Задача командного тура

В командной части заключительного этапа участники должны спроектировать автономную систему управления роботом-погрузчиком для решения задач навигации на модели логистического центра с использованием алгоритмов компьютерного зрения.

Командная часть заключительного этапа имеет продолжительность 3,5 дня (всего 24 астрономических часа), которые включают работу по оснащению роботов необходимыми датчиками, программированию, пробные заезды на макете логистического центра, зачетные попытки.

9.1. Легенда

В автоматических логистических центрах практически не будет людей, всю работу будут выполнять роботы-погрузчики, которыми будет управлять интеллектуальное ПО для распределения задач.

Несмотря на отсутствие человеческого фактора, не следует думать, что в таких центрах никогда не будет никаких проблем. Форс-мажор может произойти и система из роботов и ПО должна уметь справляться с такими ситуациями.

Поэтому для финальной задачи профиля «Интеллектуальные робототехнические системы» предлагается рассмотреть следующий эпизод: в логистическом центре произошел инцидент с обрушением стеллажей, завалы привели к изменению структуры проездов по помещениям центра, поэтому робот-погрузчик должен уметь перемещаться в условиях измененного окружения.

В начале выполнения задания считается, что робототехническое устройство активируется в одном из секторов логистического центра, координаты сектора активации и структура логистического центра известны заранее. Робот-погрузчик должен переместиться в сектор распределения заданий, где должен считать ARTag маркеры - координаты сектора забора груза - следующей точки своего пути. При перемещении робот должен обнаруживать заваленные проходы и перестраивать свой маршрут без нанесения вреда упавшим стеллажам.

Задача участников Олимпиады - разработать программу управления робототехническим устройством для выполнения задания описанного выше.

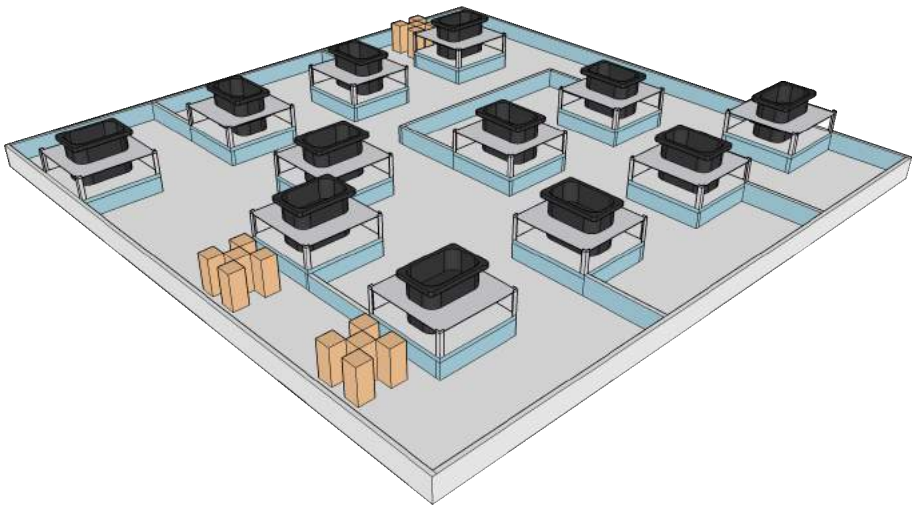

Рис. 9.1: Внешний вид полигона с тремя завалами

Рис. 9.2: Полигон, использовавшийся на финале Олимпиады НТИ

9.2. Набор заданий

Решение командной задачи было разбито на 3 этапа. Первые два этапа итеративно подводили участников к решению полной финальной задачи, осуществляемому во время последнего третьего этапа. На каждом этапе в проверку решения заданий данного этапа входили:

- способность проверить гипотезу о работоспособности алгоритма через демонстрацию решения в симуляторе;
- полнота решения задания конкретного этапа;
- воспроизводимость результатов — робототехническое устройство участников должно было неоднократно выполнить требуемые действия.

Первый этап

Задача: робототехническое устройство должно проехать по модели логистического центра из сектора старта в сектор финиша. Путь перемещения робота будет заранее выбран так, чтобы сектор старта не будет доступен при движении по правилу

правой или левой руки. Устройство должно заехать в сектор финиша, остановиться и вывести на экран количество секторов, которые находились справа от робота, но проезд к ним был перекрыт препятствием.

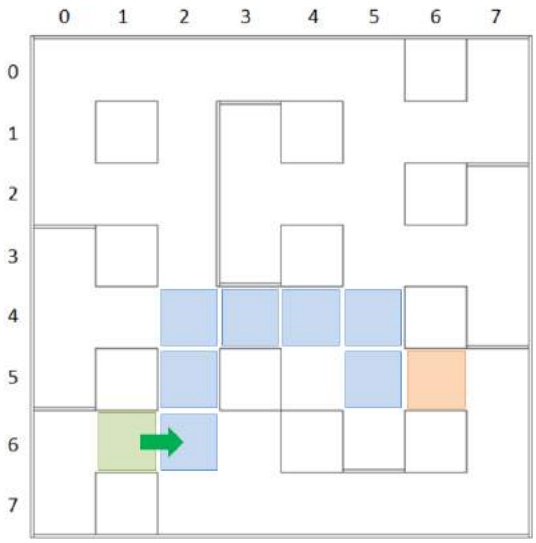

Рис. 9.3: Один из возможных оптимальных путей перемещения

Пример: Если путь перемещения робота, как на рис. 9.3, то во время движения робота были 3 сектора смежные с правой стороны с посещенными секторами, и при этом между ними располагалось препятствие.

Включая содержательные задачи:

- Сборка робототехнического устройства
- Реализация навигации
  - Выравнивание, используя синхронизацию моторов, и физические свойства среды;
  - Перемещение по азимуту с использованием показаний датчиков гироскопа или акселерометра.
  - Поворот на заданный угол с использованием показаний датчиков гироскопа или акселерометра.
- Реализация планирования маршрута
  - Реализация алгоритмов построения оптимальных маршрутов от одного сектора до другого;
  - Реализация алгоритмов автоматического планирования перемещения.
- Реализация сбора и обработки информации с датчиков
- Реализация счисления пути

Второй этап

Задача: робототехническое устройство должно проехать по модели логистического центра из сектора старта в сектор финиша через сектор распределения заданий. Координаты сектора финиша робот должен определить самостоятельно по ARTag меткам, расположенным на стеллаже, прилегающем к сектору распределения заданий.

Включая содержательные задачи:

- Калибровка камеры робототехнического устройства;
- Реализация алгоритмов компьютерного зрения: считывание ARTag меток без использования дополнительных библиотек;
- Декодирование бинарного кода, с использованием алгоритма обнаружения ошибок в кодах с битом четности.

Третий этап

Задача: робототехническое устройство должно проехать по модели логистического центра из точки старта сектор распределения заданий в сектор финиша, выявив все зоны, которые стали недоступны из-за обрушения стеллажей. Координаты сектора финиша должны будут определены роботом самостоятельно посредством декодирования ARTag маркеров.

Пример: На рис. 9.12 изображено поле, где недоступно 11 секций из-за обрушения стеллажей.

Включая содержательные задачи:

- Реализация алгоритмов построения карты;
- Реализация сопоставления шаблонов для выявления различий;
- Реализация алгоритмов обнаружения препятствий и перепланирования маршрута движения;
- Реализация алгоритмов оценки полноты исследования заранее известной карты.

9.3. Описание модели логистического центра

Полигон - квадратное поле 3200x3200 мм., разделенное на квадратные сектора 400x400 мм. Некоторые сектора отделены друг от друга перегородкой высотой 100 мм. Некоторые сектора недоступны для посещения робототехническим устройством и представляют из себя модель стеллажа высотой 210 мм. На каждой полке стеллажа располагается черный контейнер с грузом (рис. 9.4) размером $200 \times 300 \times 100$ мм (груз - 2-3 коробки (рис. 9.5) размером $180 \times 80 \times 85$ мм).

Полигон окружен бортом высотой 100 мм. Конфигурация полигона определяется в первый день финального этапа и объявляется участникам. Данная конфигурация будет использоваться все дни финального этапа.

На нижней полке стеллажа, прилегающего одной из четырех сторон к сектору распределения заданий, на высоте от 100-150 мм от уровня поверхности поля закреплены два ARTag маркера (`https://goo.gl/WaTFMB`), определяющие координаты сектора забора груза (сектор финиша). Размер каждого маркера - 30×30 мм. Расстояние между маркерами — $(70–150 \pm 5)$ мм. Маркеры обращены лицевой стороной внутрь сектора старта. Конкретная высота расположения маркеров определяется в первый день финала и остается постоянной во все дни финального этапа. При этом допустимая погрешность установки маркеров ± 5 мм. Пример расположения маркеров на стелаже представлен на рисунке 9.6

Рис. 9.4: Внешний вид контейнера

Рис. 9.5: Внешний вид коробки

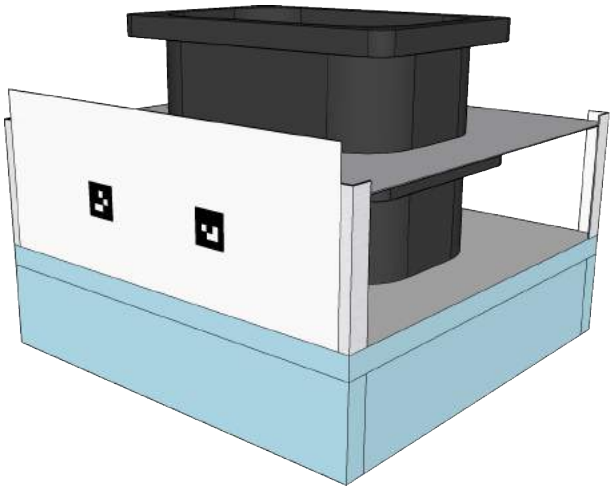

Рис. 9.6: Стеллаж с установленными ARTag маркерами

Маркер состоит из 5×5 элементов одинакового размера. Элементы маркера, расположенные по его границе - всегда черные. Четыре элемента, находящиеся в углах внутреннего 3×3 квадрата определяют ориентацию маркера таким образом, что только элемент в нижнем правом углу квадрата - белый. Центральный элемент квадрата используется для проверки четности (parity check): если количество единичных бит в двоичной записи закодированного в маркере числа нечетное, то он черный. Оставшиеся 4 элемента маркера кодируют число по следующему правилу: если элемент черный, то в он обозначает 1, если белый, то 0 при этом самый первый элемент — старший бит закодированного числа. Элементы пронумерованы сверху вниз, слева направо.

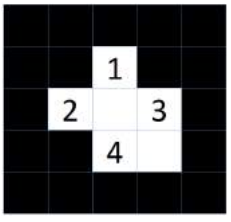

Рис. 9.7: Нумерация битов в маркера

Например, на маркере с рис. 9.8 закодировано число 0011_2, что эквивалентно 3_{10}.

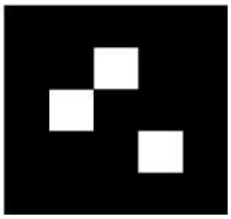

Рис. 9.8: Маркер с закодированным значением - 0011_2

Если закодированное на маркере двоичное число находится в диапазоне от 0_{10} до 7_{10}, то оно кодирует координату X сектора финиша. Если двоичное число — в диапазоне от 8_{10} до 15_{10}, то оно кодирует Y координату сектора финиша , при этом координата определяется вычитанием 8 из закодированного числа.

Рис. 9.9: Пример кодирования сектора

Левый маркер на рисунке 9.9 задает число $110_2 - 6_{10}$. Правый штрих-код задает число $1101_2 - 13_{10}$. Необходимый сектор находится в позиции с координатами $(6, 5)$ — см. рисунок 9.10.

Гарантируется, что сектор распределения заданий, будет располагаться в таких местах полигона, где к сторонам сектора прилегает, как минимум, один стеллаж.

Сектор активации робота, сектор распределения заданий и сектор забора груза никак не обозначаются на поле и определяются непосредственно перед каждым заездом робота.

Также на полигоне может присутствовать несколько завалов. Данные завалы состоят из 5 коробок, представленных на рисунке 9.5. Коробки располагаются вертикально. Пример завалов можно увидеть на рисунке 9.1.

9.4. Описание конструктора

В первый день финального тура каждой команде выдаются:

- Мобильная наземная платформа на базе конструктора TRIK в сборе (блок управления TRIK, аккумулятор, два мотора с энкодерами на датчиках Холла, колеса), но без установленных датчиков. Мобильная платформа построена по принципу дифференциального управления. Физические размеры платформы позволяют совершать все маневры внутри одного сектора модели логистического центра без касания со стенками стеллажей или бортов.
- Комплект дополнительных деталей из конструктора TRIK и набор датчиков:

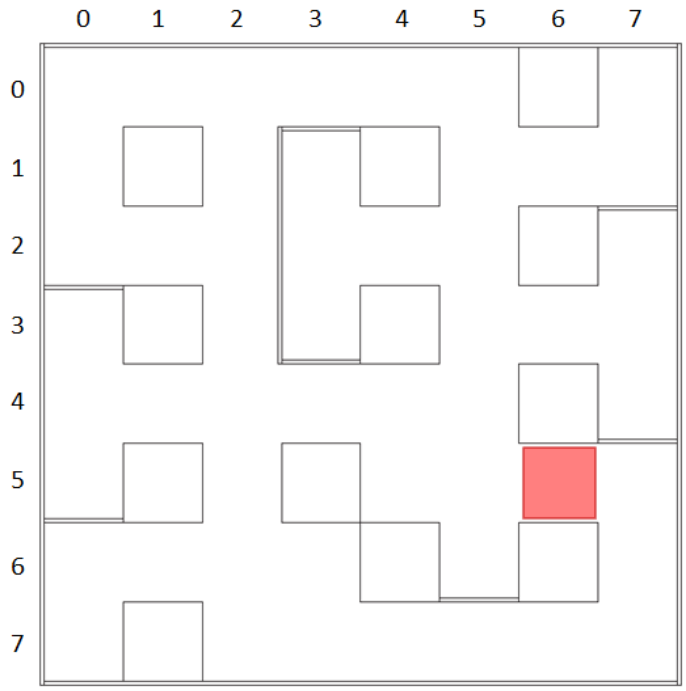

Рис. 9.10: Расположение сектора финиша для пары маркеров из рисунка 9.10

1 датчик освещенности, 2 инфракрасных датчика дальности, 2 ультразвуковых датчика расстояния и 1 VGA-камера;

- комплект дополнительных деталей из конструктора TRIK;
- Ноутбуки с установленной TRIK Studio, каждой команде по одному ноутбуку. При этом участники при этом могут пользоваться своими ноутбуками.

9.5. Условия проведения

- Из полученного набора датчиков команды могут выбирать те, с помощью которых, на их взгляд, можно решить задачу наиболее эффективным способом.
- Команды могут вносить любые изменения в мобильную наземную платформу.
- Участники во время командного этапа финального тура могут использовать интернет и заранее подготовленные библиотеки для решения задачи.
- Участники не могут использовать помощь тренера, сопровождающего лица или привлекать третьих лиц для решения задачи.
- Финальная задача формулируется участникам в первый день финального тура, но участники выполняют решение задачи поэтапно. Критерии прохождения каждого этапа формулируются для каждого дня финального тура. За подзадачи, решенные в конкретном этапе начисляются, баллы. Баллы за подзадачи можно получить только в день, закрепленный за конкретным этапом.
- Некоторые подзадачи строго требуют выполнение каких-то предыдущих подзадач. Выполнение данных подзадач, без выполнения предыдущих допускается, однако данная попытка будет оценена в 0 баллов.

Рис. 9.11: Мобильная платформа TRIK в сборе

- При выполнении подзадачи в виртуальной среде полное количество баллов можно получить лишь при первой попытке. При второй попытке сдать подзадачу может быть начислено лишь половина баллов за данную задачу. При последующих попытках баллы не начисляются.
- Во время рабочего времени команды могут проводить испытания на полигоне. Количество подходов, которое может сделать команда может быть ограничено в зависимости ограничений, накладываемых расписанием финального этапа Олимпиады.
- Испытания на полигоне должны осуществляться так, чтобы не мешать другим командам, проводящим в это время испытания на полигоне. Для осуществления этого всем командам может быть назначено ограничение по времени, которое они могут тратить на одно испытание. После истечения этого времени, команда должна дать возможность проводить испытания следующей команде.
- Часть подзадач необходимо будет решить в симуляторе TRIK Studio: команда получает 3 тестовых виртуальных полигона с соответствующими наборами входные данных для подготовки решения, в то время как приемка решения происходит на расширенном наборе полигонов для проверки универсальности управляющей программы. Начисление баллов за подзадачи может происходить только в тот этап, в котором данные подзадачи сформулированы.
- У команды есть не более двух попыток для сдачи решения подзадач в симуляторе:
  - Решения принимаются на проверку до истечения первых 4 часов работы в соответствующий соревновательный день.
  - До истечения четырех часов, команда должна загрузить свое решение на *Google Drive* в каталог, доступ к которому участники получат в

начале дня. Участники команды ответственны за то, что ссылка на каталог с их решениями не попадет участникам других команд.

  - В директории должен быть только один файл с решением.
  - До истечения указанного времени, команды могут изменять файл с решением сколько угодно раз. Проверяться будет всегда только последняя доступная версия.
  - Если решение отправлено на проверку в течение первых 3.5 часов работы в в соответствующий соревновательный день, то команда имеет право на вторую попытку, если результаты проверки решения ее не устраивают.
  - Если команда хочет воспользоваться правом проверки решения до истечения 3.5 часов, то она должна загрузить в каталог на *Google Drive* программу со своим решением и сообщить об этом судьям.

- Часть подзадач для реального робота могут быть запрещены к приемке без решения соответствующей подзадачи в симуляторе.
- Каждый день финального тура за 2 часа (может варьироваться в зависимости от расписания) до конца выделенного рабочего времени команды должны сдать роботов в зону карантина. Время сдачи роботов в карантин может изменяться и зависит от количества команд и сложности подзадач, принимаемых в конкретный этап.
- Перед сдачей робота в карантин команда должна загрузить на робота управляющую программу, подготовленную для демонстрации решения задачи, а также ее копию в *Google Drive* в каталог, доступ к которому участники получат в начале соревновательного дня. Без программы, загруженной в каталог *Google Drive*, команды не допускаются до проверки решения на реальном роботе.
- После момента, когда все роботы сданы в карантин, судьи вызывают команды по одной для приемки решения подзадач, закрепленных за этапом конкретного дня финального тура.
- Может быть предусмотрено до двух попыток сдачи решения одной и той же подзадачи на реальном роботе. Конкретное количество попыток определяется в конкретных подзадачах.
- После прохождения приемочных запусков, баллы набранные командой заносятся судьями в протокол. Один из участников команды расписывается за набранный результат, подтверждая согласие команды с оценкой проведенных запусков.
- Робот должен выполнять задание полностью автономно. Удаленное управление не допускается. Касание робота участником команды после его старта во время приемочных запусков не допускается. Алгоритм, реализующий систему управления робота должен планировать свое выполнение, полагаясь только на информацию с датчиков.
- Если подзадача подразумевает введение данных в программу (например, координат робота) до старта устройства. Такое разрешается только для тех задач, где это явно прописано. Во всех других случаях, введение данных в программу робота перед запуском запрещено.
- Если какая-то подзадача подразумевает считывание информации с элемен-

тов, расположенных на полигоне, запрещается при запуске робота вводить информацию о положении этих элементов или значениях, которые данные элементы определяют.

- Если во время приемочных запусков у судьи возникли сомнения о том, что задачи подэтапа решены корректно (робот не выполняет задачу полностью автономно, участник вводит значения в робота перед запуском), то он вправе провести инспекцию кода. По результатам инспекции, судья вправе снять с команды баллы, набранные за данный этап.
- Если во время приемочных запусков у судьи возникает ситуация, когда он не может однозначно решить выполняются ли критерии решения подзадачи, он вправе принять решение не в пользу команды.
- Команда вправе обсуждать с судьей результаты приемочных запусков до вызова следующей команды, но финальное решение остается о начислении баллов остается за судьей.

9.6. Процедура проведения приемочных запусков и критерии оценки

Первый этап

1. В качестве задачи для симулятора участникам необходимо решить следующую задачу:
    - Робот устанавливается в модели логистического центра в заранее неизвестном секторе. Задача робота проехать из точки старта в точку финиша. Гарантируется, что точка старта и точка финиша не будут находиться на пути перемещения робота по алгоритму правой или левой руки.

      Входные данные: через файл `input.txt` управляющей программе передается координаты сектора старта X_s, Y_s и направление старта D_s (направление робота от 0 до 3 начиная с направления вверх и дальше по часовой стрелке) - первая строка, сектор финиша X_f, Y_f - вторая строка, $0 \leqslant X_s, Y_s, X_f, Y_f \leqslant 7$. Три числа в первой строке разделены пробелом. Для чисел во второй строке разделитель - также пробел.

      Ожидаемый результат: после запуска программы в консоль выводится оптимальный путь перемещения робота от сектора старта в сектор финиша. После этого робот перемещается в сектор финиша. После остановки на экран робота выведено `finish`. Оптимальным является маршрут, при описании которого будет использоваться наименьшее количество символом, при использовании следующих обозначений: F - одна секция прямо, R- поворот направо на 90°, L - поворот налево на 90°: *например*, для маршрута, представленном на рис. 9.3, запись оптимального маршрута "`FLFFRFFFRFLF`".
    - Имя файла с управляющей программой для проверки решения в симуляторе: `sim-day1.js`.
2. Командам будет предоставлено две попытки на демонстрацию решения задачи на реальном роботе.

3. Первая попытка будет осуществляться 24 февраля, вторая - 25 февраля.
4. За 15 минут до перед каждой попыткой судья определяет сектора старта и направление робота в секторе старта, также определяется сектор финиша. Данные значения команда должна внести в программу перед тем, как сдать робота в карантин.
5. Правила именования файлов с программой управления:
   - для первой попытки: `day1-1.js`.
   - для второй попытки: `day1-2.js`.
6. Максимальное время выполнения одной попытки - 2 минуты.
7. Баллы за решение задач этапа:
   (a) **Симулятор:**
      i. Программа управления вывела оптимальный маршрут верно для всех проверочных полигонах — 4 баллов
      ii. Робот проехал из точки старта в точку финиша на всех проверочных полигонах — 8 баллов
   (b) **Реальный робот:**
      i. Робот покинул сектор старта — 4 баллов
      ii. Робот проехал половину от всего количества секторов, находящихся на оптимальном пути от сектора старта до сектора финиша — 8 баллов
      iii. Робот проехал от старта до финиша, остановился и вывел на экран `finish` — 16 баллов
      iv. Маршрут, по которому проехал робот от старта до финиша является оптимальным — 12 баллов
      v. Вместо `finish` на экран робота выведено верное количество перекрытых справа от робота секторов — 4 баллов
8. Баллы за подзадачи *iii*, *iv* и *v* не начисляются, если не было засчитано решение в симуляторе.
9. Баллы за две попытки суммируются.
10. Выполнение всех критериев в каждой из двух попыток дает дополнительные 32 баллов.
11. Максимальное количество баллов за этап — 132.

Второй этап

1. В качестве задачи для симулятора участникам необходимо решить следующую задачу:
   - Робот устанавливается в модели логистического центра в заранее неизвестном секторе. Задача робота проехать из точки старта в точку финиша, чьи координаты заданы изображениями ARTag маркеров, заранее считанным с камеры реального устройства.

     Входные данные: через файл `input.txt` управляющей программе передаются:

- В первой строке через пробел — координаты сектора старта X_s, Y_s и направление старта D_s (направление робота в секторе старта от 0 до 3 начиная с направления вверх и дальше по часовой стрелке), $0 \leqslant X_s, Y_s, \leqslant 7$;
- Во второй строке 19200, разделенных пробелом, целых чисел $P_{1,i}$ $(0 \leqslant P_{1,i} \leqslant 2^{32})$ — изображение первого ARTag маркера;
- Во третьей строке 19200, разделенных пробелом, целых чисел $P_{2,j}$ $(0 \leqslant P_{1,j} \leqslant 2^{32})$ — изображение второго ARTag маркера.

Каждое число в макере - точка, закодированная в формате RGB, т.е. строка с изображением маркера эквивалентна снимку разрешением 160×120 точек. Один маркер кодирует координату X финиша, второй - Y. Порядок маркеров не определяет порядок координат.

Ожидаемый результат: После запуска программы робот перемещается в сектор финиша. После остановки на экран робота выведено `finish`.

- Имя файла с управляющей программой для проверки решения в симуляторе: `sim-day2.js`.

2. Команде необходимо будет подготовить решения для трех разных подзадач для реального робота. На демонстрацию каждого решения предоставляется 2 попытки.
3. Все попытки осуществляются 26 февраля.
4. За 15 минут до времени сдачи роботов в карантин для 2ой и 3ей подзадач судья определяет сектора старта и направление робота в секторе старта, а также — сектор распределения задач (сектор, из которого необходимо сканировать ARTag метки) и направление расположения ARTag меток (0 - маркеры находятся на "верхнем" стеллаже, 1 - на "правом" стеллаже и т.д. по часовой стрелке).
   - Для второй подзадачи сектор старта и сектора распределения задач будет один и тот же для каждой попытки;
   - Для третьей подзадачи сектор старта и сектор распределения задач будут разные для разных попыток.

   Данные значения команда должна внести в программу перед тем, как сдать робота в карантин.
5. Правила именования файлов с программой управления:
   - для первой подзадачи: `day2-1.js`;
   - для второй подзадачи: `day2-2.js`.
   - для первой попытки третьей подзадачи: `day2-3-1.js`.
   - для второй попытки третьей подзадачи: `day2-3-2.js`.
6. Максимальное время выполнения одной попытки - 3 минуты.
7. Баллы за решение задач этапа:
   (a) **Симулятор**: робот проехал из точки старта в точку финиша на всех проверочных полигонах — 12 баллов

   (b) **Первая подзадача на реальном роботе:** Робот распознал верно ARTag метку, которая установлена прямо перед камерой, и вывел на экран верное значение, закодированное на метке — 4 балла

(c) **Вторая подзадача на реальном роботе:** Робот располагается за два сектора до сектора распределения задач, так что сектор старта расположен перпендикулярно сектору распределения задач.

i. Робот доехал до сектора распределения задач, остановился, издал звуковой сигнал, вывел на экран верное значение первой по ходу движения ARTag метки и продолжил движение через 10 секунд — 4 балла

ii. Робот вывел на экран верное значение второй по ходу движения ARTag метки и продолжил движение через 10 секунд — 4 балла

(d) **Третья подзадача на реальном роботе:** *полное выполнение задание второго этапа*

i. Робот проехал от старта до сектора распредения задач, остановился, издал звуковой сигнал, вывел на экран верные координаты сектора финиша и продолжил движение через 10 секунд — 28 баллов

ii. Робот остановился в секторе финиша и вывел `finish` — 4 балла

8. Баллы за третью подзадачу не начисляются, если не было засчитано решение в симуляторе.
9. Баллы за все попытки в каждой подзадаче суммируются.
10. Выполнение всех критериев в каждой из двух попыток всех трех подзадач дает дополнительные 32 балла.
11. Максимальное количество баллов за этап — 132.

Третий этап

1. В качестве задачи для симулятора участникам необходимо решить следующую задачу:
   - Робот устанавливается в модели логистического центра в заранее неизвестном секторе. При этом структура логистического центра известна заранее, но из-за завалов в нем образовалось N новых препятствий, заграждая путь, по которому может перемещаться робот. Задача робота проехать из точки старта в точку финиша, чьи координаты заданы изображениями ARTag маркеров, заранее считанным с камеры реального устройства. На финише необходимо вывести максимальное количество секторов во всем логистическом центре, которые стали недоступно для посещения роботом из-за появившихся завалов.

     Входные данные: через файл `input.txt` управляющей программе передаются:

     – В первой строке через пробел — координаты сектора старта X_s, Y_s и направление старта D_s (направление робота в секторе старта от 0 до 3 начиная с направления вверх и дальше по часовой стрелке), $0 \leqslant X_s, Y_s, \leqslant 7$;

     – Во второй строке 19200, разделенных пробелом, целых чисел $P_{1,i}$ ($0 \leqslant P_{1,i} \leqslant 2^{32}$) — изображение первого ARTag маркера;

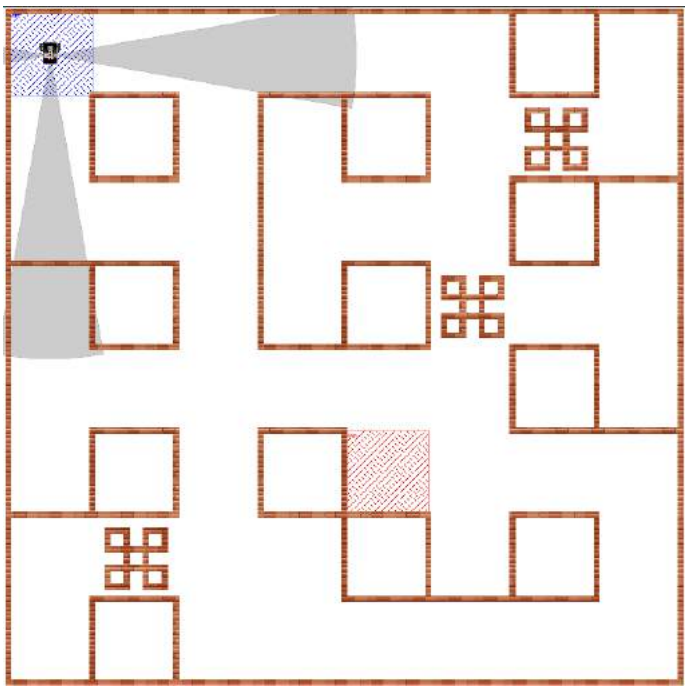

Рис. 9.12: 3 обрушения заблокировали 11 секторов

- Во третьей строке 19200, разделенных пробелом, целых чисел $P_{2,j}$ $(0 \leqslant P_{1,j} \leqslant 2^{32})$ — изображение второго ARTag маркера;
- В четвертой строке — количество завалов $(1 \leqslant N \leqslant 5)$.

Каждое число в маркере - точка, закодированная в формате RGB, т.е. строка с изображением маркера эквивалентна снимку разрешением 160×120 точек. Один маркер кодирует координату X финиша, второй - Y. Порядок маркеров не определяет порядок координат.

Ожидаемый результат первой подзадачи: После запуска программы робот перемещается в сектор финиша. После остановки на экран робота выведено `finish`.

Ожидаемый результат второй подзадачи: После запуска программы робот перемещается в сектор финиша. После остановки на экран робота выведено одно число количество секций недоступных для посещения роботом из-за завалов.

- Имя файла с управляющей программой для проверки решения в симуляторе: `sim-day3.js`.

2. Командам будет предоставлено две попытки на демонстрацию решения задачи на реальном роботе.
3. Первая попытка будет осуществляться 27 февраля, вторая - 28 февраля.
4. За 15 минут до перед каждой попыткой судья определяет сектора старта и направление робота в секторе старта, также определяется сектор распределения задач (сектор, из которого необходимо сканировать ARTag метки) и направление расположения ARTag меток (0 - маркеры находятся на "верхнем" стеллаже, 1 - на "правом" стеллаже и т.д. по часовой стрелке). Данные значения команда должна внести в программу перед тем, как сдать робота в карантин.
5. Правила именования файлов с программой управления:

- для первой попытки: `day3-1.js`.
- для второй попытки: `day3-2.js`.

6. Максимальное время выполнения одной попытки - 5 минут.
7. Баллы за решение задач этапа:
   (a) **Симулятор:**
      i. Робот проехал из точки старта в точку финиша и вывел `finish` на всех проверочных полигонах — 8 баллов
      ii. Робот проехал из точки старта в точку финиша и вывел количество недоступных секторов на всех проверочных полигонах — 24 балла

   Баллы не суммируются: принимается либо первая либо вторая подзадача.

   (b) **Реальный робот:** для проверки решения задачи этапа на реальном роботе на макете будет **ВСЕГДА** выставлено только 2 завала.
      i. Робот выехал со старта и остановился в секторе соседнем от сектора с первым по ходу движения завалом и воспроизвел звук, робот не касался элементов завалов до момента остановки рядом с данным завалом — 4 балла
      ii. После остановки робота в секторе с первым по ходу движения завалом на экран верно выведено минимально определяемое количество секторов недоступных для посещения, согласно изменениям на всей карте, исследованным к моменту остановки — 4 балла
      iii. Робот остановился в секторе соседнем от сектора со вторым по ходу движения завалом и воспроизвел звук, робот не касался элементов завалов во время выполнения задания до момента остановки рядом с данным завалом — 4 балла
      iv. После остановки робота возле сектора со вторым по ходу движения завалом на экран верно выведено минимально определяемое количество секторов недоступных для посещения, согласно изменениям на всей карте, исследованным к моменту остановки — 4 балла
      v. Робот доехал до сектора распределения задач, остановился, издал звуковой сигнал, вывел на экран верные координаты сектора финиша и продолжил движение через 10 секунд, до момента остановки в секторе распределения задач робот не касался элементов любых завалов — 4 балла
      vi. Робот остановился в секторе финиша и вывел `finish`, до момента остановки в секторе финиша робот не касался элементов любых завалов — 12 баллов
      vii. Вместо `finish` на экран робота верно выведено количество недоступных для посещения секторов — 8 баллов
8. Баллы за подзадачи *v*, *vi* и *vii* не начисляются, если не было засчитано решение любой из подзадач в симуляторе.
9. Баллы за все попытки в каждой подзадаче суммируются.

10. Выполнение всех критериев в каждой из двух попыток всех трех подзадач дает дополнительные 32 балла.
11. Максимальное количество баллов за этап — 136.

9.7. Решение

```
// Параметры робота
var pi = 3.141592653589793;
var d = 5.6 // Диаметр колеса, см
var l = 17.5 // База, см
var x = 0 // Начальные координаты робота
var y = 0
var a = 0

// Моторы
var mLeft = brick.motor(M4).setPower;
var mRight = brick.motor(M3).setPower;
var cpr = 360 // Показания энкодера за оборот

// Энкодеры
var eLeft = brick.encoder(E4);
var eRight = brick.encoder(E3);

// Длина клетки
var cellLength = 40 * cpr / (pi * d);

// Датчики расстояния
var svFront = brick.sensor(D1).read;
var svLeft = brick.sensor(A1).read;
var svRigth = brick.sensor(A2).read;

var readGyro = brick.gyroscope().read

function readYaw() {
  return -readGyro()[6];
}

var direction = 0; // absolute angle of direction movement
var directionOld = 0;
var azimut = 0; // we should go on azimut or turn to it
print("----------------------------------------");

eLeft.reset();
eRight.reset();

var el = eLeft.readRawData();
var er = eRight.readRawData();
mLeft(0);
mRight(0);

//инициализация и калибровка гироскопа
brick.gyroscope().calibrate(12000);
script.wait(13000);
print("gyro inited");
brick.display().addLabel(30, 10, "Start!");
brick.display().redraw();
script.wait(1000);
```

```
52
53  var v = 50; // velocity
54
55  var ex = 0;
56  var ey = 0;
57  var encLeftOld = 0;
58  var encRightOld = 0;
59  var encLeft = 0;
60  var encRight = 0;
61  var n = 0;
62
63  // вычисление абсолютного угла относительно начального положения
64  function angle() {
65    var sgn = 0;
66    var _direction = readYaw(); // mgrad
67    var dtDirection = _direction - directionOld;
68
69    sgn = directionOld == 0 ? 0 : directionOld / Math.abs(directionOld);
70    n += sgn * Math.floor(Math.abs(dtDirection / 320000));
71    direction = _direction + n * 360000;
72    directionOld = _direction;
73  }
74
75  // делаем прерывание основной программы с частотой 200Гц,
76  // чтобы посчитать абсолютный угол с гироскопа
77  var mtimer = script.timer(50);
78  mtimer.timeout.connect(angle);
79
80  //поворот на угол по гироскопу _angle - относительный угол на который необходимо повернуться
81  function turnDirection(_angle, _v) {
82    _angle = azimut + _angle;
83    azimut = _angle;
84    _angle = _angle * 1000; //toMGrad
85
86    eLeft.reset();
87    eRight.reset();
88
89    var _vel = _v == undefined & 40: _v; // скорость по умолчанию
90    var angleOfRotate = _angle - direction;
91    var sgn = angleOfRotate == 0 ? 0 : angleOfRotate / Math.abs(angleOfRotate);
92    mLeft(-_vel * sgn);
93    mRight(_vel * sgn);
94    eLeftOld = eLeft.read();
95    eRightOld = eRight.read();
96    target = 211;
97
98    while (Math.abs(eRight.read() - eLeft.read()) / 2 < target) {
99      if ((eLeft.read() - eLeftOld) == 0) && (eRight.read() - eRightOld == 0) {
100       _vel += 5;
101       mLeft(-_vel * sgn);
102       mRight(_vel * sgn);
103     }
104     eLeftOld = eLeft.read();
105     eRightOld = eRight.read();
106     script.wait(20);
107   }
108
109   brick.motor(M3).powerOff(500);
110   brick.motor(M4).powerOff(500);
111   script.wait(500);
```

```
112 }
113
114 // проезд в перед на количество ячеек _kceel со скоростью _v
115 // выравниваясь по гироскопу на угол azimut
116 function forward(_v, _kcell) {
117   print("azimut = " + azimut);
118   _alpha = azimut;
119   var _vel = _v == undefined ? 40 : _v; // скорость по умолчанию
120   var u = 0;
121   eLeft.reset();
122   eRight.reset();
123   var el = Math.abs(eLeft.readRawData());
124   while (Math.abs(eLeft.readRawData()) < (el + (_kcell * cellLength))) {
125     u = 1.5 * (_alpha - direction / 1000);
126     mLeft(_vel - u);
127     mRight(_vel + u);
128     script.wait(5);
129   }
130   brick.motor(M3).powerOff();
131   brick.motor(M4).powerOff();
132   script.wait(300);
133 }
134
135 min = function(a, b) {
136   return a < b ? a : b;
137 }
138 max = function(a, b) {
139   return a > b ? a : b;
140 }
141
142 //============================================================================
143 // массив для изображения
144 var pic = [];
145
146 var marker_size = 5;
147 var total_width = 320 / 2;
148 var total_height = 240 / 2;
149 var prob = 25 * 5 * 5;
150
151 function getColor(pic, x, y) {
152   return pic[y * total_width + x];
153 }
154
155 function squareAverage(pic, x, y, diam) {
156   var sum = 0;
157   var start_row = y * total_width;
158   var end_row = start_row + diam * total_width;
159
160   for (var index = start_row + x; index < end_row; index += total_width)
161     for (var j = 0; j < diam; j += 1)
162       sum += pic[index + j];
163
164   return sum;
165 }
166
167 // lu - left-up corner. Coordinates: (x, y)
168 // ld - left-down corner
169 // ru - right-up corner
170 // rd - right-down corner
171 function getCenterColor(pic, lu, ld, ru, rd, diam) {
```

```
  var x = (lu[0] + ld[0] + ru[0] + rd[0]) >> 2;
  var y = (lu[1] + ld[1] + ru[1] + rd[1]) >> 2;
  var color = squareAverage(pic, x - diam, y - diam, diam << 1);
  return color;
}

function findGridCorners(corners, marker_size) {
  var grid_corners = [];

  var vertical_lines = [];
  var upper_line_x1 = corners[0][0];
  var upper_line_y1 = corners[0][1];
  var upper_line_x2 = corners[2][0];
  var upper_line_y2 = corners[2][1];
  var down_line_x1 = corners[1][0];
  var down_line_y1 = corners[1][1];
  var down_line_x2 = corners[3][0];
  var down_line_y2 = corners[3][1];

  var mks = 1.0 / marker_size;
  var k_ux = (upper_line_x2 - upper_line_x1) * mks;
  var k_uy = (upper_line_y2 - upper_line_y1) * mks;
  var k_dx = (down_line_x2 - down_line_x1) * mks;
  var k_dy = (down_line_y2 - down_line_y1) * mks;

  for (var i = 0; i < marker_size + 1; i += 1) {

    var up_x = upper_line_x1 + k_ux * i;
    var up_y = upper_line_y1 + k_uy * i;

    var down_x = down_line_x1 + k_dx * i;
    var down_y = down_line_y1 + k_dy * i;

    var k_x = (down_x - up_x) * mks;
    var k_y = (down_y - up_y) * mks;

    for (j = 0; j <= marker_size; j += 1) {

      var point_x = up_x + k_x * j;
      var point_y = up_y + k_y * j;

      grid_corners.push([Math.floor(point_x), Math.floor(point_y)]);
    }
  }
  return grid_corners;
}

function detectCode(pic, grid_corners, diam) {
  var calculated_colors = []
  var markerSizePlusOne = marker_size + 1;
  var shiftedDiam = diam << 8;

  for (var i = 0; i < marker_size; i += 1) {
    for (var j = 0; j < marker_size; j += 1) {
      lu_index = i * (markerSizePlusOne) + j;
      ld_index = i * (markerSizePlusOne) + j + 1;
      ru_index = (i + 1) * (markerSizePlusOne) + j;
      rd_index = (i + 1) * (markerSizePlusOne) + j + 1;

      var lu = grid_corners[lu_index];
```

```
      var ld = grid_corners[ld_index];
      var ru = grid_corners[ru_index];
      var rd = grid_corners[rd_index];

      grid_color = getCenterColor(pic, lu, ld, ru, rd, diam);
      if (grid_color < shiftedDiam) {
        calculated_colors.push(0);
      } else {
        calculated_colors.push(1);
      }
    }
  }
  return calculated_colors;
}

function findULCorner(pic, diam) {
  var color = 1;
  for (var i = 0; i < total_height; i += 1) {
    for (var j = 0; j <= i; j += 1) {
      var x = j;
      var y = i - j;
      if (getColor(pic, x, y) == 0) {
        color = squareAverage(pic, x, y, diam);
        if (color < prob) {
          return [x, y];
        }
      }
    }
  }
}

function findDLCorner(pic, diam) {
  var color = 1;
  for (var i = 0; i < total_height; i += 1) {
    for (var j = 0; j <= i; j += 1) {
      var x = j;
      var y = total_height - (i - j);
      if (getColor(pic, x, y) == 0) {
        color = squareAverage(pic, x, y - diam + 1, diam);
        if (color < prob) {
          return [x, y];
        }
      }
    }
  }
}

function findURCorner(pic, diam) {
  for (var i = 0; i < total_height; i += 1) {
    for (var j = 0; j <= i; j += 1) {
      var x = total_width - j;
      var y = i - j;
      if (getColor(pic, x, y) == 0) {
        var color = squareAverage(pic, x - diam + 1, y, diam);
        if (color < prob) {
          return [x, y];
        }
      }
    }
  }
```

```
}

function findDRCorner(pic, diam) {
  for (var i = 0; i < total_height; i += 1) {
    for (var j = 0; j <= i; j += 1) {
      var x = total_width - j;
      var y = total_height - (i - j);
      if (getColor(pic, x, y) == 0) {
        var color = squareAverage(pic, x - diam + 1, y - diam + 1, diam);
        if (color < prob) {
          return [x, y];
        }
      }
    }
  }
}

function findCorners(pic, diam) {
  return [findULCorner(pic, diam), findDLCorner(pic, diam), findURCorner(pic,
    diam), findDRCorner(pic, diam)];
}

function threshold2(level, pic, height, width) {
  var length = pic.length;
  for (var i = 0; i < length; i += 1) {
    var color = pic[i];
    if (color < level) {
      pic[i] = 0;
    } else {
      pic[i] = 255;
    }
  }

  return pic;
}

// ------------------------------------------------------------------------------
var scale = 1;
var histogram = [];
var histSize = 256;

function calculateHistogram() {
  for (var i = 0; i < histSize; i += 1)
    histogram[i] = 0;

  var curPixelLine = 0;
  for (var i = 0; i < total_height; i += 1) {
    curPixelLine = i * total_width;
    for (var j = 0; j < total_width; j += 1)
      histogram[Math.floor(pic[curPixelLine + j])] += 1;
  }
}

// binarization using 2 elems in grayscale
var grayscale = "@#ao|-. ";
var numOfBins = grayscale.length;
var rangeBins = [];
var binCapacity = total_height * total_width / numOfBins;

function getRange() {
```

```
352   for (var i = 0; i < numOfBins; i += 1)
353     rangeBins[i] = 0;
354 
355   var curBin = 0;
356   var curSum = 0;
357   var i = 0;
358   var lastIndexBin = numOfBins - 1;
359 
360   for (;
361     (i < histSize) && (curBin < lastIndexBin); i += 1) {
362     var diff = binCapacity - curSum;
363 
364     if (Math.abs(diff) < Math.abs(diff - histogram[i])) {
365       curBin++;
366       curSum = 0;
367     }
368 
369     curSum += histogram[i];
370     rangeBins[curBin] = i;
371   }
372 
373   for (; curBin <= lastIndexBin; curBin += 1)
374     rangeBins[curBin] = histSize;
375 }
376 
377 var mapColorToLetter = [];
378 
379 function initMapColorToLetter() {
380   var curBin = 0;
381   for (var i = 0; i < histSize; i += 1) {
382     if (rangeBins[curBin] <= i) {
383       curBin += 1;
384     }
385     mapColorToLetter[i] = grayscale[curBin];
386   }
387 }
388 
389 // Возвращает значение ARTag
390 function getARTagValue() {
391   source_pic = getPhoto();
392 
393   // init pic, grayscale mode
394   for (var i = 0; i < total_height; i += 1) {
395     for (var j = 0; j < total_width; j += 1) {
396       var x = (j + i * scale * total_width) * scale;
397       var p = source_pic[x];
398       p = (((p & 0xff0000) >> 18) + ((p & 0xff00) >> 10) + ((p & 0xff) >> 2));
399       pic[j + i * total_width] = p;
400     }
401   }
402   calculateHistogram();
403   getRange();
404   initMapColorToLetter();
405 
406   var thresh = threshold2(rangeBins[1], pic, total_height, total_width);
407   var corners = findCorners(thresh, 9);
408   var grid_corners = findGridCorners(corners, marker_size)
409   var values = detectCode(thresh, grid_corners, 3);
410 
411   var ans = 0;
```

```
412     if (values[1][1] == 0)
413       ans = 8 * values[3][2] + 4 * values[2][3] + 2 * values[2][1] + values[1][2];
414     else if (values[1][3] == 0)
415       ans = 8 * values[2][1] + 4 * values[3][2] + 2 * values[1][2] + values[2][3];
416     else if (values[3][3] == 0)
417       ans = 8 * values[1][2] + 4 * values[2][1] + 2 * values[2][3] + values[3][2];
418     else if (values[3][1] == 0)
419       ans = 8 * values[2][3] + 4 * values[2][3] + 2 * values[3][2] + values[2][1];
420     else
421       print("Error: Incorrect ARTag");
422     return ans;
423   };
424   //=====================================================================
425
426   // Местонахождение робота
427   var positionOfRobot = 0;
428   var directionOfRobot = 0;
429
430   // карта
431   lab = [
432     [-1, 1, 8, -1],
433     [-1, 2, -1, 0],
434     [-1, 3, 10, 1],
435     [-1, 4, -1, 2],
436     [-1, 5, -1, 3],
437     [-1, -1, 13, 4],
438     [-1, -1, -1, -1],
439     [-1, -1, 15, -1],
440     [0, -1, 16, -1],
441     [-1, -1, -1, -1],
442     [2, -1, 18, -1],
443     [-1, -1, 19, -1],
444     [-1, -1, -1, -1],
445     [5, 14, 21, -1],
446     [-1, 15, -1, 13],
447     [7, -1, -1, 14],
448     [8, 17, -1, -1],
449     [-1, 18, -1, 16],
450     [10, -1, 26, 17],
451     [11, 20, 27, -1],
452     [-1, 21, -1, 19],
453     [13, -1, 29, 20],
454     [-1, -1, -1, -1],
455     [-1, -1, 31, -1],
456     [-1, -1, 32, -1],
457     [-1, -1, -1, -1],
458     [18, -1, 34, -1],
459     [19, -1, -1, -1],
460     [-1, -1, -1, -1],
461     [21, 30, 37, -1],
462     [-1, 31, -1, 29],
463     [23, -1, 39, 30],
464     [24, 33, 40, -1],
465     [-1, 34, -1, 32],
466     [26, 35, 42, 33],
467     [-1, 36, -1, 34],
468     [-1, 37, 44, 35],
469     [29, -1, 45, 36],
470     [-1, -1, -1, -1],
471     [31, -1, -1, -1],
```

```
    [32, -1, -1, -1],
    [-1, -1, -1, -1],
    [34, -1, 50, -1],
    [-1, -1, -1, -1],
    [36, 45, -1, -1],
    [37, 46, 53, 44],
    [-1, 47, -1, 45],
    [-1, -1, 55, 46],
    [-1, 49, 56, -1],
    [-1, 50, -1, 48],
    [42, 51, 58, 49],
    [-1, -1, 59, 50],
    [-1, -1, -1, -1],
    [45, -1, -1, -1],
    [-1, -1, -1, -1],
    [47, -1, 63, -1],
    [48, -1, -1, -1],
    [-1, -1, -1, -1],
    [50, 59, -1, -1],
    [51, 60, -1, 58],
    [-1, 61, -1, 59],
    [-1, 62, -1, 60],
    [-1, 63, -1, 61],
    [55, -1, -1, 62]
];

// double cycle for traveling in maze
cycle = [0, 2, 4, 13, 15, 21, 19, 29, 31, 37, 45, 46, 55, 62, 60, 50, 48, 34,
  32, 18, 16, 0, 2, 4, 13, 15, 21, 19, 29, 31, 37, 45, 46, 55, 62, 60, 50, 48,
  34, 32, 18, 16
];

var foundedDestroyedSectors = 0;
var destroyedSectors = 0;

// получаем маршрут перемещения до необходимой точки из текущей
// в нотации F, L, R
function getPath(finishSector) {
  from = [];
  for (var i = 0; i < 64; i++) {
    from[i] = [];
    for (var j = 0; j < 4; j++)
      from[i][j] = "N";

  }

  var queue = [];
  queue.push([positionOfRobot, directionOfRobot]);
  var finishDir = 0;
  while (queue.length > 0) {
    temp = queue.shift();
    currentSector = temp[0], currentDir = temp[1];
    if (currentSector == finishSector) {
      finishDir = currentDir;
      break;
    }
    // Вперед
    if (lab[currentSector][currentDir] > -1) {
      adjSector = lab[currentSector][currentDir];
      adjDir = currentDir;
```

```
532         if (from[adjSector][adjDir] == "N") {
533           from[adjSector][adjDir] = "F";
534           queue.push([adjSector, adjDir]);
535         }
536       }
537       // Направо
538       if (from[currentSector][(currentDir + 1) % 4] == "N") {
539         adjSector = currentSector;
540         adjDir = (currentDir + 1) % 4;
541         from[adjSector][adjDir] = "R";
542         queue.push([adjSector, adjDir]);
543       }
544       // Налево
545       if (from[currentSector][(currentDir + 3) % 4] == "N") {
546         adjSector = currentSector;
547         adjDir = (currentDir + 3) % 4;
548         from[adjSector][adjDir] = "L";
549         queue.push([adjSector, adjDir]);
550       }
551     }
552
553     path = "";
554     // Восстанавливаем путь
555     if (from[finishSector][finishDir] == "N")
556       print("No way");
557     else {
558       currentSector = finishSector;
559       currentDir = finishDir;
560       while (currentSector != positionOfRobot || currentDir != directionOfRobot) {
561         action = from[currentSector][currentDir];
562         if (action == "F") {
563           path = "F" + path;
564           currentSector = lab[currentSector][(currentDir + 2) % 4];
565         } else if (action == "R") {
566           path = "R" + path;
567           currentDir = (currentDir + 3) % 4;
568         } else if (action == "L") {
569           path = "L" + path;
570           currentDir = (currentDir + 1) % 4;
571         }
572       }
573     }
574     return path;
575   }
576
577   // Внести информацию о недоступности сектора
578   function isolateSector(sector) {
579     foundedDestroyedSectors++;
580     for (dir = 0; dir < 4; dir++)
581       if (lab[sector][dir] > -1) {
582         lab[lab[sector][dir]][(dir + 2) % 4] = -2;
583         lab[sector][dir] = -2;
584       }
585     calcAvailable();
586     print("countUnavailable=", countUnavailable);
587   }
588
589   // Проверка окружающих секторов
590   function checkAdjacentSectors() {
591     if (!(svFront() > 25) &&
```

```
592           lab[positionOfRobot][directionOfRobot] > -1)
593       isolateSector(lab[positionOfRobot][directionOfRobot]);
594     if (!(svLeft() > 25) &&
595           lab[positionOfRobot][(directionOfRobot + 3) % 4] > -1)
596       isolateSector(lab[positionOfRobot][(directionOfRobot + 3) % 4]);
597     if (!(svRigth() > 25) &&
598           lab[positionOfRobot][(directionOfRobot + 1) % 4] > - 1)
599       isolateSector(lab[positionOfRobot][(directionOfRobot + 1) % 4]);
600   }
601
602   // Перемещение по кратчайшему пути до finishSector
603   function follow_path(finishSector) {
604     path = getPath(finishSector);
605     while (positionOfRobot != finishSector && path != "") {
606       if (foundedDestroyedSectors < destroyedSectors)
607         checkAdjacentSectors();
608       for (var i = 0; i < path.length; i++) {
609         if (path[i] == "R") {
610           turnDirection(-90, v);
611           directionOfRobot = (directionOfRobot + 1) % 4;
612         } else if (path[i] == "L") {
613           turnDirection(90, v);
614           directionOfRobot = (directionOfRobot + 3) % 4;
615         } else if (path[i] == "F") {
616           if (lab[positionOfRobot][directionOfRobot] < 0) {
617             print("The path is blocked. No way!");
618             break;
619           }
620           forward(v, 1);
621           positionOfRobot = lab[positionOfRobot][directionOfRobot];
622         }
623         if (foundedDestroyedSectors < destroyedSectors)
624           checkAdjacentSectors();
625         else if (foundedDestroyedSectors == destroyedSectors)
626           break;
627       }
628       if (foundedDestroyedSectors == destroyedSectors) {
629         foundedDestroyedSectors++;
630         break;
631       }
632       path = getPath(finishSector);
633     }
634   }
635
636   // Поворот робота на заданное направление
637   function turnTo(targetDir) {
638     var dDir = (targetDir - directionOfRobot + 4) % 4;
639     if (dDir == 1) {
640       turnDirection(-90, v);
641     } else if (dDir == 3) {
642       turnDirection(90, v);
643     } else if (dDir == 2) {
644       turnDirection(-180, v);
645     }
646     directionOfRobot = targetDir;
647   }
648
649   // Проход по лабиринту
650   function travelThroughoutMaze() {
651     var firstSector = 0;
```

```
652   for (firstSector = 0; firstSector < cycle.length / 2; ++firstSector) {
653     if (positionOfRobot == cycle[firstSector] || lab[positionOfRobot][0] ==
654       cycle[firstSector] || lab[positionOfRobot][1] == cycle[firstSector] ||
655       lab[positionOfRobot][2] == cycle[firstSector] ||
656       lab[positionOfRobot][3] == cycle[firstSector])
657       break;
658   }
659   for (i = firstSector; i < firstSector + cycle.length / 2; i++) {
660     follow_path(cycle[i]);
661     if (foundedDestroyedSectors >= destroyedSectors) {
662       break;
663     }
664   }
665 }
666
667 function dfs(sector) {
668   used[sector] = true;
669   countAvailable++;
670   for (var dir = 0; dir < 4; ++dir) {
671     nextSector = lab[sector][dir];
672     if (nextSector > -1 && !used[nextSector])
673       dfs(nextSector);
674   }
675 }
676
677 // Вычисление доступных/недоступных секторов
678 function calcAvailable() {
679   used = [];
680   for (var i = 0; i < 64; i++)
681     used = [];
682   // Сколько доступных секторов
683   countAvailable = 0;
684   dfs(positionOfRobot);
685   // Сколько недоступных секторов
686   countUnavailable = 64 - countAvailable - 12;
687 }
688
689 var value1, value2;
690 // Считывание двух ARTag маркеров
691 function readARTag(dist) {
692   forward(-v, dist);
693   value1 = getARTagValue();
694
695   forward(v, dist);
696   value2 = getARTagValue();
697   print(value1 + " " + value2);
698 }
699
700 //============================================================================
701 //==================================M A I N ===================================
702 var main = function() {
703
704   var xStart = 7;
705   var yStart = 1;
706   directionOfRobot = 3;
707   var xARTag = 5;
708   var yARTag = 6;
709   // С какой стороны от сектора распредления решений располагается ARTag
710   var directionOfARTag = 3;
711
```

```
712     // Количество обрушенных секций
713     destroyedSectors = 2;
714     positionOfRobot = xStart + yStart * 8;
715 
716     travelThroughoutMaze();
717     follow_path(xARTag + yARTag * 8);
718     turnTo((directionOfARTag + 3) % 4);
719 
720     //detecting ARTag markers until it isn't successful
721     value1 = 0;
722     value2 = 0;
723     dist = 0.3;
724     while (value1 < 8 && value2 < 8 || value1 > 7 && value2 > 7) {
725       readARTag(dist);
726       dist += 0.05;
727     }
728 
729     // Координаты финиша
730     xFinish = 0;
731     yFinish = 0;
732     if (value1 < 8) {
733       xFinish = value1;
734       yFinish = value2 - 8;
735     } else {
736       xFinish = value2;
737       yFinish = value1 - 8;
738     }
739 
740     print(xFinish + " " + yFinish);
741     finishSector = xFinish + yFinish * 8;
742     follow_path(finishSector);
743     print(countUnavailable);
744     return;
745   }
```