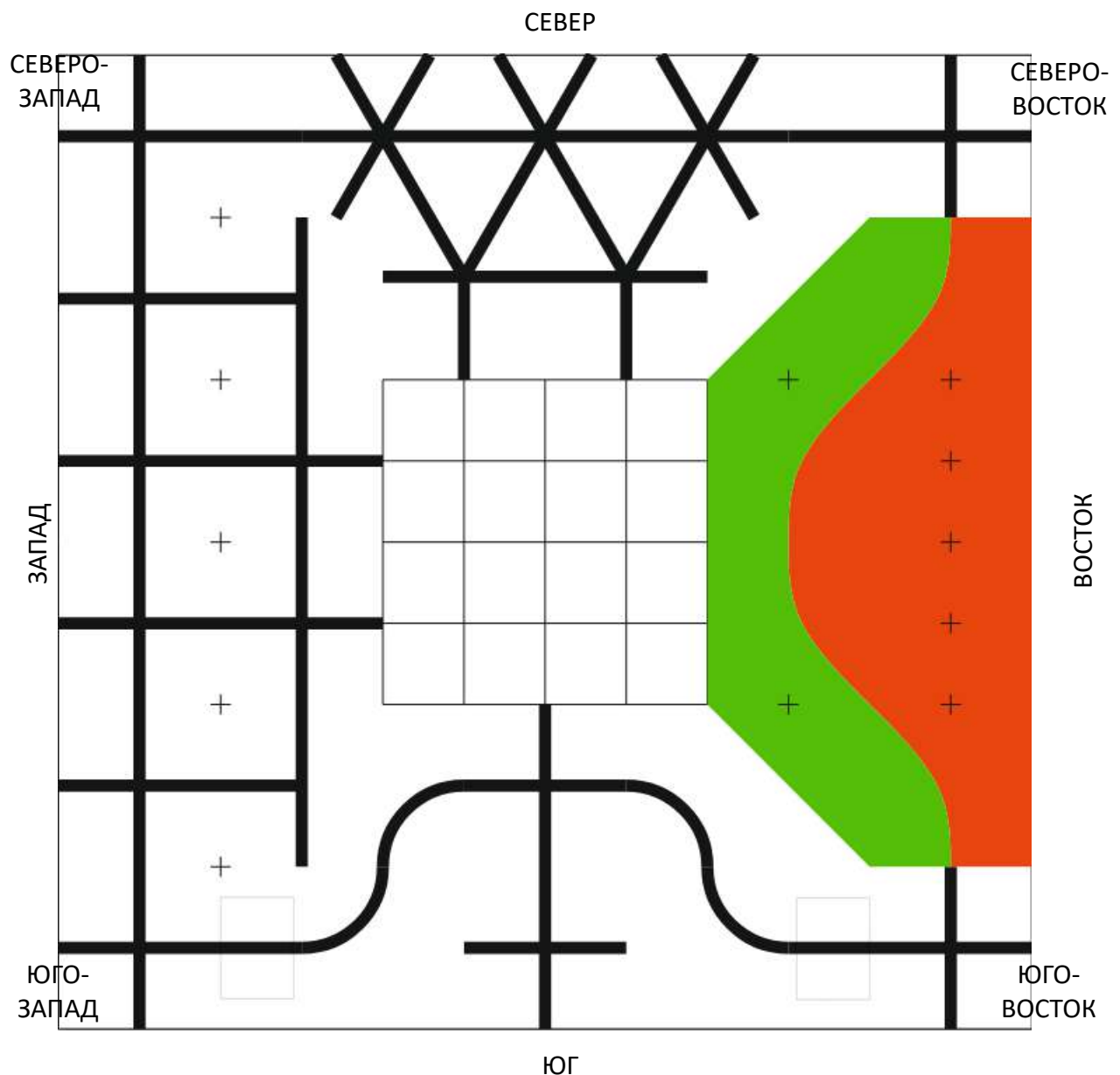
Задача первого дня

1. Краткое описание задачи:

Роботу необходимо найти путь между зонами полигона и переместить игровые объекты из зон погрузки в зону выгрузки. Ячейки зоны выгрузки, в которые необходимо доставить объекты, зашифрованы расположением игровых объектов.

Робот может, но не обязан, двигаться по *линиям разметки*.

2. Описание полигона:



- 2.1. *Зоны старта* – расположенные в углах полигона перекрёстки, обозначенные на схеме как СЕВЕРО-ВОСТОК, ЮГО-ВОСТОК, ЮГО-ЗАПАД и СЕВЕРО-ЗАПАД. Робот начинает движение в любом из них. Робот считается находящимся в зоне, если его проекция полностью закрывает пересечение линий перекрёстка (провода не засчитываются как проекция робота).
- 2.2. *Зона заданий* – зоны СЕВЕР, ВОСТОК, ЮГ и ЗАПАД, в которых роботу необходимо что-либо выполнить.
- 2.3. *Линии разметки* – линии на полигоне, позволяющие облегчить навигацию робота. Ширина всех линий разметки составляет 30 мм. Радиус плавных поворотов не менее 150 мм.
- 2.4. *Зоны погрузки* – линии разметки в форме крестов в зонах ЗАПАД и ВОСТОК, на которых могут быть установлены *игровые объекты*. В этот день *игровые объекты* могут располагаться случайным образом только на трёх центральных позициях каждой из зон. Ещё один *игровой объект* гарантировано расположен на центральном перекрёстке зоны ЮГ.
- 2.5. *Зона выгрузки* – расположена в центре полигона, в которую робот должен переместить *игровые объекты*. Зона разделена на 16 *секторов* для индивидуальной установки *игровых объектов* (границы секторов обозначены тонкими пунктирными линиями). Возле зоны выгрузки нанесены *линии разметки*, которые позволяют точно позиционироваться только на границах внешних *секторов*.
- 2.6. *Игровой объект* – куб размером 40х40х40 мм, весом около 3-4 г, который роботу необходимо переместить из *зоны погрузки* и северного перекрёстка зоны ЮГ в *зону выгрузки*.
- 2.7. *Препятствие* – объект шириной и длиной 48х48 мм и высотой не менее 70 мм, на полигоне расположено два препятствия: на центральном перекрёстке зоны СЕВЕР и на южном перекрёстке зоны ЮГ.

3. Миссии робота:

- 3.1. **Выезд из зоны старта / финиша (10 баллов)** – все точки проекции робота (вид сверху) покинули пересечение линий перекрёстка, на котором робот начал движение. Миссия засчитывается однократно, за возврат и повторный выезд, либо пересечение любого другого возможного стартового перекрёстка, баллы не начисляются. Выполнение миссии фиксируется и озвучивается судейской бригадой во время заезда робота.
- 3.2. **Проезд в зону старта (10 баллов)** – робот совершил перемещение в *зону старта*, в которой не начинал движение. Проекция робота (вид сверху) полностью закрывает пересечение линий перекрёстка *зоны старта*. Миссия засчитывается однократно для каждой *зоны старта*, отличной от той, в которой робот начинал движение; за повторный возврат в уже засчитанную зону дополнительные баллы не начисляются. Выполнение миссии фиксируется и озвучивается судейской бригадой во время заезда робота.
- 3.3. **Корректное вычисление столбцов зоны выгрузки (15 баллов)** – робот остановился в одной из *зон старта* не менее чем на 5 секунд и вывел на экран сообщение в формате X: NNNN (символ X, двоеточие и четыре цифры 0 или 1). Цифра 0 означает, что столбец должен остаться пустым. Цифра 1 означает, что в столбце могут находиться *игровые*

объекты. Миссия засчитывается однократно, за повторный вывод сообщения на экран баллы не начисляются. Выполнение миссии фиксируется и озвучивается судейской бригадой во время заезда робота.

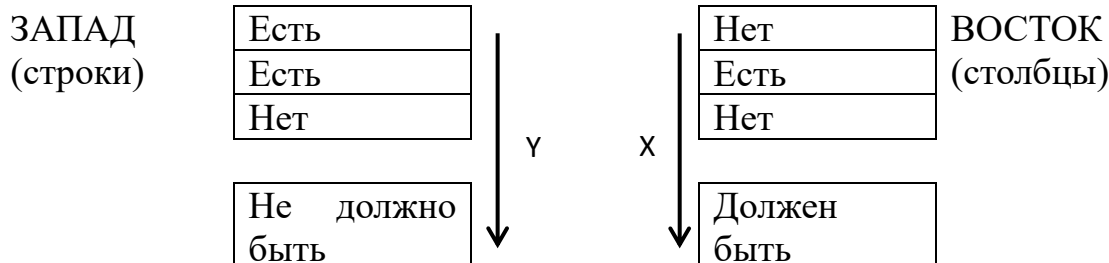
- 3.4. **Корректное вычисление строк зоны выгрузки (15 баллов)** – робот остановился в одной из зон *старта* не менее чем на 5 секунд и вывел на экран сообщение в формате Y: NNNN (символ Y, двоеточие и четыре цифры 0 или 1). Цифра 0 означает, что строка должна остаться пустой. Цифра 1 означает, что в строке могут находиться *игровые объекты*. Миссия засчитывается однократно, за повторный вывод сообщения на экран баллы не начисляются. Выполнение миссии фиксируется и озвучивается судейской бригадой во время заезда робота.
- 3.5. **Погрузка игрового объекта (5 баллов)** – *игровой объект* касается робота и не касается метки первоначальной установки объекта в зоне *погрузки*. Миссия засчитывается для каждого *игрового объекта* однократно, за выброс и повторную погрузку одного и того же *игрового объекта* баллы не начисляются. Выполнение миссии фиксируется и озвучивается судейской бригадой во время заезда робота.
- 3.6. **Выгрузка игрового объекта в зоне выгрузки (5 баллов)** – *игровой объект* касается только зоны *выгрузки* (белого поля; ограничивающая тонкая пунктирная линия не входит в зону выгрузки). Миссия засчитывается для каждого *игрового объекта* однократно. Выполнение миссии фиксируется и озвучивается судейской бригадой после остановки заезда робота.
- 3.7. **Выгрузка объектов в заданные сектора (10 баллов)** – *игровой объект* выгружен согласно предыдущей миссии и расположен индивидуально в каком-либо *секторе зоны выгрузки*: в этом секторе находится только он и никакой другой *игровой объект*, *препятствие* или робот. При этом строка и столбец сектора были зашифрованы расположением *игровых объектов* во время старта. Миссия засчитывается для каждого *сектора* однократно. Выполнение миссии фиксируется и озвучивается судейской бригадой после остановки заезда робота.
- 3.8. **Смещение препятствия (штраф 20 баллов)** – робот сдвинул *препятствие*. Штраф начисляется за смещение *препятствия* до потери касания линий пересечения перекрёстка или за изменение положения *препятствия* на горизонтальное. Штраф засчитывается однократно для каждого *препятствия*. Штраф фиксируется и озвучивается судейской бригадой после остановки заезда робота.

4. Жеребьевка:

- 4.1. Перед попыткой определяются два *бинарных кода*: для строк и для столбцов зоны *выгрузки*, в которые требуется перевезти *игровые объекты*. Кодирование осуществляется следующим образом:
- в зоне ВОСТОК в трёх центральных зонах *погрузки* (на красном цвете) устанавливается один или два *игровых объекта*; они кодируют столбцы зоны *выгрузки*;
 - в зоне ЗАПАД в трёх центральных зонах *погрузки* устанавливается один или два *игровых объекта*; они кодируют строки зоны *выгрузки*;
 - суммарно в зонах ВОСТОК и ЗАПАД устанавливается три *игровых объекта*;
 - наличие *игрового объекта* в зоне *погрузки* означает, что в соответствующем столбце или строке зоны *выгрузки* после финиша могут находиться *игровые объекты*;

- ближняя к СЕВЕРУ зона погрузки относится к первому столбцу или строке, средняя – ко второму столбцу или строке, дальняя – к третьему столбцу или строке;
- четвёртый столбец или строка вычисляется методом исключения: игровые объекты могут находиться строго в двух столбцах и двух строках.

4.2. Пример жеребьёвки:



4.3. Сообщения на экране робота для приведённой жеребьёвки:

- X:0101
- Y:1100

4.4. Итоговая расстановка объектов для приведённой жеребьёвки:

X	V	X	V
X	V	X	V
X	X	X	X
X	X	X	X

5. Порядок проведения тура:

- 5.1. Практический тур состоит из не менее чем двух зачетных попыток. *Попытка* – заезд всех роботов всех участников.
- 5.2. Перед каждой *попыткой* участникам предоставляется не менее 60 минут для подготовки. Время начала, остановки и продолжения подготовки едино для всех участников.
- 5.3. После времени подготовки объявляется *карантин*, во время которого участники должны прекратить изменять конструкцию или программный код робота.
- 5.4. Во время *попытки* участники по одному вызываются для совершения *заезда*. Участник может взять робота из зоны карантина, установить его на полигон в *зону старта / финиша* и запустить программу с именем IOR_2026 по команде судьи. Внесение каких-либо данных в программу робота путем нажатия кнопок на контроллере, вращения колес, изменения положения частей робота, беспроводным или любым другим способом запрещено.
- 5.5. Если после команды судьи и запуска программы робот не совершил движений (или участник понял, что программа или какой-либо модуль робота не запустился), то участник может попросить судью о перезапуске в течение 20 секунд. При этом набранные до перезапуска баллы будут потеряны. При этом допускается запустить программу с именем `hardware_test.py` и/или перезагрузить контроллер.
- 5.6. Во время заезда участник может находиться возле полигона и остановить попытку громким произнесением слова «СТОП», нажатием кнопки контроллера или

воздействием на датчик. Если после слова «СТОП» робот продолжит движение (в том числе по инерции), то участнику будут начислены баллы на момент остановки робота, а не на момент произнесения слова «СТОП» или нажатия кнопки (например, робот может сместить препятствие или выехать из какой-либо зоны). Любое воздействие участника на датчики робота будет расценено как остановка попытки и судейская бригада может остановить робота самостоятельно.

5.7. После *заезда* участник сдает робота в зону карантина. Подготовка продолжается только после окончания *попытки* – выполнения заезда всеми участниками.

6. Остановка заезда, баллы и рейтинг:

6.1. Заезд останавливается в следующих случаях:

Событие	Фиксирует остановку	Фиксируемые в протоколе баллы
Выгрузка четвёртого игрового объекта в зоне выгрузки	Судейская бригада	Набранные роботом к моменту выгрузки объекта
Закончилось время заезда	Судейская бригада	Набранные роботом за 180 секунд заезда
Робот остановился на 20 секунд	Судейская бригада	Набранные роботом до остановки на 20 секунд
Участник громко сказал «СТОП», нажал кнопку или воздействовал на датчик, робот больше не двигается	Участник	Набранные роботом на момент остановки попытки
Участник коснулся полигона, игрового объекта или препятствия	Судейская бригада	0 баллов

6.2. Набранные участниками в заездах баллы выстраиваются от большего (в лучшем заезде) до меньшего (в худшем заезде). Баллы лучшего заезда учитываются как результат дня. Общий рейтинг участников выстраивается в последний зачетный день по результатам всех дней.

7. Сводная таблица баллов:

Миссия	Балл	Макс. балл
Выезд из зоны старта (фиксируется во время заезда)	10	10
Проезд в зону старта (фиксируется во время заезда)	10	30
Корректное вычисление столбцов зоны выгрузки (фиксируется во время заезда)	15	15
Корректное вычисление строк зоны выгрузки (фиксируется во время заезда)	15	15
Погрузка игрового объекта (фиксируется во время заезда)	5	20
Выгрузка игрового объекта в зоне выгрузки (фиксируется после остановки заезда)	5	20
Выгрузка игрового объекта в заданные сектора (фиксируется после остановки заезда)	10	40
Смещение препятствия (фиксируется после остановки заезда)	-20	-40
Итоговые максимальные баллы:		150

Решение:

Участникам было предоставлено одинаковое оборудование (наборы «ТРИК. Учебная пара») и одинаковые мобильные платформы. В процессе работы над заданием участники могли самостоятельно устанавливать сенсоры и исполнительные органы, поэтому участники располагали значительно отличающимися версиями роботов, с различиями в установке датчиков, камер, схватов и прочей периферии. Вследствие этого единое эталонное решение, гарантировано и одинаково исполняемое на роботах всех участников, подготовить невозможно. Поэтому предлагается декомпозировать задачу на составляющие и в дальнейшем описать решение ключевых ее элементов.

При работе с реальным роботом необходимо корректно подключить и расположить датчики. Во-первых, у датчиков линии необходимо подобрать высоту, на которой показания будут в достаточно широком диапазоне, не менее 50% разницы между показаниями на белом фоне и чёрной линии. Во-вторых, у роботов ТРИК аналоговые порты преднастроены на определённые датчики. Необходимо убедиться, что настройки портов соответствуют подключенным к ним датчикам. Это можно сделать через веб-интерфейс управления контроллером ТРИК.

Если добиться околонулевых показаний на белом цвете и 100% на черном от датчика линии не удаётся, то можно использовать нормализацию – «отрезание» показаний выше и ниже

возможных и приведение оставшихся значений к диапазону 0..1. Пример кода одного из участников (язык программирования Python):

```
1. def norm(value, mn, mx):
2.     n = (value - mn) / (mx - mn) * 100
3.     if n < 0: return 0
4.     if n > 100: return 100
5.     return n
```

Здесь *mn* и *mx* – минимальное и максимально возможные показания датчика. Их можно запоминать заранее (во время калибровки) и обновлять во время выполнения программы. Но функция будет работать и без этого – в неё присутствуют проверки на выход за допустимый диапазон. Кстати, в этой функции диапазон нормализации увеличен до 0..100, чтобы сразу получать показания в процентах.

Декомпозируем задачу на составляющие, реализующие следующие алгоритмы:

- перемещение мобильного робота с дифференциальной платформой по линиям и дугам;
- перемещение мобильного робота с дифференциальной платформой на заданное расстояние, на заданный угол и т.д.;
- определение объектов в возможных зонах их установки и считывание кодов;
- захват, перемещение, выгрузка игровых объектов;
- считывание кодов и дополнение в них недостающих битов;
- определение секторов выгрузки объектов;
- вычисление координат центров секторов для выгрузки объектов.

В процессе выполнения задания робот мог перемещаться по заранее известному маршруту. Расположение всех зон и препятствий было известно заранее. Так, движение между зонами СЕВЕР, ВОСТОК, ЮГ и ЗАПАД могло выполняться по линиям. Движение по линиям отрабатывалось на втором отборочном этапе, например, в задаче «Движение по линии». Пример кода можно посмотреть в решении этой задачи. Участники могли использовать собственный код с этого этапа. В этой же задаче рассматривается движение по линии до перекрёстка.

Разметка в зоне СЕВЕР аналогична таковой из задачи «Сотовый лабиринт с линиями». Повороты и проезды по такой разметке разобраны в решении этой задачи.

В зоне ВОСТОК можно было перемещаться по границе линий, как в задачах «Движение по границе цветов» и «Одометрия по границе цветов» со второго отборочного этапа. Для движения по границе цветов можно использовать код из решения этих задач, но потребуется откалибровать датчики под цвета полигона.

Перемещения на заданное расстояние рассматривались в решении задачи «Циферблат» второго отборочного этапа. Там же описывается движение робота по гироскопу, которое может пригодиться при решении задачи первого дня. Пример кода одного из участников для поворота по гироскопу в заданном направлении (код незначительно изменён для лучшей читаемости, язык программирования Python)Ж

```
1. def gyro_rotate(target):
2.     global gyro
3.     while True:
4.         s = brick.gyroscope().read()
```

```

5.         angle = s[6] // 1000
6.         diff = (target - angle + 540) % 360 - 180
7.         if abs(diff) < 2:
8.             break
9.         if diff > 0:
10.            brick.motor("M3").setPower(-20)
11.            brick.motor("M4").setPower(20)
12.        else:
13.            brick.motor("M3").setPower(20)
14.            brick.motor("M4").setPower(-20)
15.        script.wait(5)
16.        brick.motor("M3").setPower(0)
17.        brick.motor("M4").setPower(0)

```

Для решения задания первого дня может потребоваться синхронизация моторов. Способы синхронизации моторов описаны в учебном пособии [«Управление моторами тележки с контроллером Трик на JavaScript»](#), которое рассматривалось на предыдущих этапах и сезонах олимпиады. Метод синхронизации колеса к колесу достаточно эффективен и прост в реализации за ограниченное время тура. Там же подробно описаны алгоритмы поворота по энкодерам на заданный угол.

Участникам были предоставлены одинаковые манипуляторы со схватами, с помощью которых можно было эффективно перемещать игровые объекты. Пример типичного кода одного из участников для управления сервоприводами манипулятора и схвата (язык программирования Python):

```

1. def down():
2.     brick.motor("s1").setPower(90)
3.
4. def up():
5.     brick.motor("s1").setPower(-90)
6.
7. def open():
8.     brick.motor("s2").setPower(-90)
9.
10. def close():
11.     brick.motor("s2").setPower(90)

```

Эти функции могут вызываться и отключаться через некоторое время, достаточное для выполнения сервоприводами нужных движений.

Записывать коды и вычислять недостающие биты удобнее с помощью массивов. При проездах важно вычислять наличие объекта в соответствующей зоне и записывать 0 или 1 в соответствующий элемент массива. Вычислять недостающий четвертый бит можно как разницу:

```

1. x[3]=2-x[0]-x[1]-x[2]
2. y[3]=2-y[0]-y[1]-y[2]

```

Выводить коды на экран можно следующим способом (пример кода одного из участников, язык программирования Python):

```

1. brick.display().clear()
2. brick.display().addLabel(f"x: {''.join(list(map(str, x)))}", 1, 1,
    50)
3. brick.display().redraw()

```

```
4. brick.playTone(1000, 100)
5. script.wait(5000)
```

Одна из возможных стратегий решения задачи:

1. Предварительная подготовка робота: установка двух датчиков линии на переднем бампере робота (с расстоянием между линзами датчиков около 25-30 мм, необходимо зафиксировать расстояние от оси колёс до датчиков линии, далее это расстояние обозначается как Sdoezd); установка ультразвукового датчика расстояния в направлении влево и инфракрасного датчика расстояния в направлении вправо относительно направления робота (необходимо зафиксировать расстояние, которое робот проезжает между обнаружением перекрёстка датчиками линии и попаданием проекции датчиков расстояния на перекрёстную линию, далее это расстояние обозначается как Ssens).
2. Предварительная калибровка: сохранение в программе показаний всех датчиков линии на белом, чёрном, красном и зелёном цветах. Для калибровки можно использовать элемент баннера, выданный каждому участнику.
3. Старт в зоне СЕВЕРО-ЗАПАД в направлении на ЮГ, с линией между датчиками.
4. Проезд по линии до первого перекрёстка, далее проезд на расстояние 200 мм + Ssens, чтобы ультразвуковой датчик расстояния оказался напротив первой зоны погрузки.
5. Считывание показаний ультразвукового датчика расстояния, определение наличия объекта в зоне погрузки.
6. Повторение пунктов 4-5 для считывания наличия объектов в трёх зонах погрузки зоны ЗАПАД и трёх битов этого кода.
7. Вычисление недостающего бита кода.
8. Проезд на два перекрёстка вперёд, чтобы оказаться датчиками на перекрёстке ЮГО-ЗАПАДНОЙ зоны старта.
9. Проезд (желательно с торможением на Sdoezd, чтобы оказаться колёсами на перекрёстке).
10. Остановка на 5 секунд и вывод на экран кода.
11. Поворот налево на 90 градусов или до линии, чтобы между датчиками линии оказалась линия разметки, ведущая от ЮГО-ЗАПАДНОЙ зоны старта к ЮГО-ВОСТОЧНОЙ.
12. Проезд по линии до второго перекрёстка (повторение пунктов 8-9), чтобы оказаться в ЮГО-ВОСТОЧНОЙ зоне старта.
13. Далее необходимо проехать через зону ВОСТОК и считать в ней расположение объектов и код. Это можно сделать, перемещаясь по границе цветов, либо проездами на заданное расстояние. Второй способ может оказаться предпочтительнее при проблемах с калибровкой на зелёном и красном цветах. При перемещении по энкодерам удобнее направить робота между зонами погрузки в зелёной и красной цветовых зонах, чтобы робот совершал прямолинейное движение. Опционально можно использовать выравнивание по гироскопу, как в задаче «Циферблат» второго отборочного этапа. При этом считывание инфракрасного датчика расстояния (направленного вправо) должно производиться по показаниям энкодера, а начинать отсчёт этого расстояния можно от границы зелёной зоны.

14. Движение в пункте 13 должно завершиться в СЕВЕРО-ВОСТОЧНОЙ зоне старта. В ней необходимо остановиться, дополнить недостающий бит кода и вывести его на экран в течение 5 секунд.
15. После получения обоих кодов необходимо вычислить сектора, в которые требуется доставить игровые объекты. Необходимо вычислить координаты центров этих секторов относительно одно или нескольких точек, в которые гарантировано можно доехать по разметке (например, от перекрёстка в зоне ЮГ или центрального перекрёстка зоны ЗАПАД). Вычислить координаты можно математически, отталкиваясь от размеров секторов (200x200 мм) и используя измерительный инструмент, доступный на площадке проведения тура.
16. Далее логика перевозки объектов может быть как линейной (проезд через все возможные зоны погрузки и все сектора выгрузки), так и с различными условиями, в порядке по выбору участника. Движение по линиям и на заданные расстояния аналогичны тем, что приводились ранее.

Подобная стратегия позволяет не перемещаться по самой сложной зоне – СЕВЕР, при этом посетив все стартовые зоны и набрав за них баллы. Кроме того, такая стратегия позволяет набирать баллы на простых действиях в начале, так как ошибка и сход с дистанции у реального робота возможны, так как время на отладки на олимпиаде ограничено.

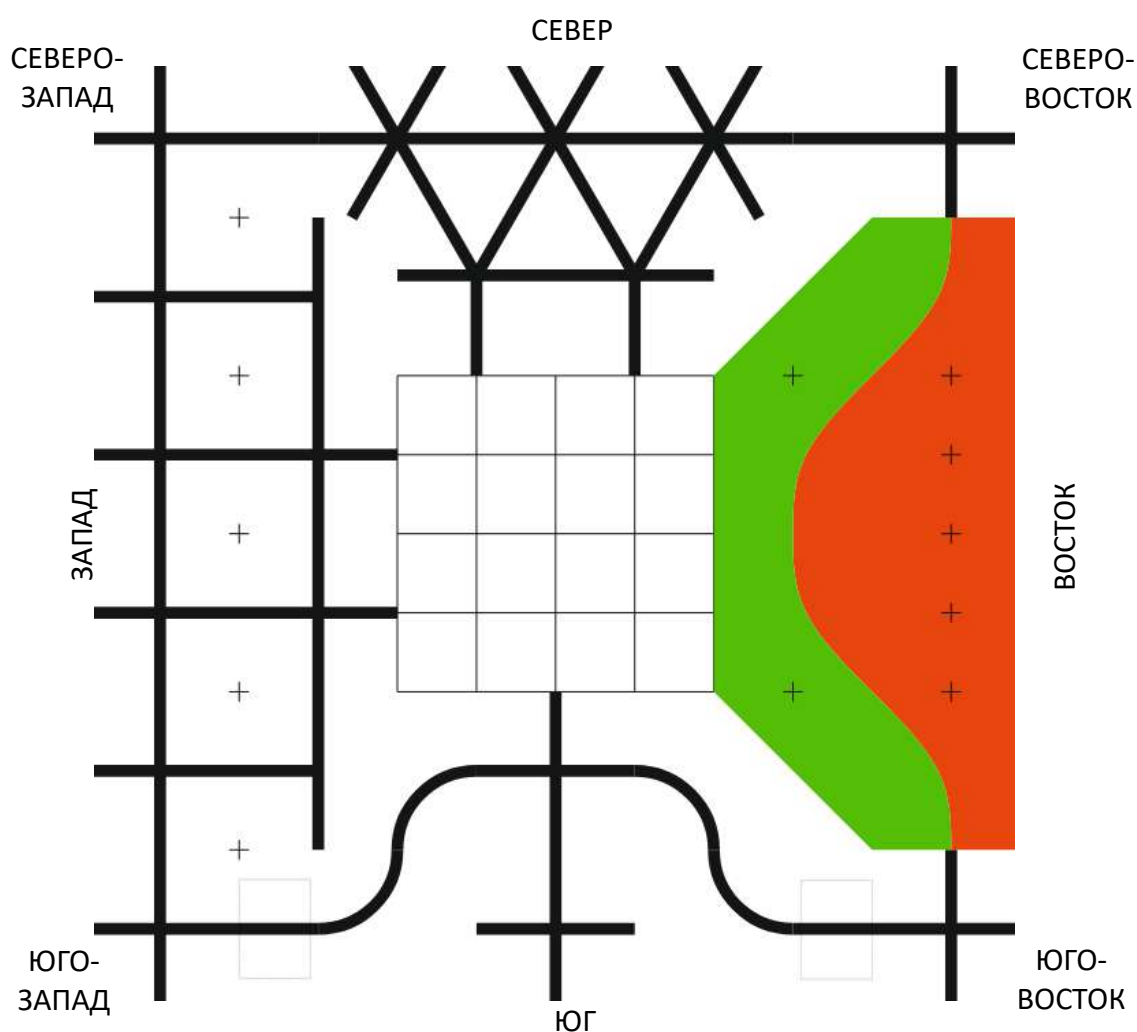
Задача второго дня

8. Краткое описание задачи:

Роботу необходимо найти путь между зонами полигона и переместить игровые объекты из зон погрузки в зону выгрузки. Ячейки зоны выгрузки, в которые необходимо доставить объекты, зашифрованы различными способами в зонах заданий: штрих-кодами, расположением объектов и т.д.

Робот может, но не обязан, двигаться по *линиям разметки*.

9. Описание полигона:



9.1. *Зоны старта* – расположенные в углах полигона перекрёстки, обозначенные на схеме как СЕВЕРО-ВОСТОК, ЮГО-ВОСТОК, ЮГО-ЗАПАД и СЕВЕРО-ЗАПАД. Робот начинает движение в любом из них. Робот считается находящимся в зоне, если его проекция полностью закрывает пересечение линий перекрёстка (провода не засчитываются как проекция робота).

9.2. *Зона заданий* – зоны СЕВЕР, ВОСТОК, ЮГ и ЗАПАД, в которых роботу необходимо что-либо выполнить.

- 9.3. *Линии разметки* – линии на полигоне, позволяющие облегчить навигацию робота. Ширина всех линий разметки составляет 30 мм. Радиус плавных поворотов не менее 150 мм.
- 9.4. *Зоны установки кодирующих объектов* – линии разметки в форме крестов в зонах ЗАПАД и ВОСТОК, на которых могут быть установлены *игровые объекты* для кодирования. Правила кодирования в зонах ВОСТОК и ЗАПАД описаны в разделе 4 Жеребьёвка.
- 9.5. *Зоны погрузки* – три северных перекрёстка в зоне СЕВЕР и южный перекрёсток в зоне ЮГ, на которых устанавливаются *игровые объекты*.
- 9.6. *Зона выгрузки* – расположена в центре полигона, в которую робот должен переместить *игровые объекты*. Зона разделена на 16 *секторов* для индивидуальной установки *игровых объектов* (границы секторов обозначены тонкими пунктирными линиями). Размер каждого сектора 200x200 мм. Возле зоны выгрузки нанесены *линии разметки*, которые позволяют точно позиционироваться только на границах внешних *секторов*.
- 9.7. *Игровой объект* – куб размером 40x40x40 мм, весом около 3-4 г, который роботу необходимо переместить из *зоны погрузки* и северного перекрёстка зоны ЮГ в *зону выгрузки*. *Кодирующие игровые объекты* могут отличаться по цвету.
- 9.8. *Препятствие* – объект шириной и длиной 48x48 мм и высотой не менее 70 мм, на полигоне расположено два препятствия: на центральном перекрёстке зоны СЕВЕР и на южном перекрёстке зоны ЮГ.

10. Миссии робота:

- 10.1. **Проезд в зону старта (10 баллов)** – робот совершил перемещение в *зону старта*, в которой не начинал движение. Проекция робота (вид сверху) полностью закрывает пересечение линий перекрёстка *зоны старта*. Миссия засчитывается однократно для каждой *зоны старта*, отличной от той, в которой робот начинал движение; за повторный возврат в уже засчитанную зону дополнительные баллы не начисляются. Выполнение миссии фиксируется и озвучивается судейской бригадой во время заезда робота.
- 10.2. **Корректное вычисление сектора выгрузки (20 баллов)** – робот остановился в одной из *зон старта* не менее чем на 5 секунд и вывел на экран сообщение в формате X: N. Вместо X подставляется сторона света (один из вариантов EAST, SOUTHEAST, SOUTHWEST, WEST). Вместо N подставляется номер сектора, зашифрованный кодом из соответствующей *зоны*. Миссия засчитывается однократно для одного из четырех кодов из зон ВОСТОК, ЮГО-ВОСТОК, ЮГО-ЗАПАД и ЗАПАД; за повторный вывод сообщения на экран баллы не начисляются. Выполнение миссии фиксируется и озвучивается судейской бригадой во время заезда робота.
- 10.3. **Выгрузка игрового объекта в зоне выгрузки (10 баллов)** – *игровой объект* касается только *зоны выгрузки* (белого поля; ограничивающая тонкая пунктирная линия не входит в зону выгрузки). Миссия засчитывается для каждого *игрового объекта* однократно. Выполнение миссии фиксируется и озвучивается судейской бригадой после остановки заезда робота.
- 10.4. **Выгрузка объектов в заданные сектора (25 баллов)** – *игровой объект* выгружен согласно предыдущей миссии и расположен индивидуально в каком-либо *секторе зоны выгрузки*: в этом секторе находится только он и никакой другой *игровой объект*, *препятствие* или робот. При этом номер сектора выгрузки ранее был *корректно*

вычислен (миссия 3.2.). Миссия засчитывается для каждого сектора однократно. Выполнение миссии фиксируется и озвучивается судейской бригадой после остановки заезда робота.

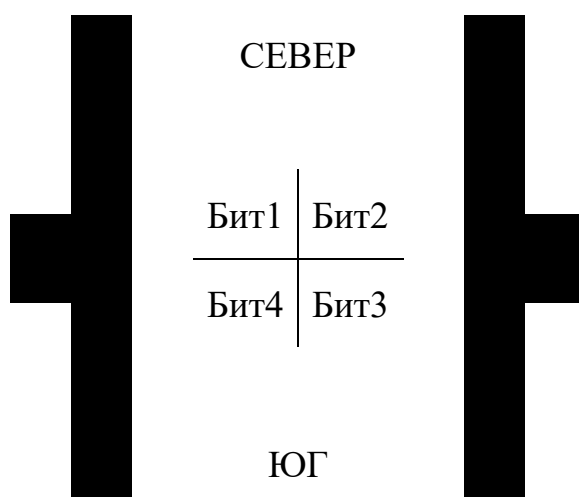
- 10.5. **Смещение препятствия (штраф 20 баллов)** – робот сдвинул *препятствие*. Штраф начисляется за смещение *препятствия* до потери касания линий пересечения перекрёстка или за изменение положения *препятствия* на горизонтальное. Штраф засчитывается однократно для каждого *препятствия*. Штраф фиксируется и озвучивается судейской бригадой после остановки заезда робота.

11. Жеребьевка:

- 11.1. Перед попыткой определяется «направление ветра»: СЕВЕРНОЕ или ЮЖНОЕ. СЕВЕРНЫЙ ВЕТЕР означает, что в кодах зон ВОСТОК и ЗАПАД старший бит находится с северной стороны. ЮЖНЫЙ ВЕТЕР – что старший бит с южной стороны. При «северном ветре» препятствие устанавливается на северный перекрёсток из тонких линий в зелёной зоне; при «южном ветре» – на южный.

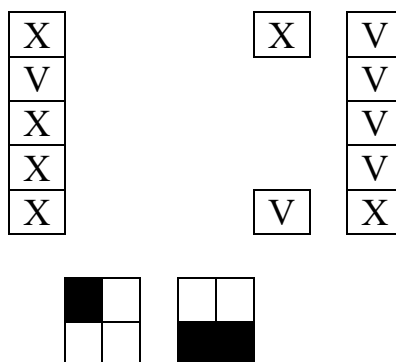
- 11.2. Далее определяются четыре *бинарных кода*, шифрующих *секторы доставки игровых объектов* в зоне *выгрузки*. Гарантируется, что числа во всех этих кодах находятся в пределах от 0 до 15 и отличаются между собой. Кодирование осуществляется следующим образом:

- в зоне ВОСТОК в пяти *зонах установки кодирующих объектов* (на красном цвете) устанавливается несколько *кодирующих игровых объектов*, образующих двоичный код – наличие кубика означает 1, отсутствие – 0;
- в зоне ЗАПАД в пяти *зонах установки кодирующих объектов* (на красном цвете) устанавливается несколько *кодирующих игровых объектов*, образующих двоичный код – наличие кубика означает 1, отсутствие – 0;
- в зоне юг устанавливается два *штрих-кода*: ЮГО-ВОСТОЧНЫЙ и ЮГО-ЗАПАДНЫЙ; формат и пример штрих-кода показаны ниже:



- ширина чёрных линий по границам кода – 30 мм, размер квадратов, кодирующих биты данных – 30x30 мм;
- белый цвет бита означает 1, чёрный – 0.

- 11.3. Пример жеребьевки:



- Ветер: южный;
 - Код в зоне ВОСТОК: $01111_2 = 15_{10}$;
 - Код в зоне ЗАПАД: $00010_2 = 2_{10}$;
 - Штрих-код ЮГО-ВОСТОЧНЫЙ: $1100_2 = 12_{10}$;
 - Штрих-код ЮГО-ЗАПАДНЫЙ: $0111_2 = 7_{10}$.
- 11.4. Сообщения на экране робота для приведённой жеребьёвки:
- EAST:15
 - WEST:2
 - SOUTHEAST:12
 - SOUTHWEST:7
- 11.5. Номера секторов:

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

12. Порядок проведения тура:

- 12.1. Практический тур состоит из не менее чем двух зачетных попыток. *Попытка* – заезд всех роботов всех участников.
- 12.2. Перед каждой *попыткой* участникам предоставляется не менее 60 минут для подготовки. Время начала, остановки и продолжения подготовки едино для всех участников.
- 12.3. После времени подготовки объявляется *карантин*, во время которого участники должны прекратить изменять конструкцию или программный код робота.
- 12.4. Во время *попытки* участники по одному вызываются для совершения заезда. Участник может взять робота из зоны карантина, установить его на полигон в *зону старта / финиша* и запустить программу с именем IOR_2026 по команде судьи. Внесение каких-либо данных в программу робота путем нажатия кнопок на контроллере, вращения колес, изменения положения частей робота, беспроводным или любым другим способом запрещено.

12.5. Если после команды судьи и запуска программы робот не совершил движений (или участник понял, что программа или какой-либо модуль робота не запустился), то участник может попросить судью о перезапуске в течение 20 секунд. При этом набранные до перезапуска баллы будут потеряны. При этом допускается запустить программу с именем `hardware_test.py` и/или перезагрузить контроллер.

12.6. Во время заезда участник может находиться возле полигона и остановить попытку громким произнесением слова «СТОП», нажатием кнопки контроллера или воздействием на датчик. Если после слова «СТОП» робот продолжит движение (в том числе по инерции), то участнику будут начислены баллы на момент остановки робота, а не на момент произнесения слова «СТОП» или нажатия кнопки (например, робот может сместить препятствие или выехать из какой-либо зоны). Любое воздействие участника на датчики робота будет расценено как остановка попытки и судейская бригада может остановить робота самостоятельно.

12.7. После заезда участник сдает робота в зону карантина. Подготовка продолжается только после окончания попытки – выполнения заезда всеми участниками.

13. Остановка заезда, баллы и рейтинг:

13.1. Заезд останавливается в следующих случаях:

Событие	Фиксирует остановку	Фиксируемые в протоколе баллы
Выгрузка четвёртого игрового объекта в зоне выгрузки	Судейская бригада	Набранные роботом к моменту выгрузки объекта
Закончилось время заезда	Судейская бригада	Набранные роботом за 180 секунд заезда
Робот остановился на 20 секунд	Судейская бригада	Набранные роботом до остановки на 20 секунд
Участник громко сказал «СТОП», нажал кнопку или воздействовал на датчик, робот больше не двигается	Участник	Набранные роботом на момент остановки попытки
Участник коснулся полигона, игрового объекта или препятствия	Судейская бригада	0 баллов

13.2. Набранные участниками в заездах баллы выстраиваются от большего (в лучшем заезде) до меньшего (в худшем заезде). Баллы лучшего заезда учитываются как результат дня. Общий рейтинг участников выстраивается в последний зачетный день по результатам всех дней.

14. Сводная таблица баллов:

Миссия	Балл	Макс. балл
Проезд в зону старта (фиксируется во время заезда)	10	30
Корректное вычисление сектора выгрузки (фиксируется во время заезда)	20	80
Выгрузка игрового объекта в зоне выгрузки	10	40

(фиксируется после остановки заезда)		
Выгрузка игрового объекта в заданные сектора (фиксируется после остановки заезда)	25	100
Смещение препятствия (фиксируется после остановки заезда)	-20	-20
Итоговые максимальные баллы:		250

Решение:

Перед решением задачи второго дня её необходимо декомпозировать на составляющие, реализующие следующие алгоритмы:

- перемещение мобильного робота с дифференциальной платформой по линиям и дугам;
- перемещение мобильного робота с дифференциальной платформой на заданное расстояние, на заданный угол и т.д.;
- определение объектов в возможных зонах их установки и считывание кодов;
- захват, перемещение, выгрузка игровых объектов;
- считывание кодов датчиками расстояния;
- считывание кодов датчиками линии;
- перевод кодов из двоичной системы счисления в десятичную;
- определение секторов выгрузки объектов;
- вычисление координат центров секторов для выгрузки объектов.

Сравнивая этот список с составляющими из первого дня, выявляется только два новых элемента: считывание кода датчиками линии и перевод кодов между системами счисления.

Один из штрих-кодов был озвучен участникам в начале второго дня. Они могли ввести его в память программы на этапе её написания. Поэтому участникам требовалось считывать лишь один штрих-код, ЮГО-ВОСТОЧНЫЙ. Пример кода одного из участников для считывания этого кода (немного изменён – удалены команды перемещения робота к коду и к зоне старта, язык программирования JavaScript):

```

1.  function read_code() {
2.      //...перемещение робота к коду
3.      getSensors();
4.      if (lineL < 20) {
5.          southeast += 8;
6.      }
7.      if (lineR < 20) {
8.          southeast += 1;
9.      }
10.     move_forward(80)
11.     getSensors();
12.     if (lineL < 20) {
13.         southeast += 4;
14.     }

```

```

15.     if (lineR < 20) {
16.         southeast += 2;
17.     }
18.     //...перемещение робота к любой зоне старта
19.     brick.display().clear();
20.     brick.display().addLabel('SOUTHEAST:' + southeast, 10, 10, 20)
21.     brick.display().redraw()
22.     script.wait(5000)
23. }

```

Перевод числа из двоичной системы в десятичную можно реализовывать с использованием встроенных функций языков программирования (например, в Python). Но можно использовать простой метод с умножением элементов массива на разные степени двойки. Пример кода одного из участников (на языке программирования JavaScript, но он применим для всех языков):

```

1. if (veter == 0) {
2.     west = Y[0]*16 + Y[1]*8 + Y[2]*4 + Y[3]*2 + Y[4];
3. }
4. if (veter == 1) {
5.     west = Y[4]*16 + Y[3]*8 + Y[2]*4 + Y[1]*2 + Y[0];
6. }

```

Основную стратегию можно изменить следующим образом:

1. Начало движения из зоны СЕВЕРО-ВОСТОК в направлении на ЮГ, чтобы первым же проездом считать не только код в зоне ВОСТОК, но и направление ветра. В этом случае код из зоны ВОСТОК можно выводить на экран уже в зоне ЮГО-ВОСТОК.
2. Перемещение между зонами в следующем порядке – через ВОСТОК к зоне ЮГО-ВОСТОК, через ЮГ к ЮГО-ЗАПАД, через ЗАПАД к СЕВЕРО-ЗАПАД.
3. Проезжая через зону ЮГ можно перевезти объект из этой зоны в зону выгрузки. К этому моменту будет известно уже три кода (из зоны ВОСТОК, ЮГО-ВОСТОК и ЮГО-ЗАПАД, так как он задан в начале дня). Для доставки можно выбрать зону с наибольшим номером из уже известных кодов – он будет ближе к зоне ЮГ.
4. Доставить объекты из зоны СЕВЕР в нужные сектора зоны выгрузки.

Логика выбора сектора в зоне выгрузки и проезда к нему могла быть реализована инструментами поиска и выбора элементов в массиве по индексу / значению, задачи на знания подобных инструментов предлагались участникам во время первого и второго отборов.

Разбор задач

1 Чайная лавка в Чэнду

Условие задачи

Путешествуя по Китаю, Ильнур оказался в старинной чайной лавке в городе Чэнду. В этой лавке чай готовят особым способом.

Сначала мастер наливает в чашку несколько одинаковых порций крепкого чайного настоя. Каждая порция имеет объём D миллилитров. Мастер может налить любое количество порций, но итоговый объём настоя всегда должен быть кратен D .

После этого мастер доливает горячую воду до полного объёма чашки.

У каждого посетителя есть своя чайная чашка объёма C миллилитров. Также посетитель сообщает, сколько воды он хотел бы получить в напиток: не меньше A миллилитров и не больше B миллилитров.

Таким образом, если мастер налил k порций настоя, то:

- объём настоя равен $k \cdot D$ миллилитров,
- объём воды равен $C - k \cdot D$ миллилитров.

Ильнуру стало интересно, может ли мастер выбрать такое количество порций настоя, чтобы объём воды оказался в желаемом диапазоне.

Вам даны числа A , B , C и D . Определите, существует ли целое число $k \geq 0$, для которого выполняется условие

$$A \leq C - kD \leq B.$$

Идея решения

Переформулируем условие. Требуется определить, существует ли целое число $k \geq 0$ такое, что

$$A \leq C - kD \leq B.$$

Здесь kD — объём настоя, а $C - kD$ — объём воды.

Ключевое преобразование

Перенесём выражение с k в другую сторону:

$$A \leq C - kD \leq B.$$

Это равносильно системе неравенств

$$C - B \leq kD \leq C - A.$$

Таким образом, задача сводится к проверке существования неотрицательного целого k , для которого число kD лежит в отрезке

$$[C - B, C - A].$$

Наблюдение

Все возможные значения объёма настоя имеют вид

$$0, D, 2D, 3D, \dots$$

то есть это кратные числа D .

Нужно проверить, существует ли кратное D , которое попадает в интервал

$$[C - B, C - A].$$

Минимальное кратное D , не меньшее чем $C - B$, равно

$$\left\lceil \frac{C - B}{D} \right\rceil \cdot D.$$

Если это значение не превосходит $C - A$, то существует подходящее значение k , иначе — нет.

2 Чередующийся маршрут

Условие задачи

Ильнур планирует путешествие по Китаю и выписал города в том порядке, в котором хочет их посетить.

Каждый город относится к одному из двух типов:

- R — город с историческими достопримечательностями;
- B — город с парками.

Типы городов заданы строкой s длины n , состоящей только из символов R и B.

Ильнуру нравится, когда маршрут выглядит разнообразно. Поэтому он хочет найти как можно более длинный участок маршрута, на котором типы городов чередуются. Это означает, что на таком участке любые два соседних города имеют разные типы.

Необходимо определить длину самого длинного непрерывного отрезка, на котором города чередуются.

Идея решения

Требуется найти максимальную длину непрерывного отрезка строки, на котором соседние символы различны, то есть выполняется чередование типов городов.

Наблюдение

Будем идти по строке слева направо и поддерживать длину текущего максимального суффикса, заканчивающегося в позиции i , на котором символы чередуются.

Обозначим эту длину через cur .

- Если $s_i \neq s_{i-1}$, то чередование продолжается, и можно увеличить текущую длину:

$$cur = cur + 1.$$

- Если $s_i = s_{i-1}$, то чередование ломается, и новый подходящий отрезок может начинаться только с позиции i , поэтому

$$cur = 1.$$

На каждом шаге достаточно обновлять ответ как максимум из всех значений cur .

Почему этого достаточно

Для каждой позиции i значение cur равно длине наибольшего чередующегося отрезка, который оканчивается в этой позиции.

Действительно:

- если текущий символ отличается от предыдущего, то любой корректный отрезок, заканчивавшийся в $i - 1$, можно продолжить;
- если символы совпадают, то никакой отрезок длины больше 1, заканчивающийся в i , уже невозможен.

Значит, максимум по всем таким значениям и будет ответом задачи.

Особые случаи

- Если $n = 1$, ответ равен 1.
- Если все символы одинаковы, ответ также равен 1.
- Если вся строка полностью чередуется, ответ равен n .

3 Чайная церемония

Условие задачи

Путешествуя по Китаю, Ильнур посетил традиционную чайную церемонию. Во время церемонии чайные мастера расставляют гайвани на специальном подносе, имеющем форму прямоугольной решётки размером $N \times M$.

В каждой позиции (i, j) находится одна гайвань. В неё кладут некоторое количество мини-точа — небольших прессованных чайных шариков. Количество мини-точа в гайвани обозначим через $A_{i,j}$.

Мастер объяснил Ильнуру важное правило расстановки. Для любых двух гайваней, расположенных в соседних клетках подноса (то есть имеющих общую сторону), суммарное количество мини-точа в них должно быть нечётным.

Формально, если клетки (i, j) и (x, y) являются соседними по стороне, то есть выполняется $|i - x| + |j - y| = 1$, то должно выполняться условие, что $A_{i,j} + A_{x,y}$ — нечётное число.

Иногда в некоторых гайванях оказывается недостаточно мини-точа, и правило может нарушаться. Ильнур может добавлять мини-точа в любую гайвань (то есть увеличивать значение $A_{i,j}$), однако удалять мини-точа нельзя.

Требуется изменить значения $A_{i,j}$ так, чтобы указанное правило выполнялось для всех соседних пар гайваней, при этом суммарное количество добавленных мини-точа было минимальным.

После внесения изменений выведите итоговую таблицу значений $A_{i,j}$.

Идея решения

Сумма двух чисел нечётна тогда и только тогда, когда одно из них чётное, а другое нечётное.

Значит, в любой корректной итоговой таблице чётности соседних клеток должны различаться. Но на прямоугольной решётке это означает ровно шахматную раскраску: все клетки одного цвета должны иметь одну чётность, а все клетки другого цвета — противоположную.

Следовательно, возможны только два варианта:

- клетки с $(i + j)$ чётным имеют чётные значения, а клетки с $(i + j)$ нечётным — нечётные;
- наоборот, клетки с $(i + j)$ чётным имеют нечётные значения, а клетки с $(i + j)$ нечётным — чётные.

Как минимально исправить одну клетку

Если для клетки требуется чётность $p \in \{0, 1\}$, то:

- если текущее число уже имеет чётность p , его менять не нужно;
- иначе достаточно увеличить его на 1.

Больше увеличивать никогда невыгодно, потому что нужно минимизировать суммарную прибавку.

Значит, для фиксированного шахматного шаблона каждая клетка исправляется независимо.

Два кандидата

Построим два варианта ответа.

Первый вариант. Требуемая чётность клетки (i, j) равна

$$(i + j) \bmod 2.$$

Второй вариант. Требуемая чётность клетки (i, j) равна

$$1 - ((i + j) \bmod 2).$$

Для каждого из двух вариантов:

- строим итоговую матрицу;
- считаем суммарное число прибавленных единиц.

Как выбрать ответ

- Если у одного варианта суммарная прибавка меньше, он лучше.
- Если суммы прибавок равны, нужно выбрать лексикографически меньшую матрицу.

Так как матрицы записываются построчно, обычное лексикографическое сравнение двух двумерных массивов даёт именно то, что требуется по условию.

4 Чайные наборы

Условие задачи

Путешествуя по Китаю, Ильнур посещает различные чайные магазины и покупает разные сорта чая. Каждый купленный пакетик он кладёт в чемодан. Если записать сорта чая в том порядке, в котором они лежат в чемодане, получится строка P , состоящая из строчных латинских букв. Каждая буква обозначает сорт чая.

Иногда Ильнур покупает чай не по одному пакетiku, а сразу небольшими наборами. В каждом таком наборе может быть несколько пакетиков чая, причём порядок сортов внутри набора может отличаться.

После возвращения из поездки Ильнур посмотрел на строку P и задумался: какими могли быть размеры наборов, которые он покупал.

Будем говорить, что строка P является *полиграммой*, если существуют целое число $k \geq 2$ и строки $S_1, S_2, \dots, S_k$, такие что

$$P = S_1 S_2 \dots S_k,$$

и все строки $S_1, S_2, \dots, S_k$ являются анаграммами друг друга.

Напомним, что две строки называются анаграммами, если одну из них можно получить из другой перестановкой символов.

Строка S_1 обозначается через S и называется *корнем полиграммы*. Её длина соответствует размеру чайного набора.

Вам дана строка P длины N . Необходимо определить все возможные длины корня полиграммы.

Идея решения

Пусть дана строка P длины N . Требуется найти все такие длины L , что строку можно разбить на

$$k = \frac{N}{L} \geq 2$$

подстрок длины L , и каждая из них является анаграммой первой.

Иными словами, если разбить строку на блоки длины L , то для всех блоков должны совпадать частоты всех букв.

Ключевое наблюдение

Если длина корня равна L , то обязательно выполняются два условия:

- $L < N$, так как блоков должно быть хотя бы два;
- L делит N , иначе разбить строку на равные блоки невозможно.

Следовательно, проверять нужно только делители числа N , меньшие N .

Для фиксированного L разобьём строку на блоки

$$P = B_1 B_2 \dots B_k, \quad |B_i| = L.$$

Тогда длина L подходит тогда и только тогда, когда для всех i строки B_i являются анаграммами друг друга.

Метод 1: префиксные частоты

Построим префиксные суммы по каждой букве: для каждой буквы c и каждой позиции i будем знать, сколько раз буква c встречается в префиксе длины i .

Тогда количество вхождений буквы c на отрезке $[l, r]$ можно получить за $O(1)$:

$$cnt_c(l, r) = pref_c(r) - pref_c(l - 1).$$

Следовательно, частотный вектор любого блока можно получить за

$$O(|\Sigma|),$$

где $|\Sigma|$ — размер алфавита. Сравнение двух блоков также занимает

$$O(|\Sigma|).$$

Для выбранного делителя L :

- находим частотный вектор первого блока;
- для каждого следующего блока длины L сравниваем его частоты с первым;
- если хотя бы для одного блока частоты отличаются, длина L не подходит;
- если все блоки совпали по частотам, длина L подходит.

Метод 2: весовой фильтр

Можно использовать более быстрый способ предварительной проверки. Назначим каждой букве некоторый вес w_c . Тогда вес блока определим как сумму весов его символов.

Если блок содержит символы $x_1, x_2, \dots, x_L$, его вес равен

$$W = w_{x_1} + w_{x_2} + \dots + w_{x_L}.$$

Если два блока являются анаграммами, то множества букв в них совпадают с учётом кратностей, а значит их веса тоже совпадают.

Чтобы быстро вычислять веса блоков, строится префиксный массив

$$pref[i] = w_{P_1} + w_{P_2} + \dots + w_{P_i}.$$

Тогда вес подстроки $[l, r]$ равен

$$pref[r] - pref[l - 1].$$

Для фиксированного L можно:

- посчитать вес первого блока;
- сравнить его с весами всех остальных блоков;
- если какой-то вес отличается, длина L точно не подходит.

Почему это только фильтр

Совпадение весов ещё не гарантирует, что строки являются анаграммами. Например, различные наборы букв могут иметь одинаковую сумму весов.

Следовательно, равенство весов является только необходимым условием.

Поэтому на практике удобно применять следующий подход:

- сначала отбрасывать большинство неподходящих длин по несовпадению весов;
- затем для оставшихся кандидатов выполнять точную проверку частот букв.

Особые случаи

- Если $N = 1$, строка не может быть полиграммой, так как требуется как минимум два блока.
- Если ни один делитель $L < N$ не проходит проверку, необходимо вывести -1 .
- Если подходящих длин несколько, их следует вывести в порядке возрастания.

5 Однотипные города

Условие задачи

Путешествуя по Китаю, Ильнур составил карту перемещений между городами. Каждый город относится к одному из двух типов:

- 0 — исторический город;
- 1 — современный город.

Между некоторыми парами городов существуют односторонние маршруты (например, скоростные поезда или внутренние авиарейсы). Известно, что по этим маршрутам невозможно вернуться в тот же город, то есть сеть маршрутов образует ориентированный ациклический граф.

Ильнур хочет добраться из города A в город B .

Он заметил, что ему быстро надоедают однотипные города, поэтому он не хочет посещать слишком много городов одного типа подряд.

Назовём *серией* максимальный подряд идущий фрагмент маршрута, в котором все города имеют одинаковый тип.

Маршрут считается допустимым, если длина любой серии на нём не превышает L .

Требуется определить количество допустимых маршрутов из города A в город B .

Поскольку число маршрутов может быть большим, выведите ответ по модулю $10^9 + 7$.

Идея решения

Дан ориентированный ациклический граф. У каждой вершины есть тип 0 или 1. Нужно посчитать количество путей из A в B , таких что длина любой максимальной подряд идущей серии вершин одного типа не превосходит L .

Иными словами, если на пути подряд идут несколько вершин одного цвета, то их количество не должно быть больше L .

Ключевая идея

Ограничение зависит не только от текущей вершины, но и от того, сколько вершин того же типа подряд уже набрано в текущей серии.

Поэтому естественно ввести динамику по состояниям:

$$dp[v][k]$$

— количество допустимых путей из A в вершину v , для которых текущая серия однотипных вершин, оканчивающаяся в v , имеет длину ровно k .

Здесь:

- $1 \leq k \leq L$;
- все значения считаются по модулю $10^9 + 7$.

Переходы

Пусть есть ребро $u \rightarrow v$.

- Если $c_u \neq c_v$, то при переходе в v начинается новая серия длины 1. Тогда все способы попасть в u можно перенести в состояние

$$dp[v][1].$$

- Если $c_u = c_v$, то текущая серия продолжается. Если в u серия имела длину k , то в v она станет длины $k + 1$. Такой переход допустим только при

$$k + 1 \leq L.$$

То есть переходы имеют вид:

- если $c_u \neq c_v$, то для всех k

$$dp[v][1] += dp[u][k];$$

- если $c_u = c_v$, то для всех $k < L$

$$dp[v][k + 1] += dp[u][k].$$

Начальное состояние:

$$dp[A][1] = 1,$$

поскольку путь из одной вершины A состоит из одной серии длины 1.

Ответ:

$$\sum_{k=1}^L dp[B][k].$$

Почему DAG сильно помогает

Так как граф ацикличен, его вершины можно обработать в топологическом порядке.

Когда мы рассматриваем вершину u , все значения динамики для неё уже полностью посчитаны, и можно безопасно распространять их по исходящим рёбрам.

Именно поэтому здесь получается обычная динамика по графу без рекурсии с мемоизацией и без риска заиклиться.

Замечания по подзадачам

- При $n \leq 20$ или $n, m \leq 2000$ можно было бы даже делать более прямолинейную динамику или поиск по путям.
- При $L = 1$ разрешены только пути, в которых типы соседних вершин всегда различны. В этом случае остаётся обычный подсчёт путей в DAG, но только по рёбрам между вершинами разных типов.
- Если граф является цепочкой, задача сводится к обычной динамике по массиву.

6 Два путешественника

Условие задачи

Путешествуя по Китаю, Ильнур и Глеб решили составить маршрут, чтобы посетить как можно больше интересных мест. Они изучили карту и выбрали n городов, которые хотят увидеть: от шумных мегаполисов до небольших древних поселений, расположенных вдоль восточного побережья страны.

Для каждого города известны его координаты (x_i, y_i) на карте. Города уже упорядочены по неубыванию координаты x , то есть

$$x_1 \leq x_2 \leq \dots \leq x_n.$$

Путешественники начинают своё путешествие вместе в городе 1. Изначально оба находятся в этом городе, поэтому город 1 считается посещённым ими обоими.

После старта каждый из путешественников может независимо от другого перемещаться между городами. Разрешается переходить только из города с меньшим номером в город с большим номером.

Каждый город с номером от 2 до n должен быть посещён хотя бы одним из путешественников.

Если путешественник перемещается из города i в город j , то длина такого перехода равна

$$\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}.$$

Суммарная длина маршрута равна сумме длин всех переходов, совершённых Ильнуром и Глебом.

Путешественники не обязаны заканчивать маршрут в одном и том же городе, и каждый из них может посетить любое количество городов.

Необходимо определить минимальную возможную суммарную длину маршрутов, при условии, что каждый город с номером от 2 до n будет посещён хотя бы одним из них.

Идея решения

Ключевая идея решения состоит в том, что для продолжения маршрута важно знать только текущие позиции двух путешественников. Все ранее посещённые города уже не влияют на стоимость будущих переходов.

Поэтому состояние решения можно описывать только двумя вершинами — городами, в которых сейчас находятся путешественники.

Ключевое наблюдение

Так как разрешено двигаться только из города с меньшим номером в город с большим номером, любой маршрут каждого путешественника представляет собой возрастающую по номерам последовательность городов.

Рассмотрим некоторый момент, когда уже покрыты все города с номерами от 1 до k , а путешественники находятся в городах i и j . Тогда обязательно

$$\max(i, j) = k.$$

Действительно, если бы оба номера были меньше k , то город k ещё не был бы посещён. Если же какой-то номер был бы больше k , то тогда уже был бы посещён и этот больший город, что невозможно.

Значит, состояние можно описывать парой последних посещённых городов:

$$dp[i][j]$$

— минимальная суммарная длина маршрутов после того, как посещены все города с номерами от 1 до $\max(i, j)$, а путешественники находятся в городах i и j .

Что означает состояние

Поскольку все города от 1 до $\max(i, j)$ уже покрыты, следующий город, который ещё не посещён, имеет номер

$$k = \max(i, j) + 1.$$

Именно этот город и нужно добавить следующим шагом. Посетить его может либо первый путешественник, либо второй.

Переходы

Пусть

$$k = \max(i, j) + 1.$$

Если $k \leq n$, то возможны два перехода:

- первый путешественник идёт из i в k , тогда новое состояние:

$$dp[k][j] = \min(dp[k][j], dp[i][j] + \text{dist}(i, k));$$

- второй путешественник идёт из j в k , тогда новое состояние:

$$dp[i][k] = \min(dp[i][k], dp[i][j] + \text{dist}(j, k)).$$

Здесь

$$\text{dist}(a, b) = \sqrt{(x_a - x_b)^2 + (y_a - y_b)^2}.$$

Начальное состояние:

$$dp[1][1] = 0,$$

потому что оба путешественника стартуют в городе 1, и никаких переходов ещё не сделано.

Почему этого достаточно

В любой допустимой конфигурации после покрытия городов $1, 2, \dots, k$ важно только следующее:

- где сейчас находится первый путешественник;
- где сейчас находится второй путешественник.

Вся более ранняя история уже не влияет на стоимость дальнейших переходов, потому что:

- все будущие города имеют номера больше k ;
- стоимость следующего шага зависит только от текущего положения того путешественника, который делает этот шаг.

Поэтому состояние $dp[i][j]$ содержит всю необходимую информацию для продолжения построения оптимального решения.

Почему каждый следующий город можно добавлять сразу

Так как города можно посещать только в порядке возрастания номеров, после того как уже покрыты все города до k , нельзя “перепрыгнуть” через город $k + 1$ и сначала посетить какой-то более поздний город.

Следовательно, на каждом шаге следующий ещё не покрытый город определяется однозначно:

$$k + 1.$$

Это и делает динамику квадратичной, а не экспоненциальной. Очень приятный момент: геометрия тут нужна только для расстояний, а вся комбинаторика сидит в порядке индексов.

Финальный ответ

Когда покрыты все города, должно выполняться

$$\max(i, j) = n.$$

Значит, ответом будет минимум по всем состояниям, в которых хотя бы один из путешественников находится в городе n :

$$\min(dp[n][i], dp[i][n])$$

по всем $1 \leq i \leq n$.

Так как таблица симметрична по смыслу, на практике можно просто взять

$$\min_{1 \leq i \leq n} dp[i][n].$$

или

$$\min_{1 \leq i \leq n} dp[n][i],$$

в зависимости от реализации.

Порядок вычисления

Состояния удобно обрабатывать в порядке возрастания

$$\max(i, j).$$

Например, можно перебирать все пары (i, j) , для которых значение уже определено, и делать переходы в город

$$\max(i, j) + 1.$$

Так как в каждом переходе максимум увеличивается ровно на 1, все зависимости направлены вперёд, и циклов нет.

Замечания по подзадачам

- При $n \leq 10$ или $n \leq 34$ можно было бы даже рассматривать перебор с мемоизацией, но общая динамика уже работает достаточно быстро.
- В случае $y_i = 0$ все города лежат на одной прямой, и расстояние между городами равно $|x_i - x_j|$, однако сама структура решения не меняется.