HELESNAKE

Ресурсы на планете Шелезяка подходят к концу, и роботы были вынуждены перейти на питание упавшими частицами метеоритов. Через несколько дней ожидается большой метеоритный дождь, поэтому вам поручили написать программу для небольшой серии из трех роботов, позволяющую им в сумме собрать максимальное число метеоритных осколков.

Поверхность планеты представляет собой плоский круг, по которому раскидано 100 метеоритных осколков. Как роботы, так и осколки считаются точками. Когда робот подходит к осколку на расстояние 1.0 или ближе, он подбирает осколок, а с неба в произвольное место планеты падает новый (таким образом, количество осколков всегда равно 100).

Действия роботов разделены на такты. У вас всего есть 5000 тактов, чтобы собрать максимальное количество осколков.

СРЕДА ВЫПОЛНЕНИЯ

Роботы программируются на языке Lua (версии 5.1). Все три ваших робота управляется одной программой. Каждый такт все роботы принимают решения о том, куда им переместиться, затем перемещаются, затем подбирают осколки, если могут.

Ваша программа запускается один раз и должна определить функцию tick. Затем на каждой секунде будет вызываться глобальная функция tick.

Вам доступны все базовые функции стандартной библиотеки Lua, за исключением dofile, loadfile, pcall, xpcall, и все функции, начинающиеся с coroutine., table., string., math. (кроме math.random).

Дополнительно для связи с тестирующей системой вам доступны следующие функции:

- **`game.params()`** возвращает три числа: ваш номер игрока (от 1 до числа игроков на поле), ограничение по времени (всегда 5000, если только вы не изменили его локально опцией --time), и N — число роботов в вашем парке (всегда 3)
- **`game.world()`** возвращает состояние мира, одно число: текущий номер такта (от 0 до 5000; он равен 0 до первого вызова **`tick`**)
- **`game.status(i)`** возвращает три числа про вашего робота номер **`i`**: координату **`x`**, координату **`y`** и количество набранных очков
- **`game.move(i, vx, vy)`** задает вектор скорости перемещения вашего робота номер i (число от 1 до N); **`(vx, vy)`** задают вектор, который будет нормализован; вы можете использовать вектор (0, 0), чтобы оставаться на месте
- **`game.find(x, y, d, filter)`** возвращает роботов или еду на поле в порядке их удаления от заданной точки **`(x, y)`**, где

  `d` — максимальное расстояние до интересующих вас объектов;

  `filter` — строка, определяющая, какие объекты вас интересуют; эта строка может включать один или несколько из следующих символов:

  `"f"` — возвращать метеоритные осколки,

  `"r"` — возвращать роботов,

  `"m"` — возвращать только ваших роботов (имеет смысл, только если указана "r"),

  `"o"` — возвращать только чужих роботов (имеет смысл, только если указана "r"),

  `"f"` и **`"r"`** можно указывать вместе, а **`"m"`** и **`"o"`** взаимно исключающие. Таким образом, допустимые фильтры такие: **`"f"`**, **`"r"`**, **`"rm"`**, **`"ro"`**, **`"fr"`**, **`"frm"`**, **`"fro"`**.

  Функция возвращает список таблиц, у каждой из которых есть такие поля:

  `player` — номер игрока, или 0, если это еда,

  `x, y` — координаты,

  `dist` — расстояние до указанной вами точки.

- **game.count(x, y, d, filter)** возвращает количество объектов, которые вернул бы аналогичный вызов **game.find**.
- **log(...)** пишет заданные значения в лог игры,
- **random_int(a, b)** возвращает случайное целое число в закрытом диапазоне [**a; b**], так, чтобы результат был воспроизводим в зависимости от seed'a всего раунда,
- **random_real(a, b)** возвращает случайное вещественной число в полуоткрытом диапазоне [a; b), так, чтобы результат был воспроизводим в зависимости от seed'a всего раунда,
- debugger() останавливает выполнение программы и подключает отладчик, если прогонка запущена с ключом **-d** (**--debug**), иначе не делает ничего (в т.ч. во время тестирования на компьютерах жюри).

Пожалуйста, используйте для генерации случайных чисел **random_int** и **random_real**, это сделает результаты раунда повторяемые с помощью опции **--seed**. Внутри эти генераторы используют **std::mersenne_twister_engine (std::mt19937)** из C++.

Роботу дается 1 секунда на ход (на реальном туре ограничение будет гораздо жестче).

Память ограничена 10 МБ. Если вы генерируете много мусора, вы можете быстро выйти за пределы памяти, потому что Lua не запускает сборку мусора в тот момент, когда кончается память. Вы можете изменить параметры сборки мусора в Lua функцией **collectgarbage**.

ЗАПУСК РЕШЕНИЙ

Мы предоставляем вам два исполняемых файла, которые принимают одинаковые опции командной строки:

- **shelesnake-runner** запускает раунд игры в консольном режиме
- **shelesnake-vis** запускает игру в интерактивном графическом визуализаторе (нажмите **h**, чтобы получить список клавиш)

В командной строке необходимо передать пути к одному или нескольким решениям на Lua, которые будут соревноваться между собой. После имени файла можно через пробел поставить целое число — количество игроков, которые нужно запустить с этим решением. Например, если вы написали одно решение sol.lua и хотите запустить 60 его копий, нужно выполнить:

```
shelesnake-vis sol.lua 60
```

Если вы хотите сравнить good.lua с 59 копиями bad.lua:

```
shelesnake-vis good.lua bad.lua 59
```

В конце раунда в стандартный поток вывода печатается результат игры.

Вы можете захотеть задать фиксированный **seed** генератора случайных чисел, чтобы повторять одну и ту же ситуацию во время отладки:

```
shelesnake-vis sol.lua 60 --seed 123
```

В целях отладки и экспериментирования вы можете отключить контроль времени и памяти, передав опцию **-N (--non-strict).** Вы также можете захотеть изменить длительность игры опцией **--time.**

ОТЛАДКА РЕШЕНИЙ

Мы реализовали поддержку отладки с использованием бесплатной среды разработки ZeroBrane Studio. Для этого переменная среды ZBS_PATH должна содержать путь к ZeroBrane. (Во время тура путь уже будет задан правильно.)

Чтобы начать отладку:

Запустите ZeroBrane, выберите команду **Project | Start Debugger Server**.

Откройте в ZeroBrane свой lua-файл.

Выберите в ZeroBrane директорию, из которой вы запускаете решения. Для этого используйте команду

Project | Project Directory | Choose...

или

Project | Project Directory | Set From Current File.

При вызове **shelesnake-runner** или **shelesnake-vis** добавьте опцию **-d**, чтобы разрешить отладку.

Вызовите **debugger()** или нажмите клавишу d в визуализаторе, чтобы остановить исполнение и начать отладку.

После окончания сеанса отладки вам может потребоваться нажать кнопку «стоп» в ZeroBrane, чтобы студия была готова для нового сеанса отладки.

Вы можете передать **-D** вместо **-d**, чтобы остановиться в отладчике сразу после начала исполнения программы. Это даст возможность использовать breakpoint'ы, устанавливаемые в отладчике, потому что подключение к отладчику происходит только при первой остановке.

ПРИМЕР РЕШЕНИЯ

Жюри предлагает вам следующий базовый шаблон решения данной задачи. В нем, в частности, демонстрируется простой способ сохранения информации про каждого робота между тактами.

```
ME, Wmaxtime, N = game.params()
FOOD = 0
EPS = 1e-6

ROBOTS = {}
for Ri = 1, N do
    ROBOTS[Ri] = {}
end

function tick()
    Wtime = game.world()
    -- log("tick ", Wtime)

    for Ri = 1, N do
        R = ROBOTS[Ri]
        Rx, Ry, Rscore = game.status(Ri)
        dx, dy = move_for_robot(Ri, R, Rx, Ry)
        game.move(Ri, dx, dy)
    end
end

function move_for_robot(Ri, R, Rx, Ry)
    local apples = game.find(Rx, Ry, 5000, "f")

    local apple
    if R.idx then
        apple = apples[R.idx]
        if (apple.x ~= R.last_apple_x) or (apple.y ~= R.last_apple_y) then
            apple = nil
        end
    end

    if apple == nil then
        R.idx = random_int(1, 20)
        apple = apples[R.idx]
        R.last_apple_x = apple.x
        R.last_apple_y = apple.y
    end

    -- log("robot ", Ri, " (", Rx, "; ", Ry, ") moving towards food (", food[1].x,
"; ", food[1].y, ")")

    return apple.x - Rx, apple.y - Ry
end
```