Последние дни Шелезяки

Через 45 лет после экспедиции профессора Селезнева на планете Шелезяка создали первого робота-пылесоса, и уже к лету с вновь построенного конвейера каждый месяц сходили десятки таких роботов. Наконец-то появилась надежда расчистить тонны пыли, равномерно покрывающей всю планету, и вновь начать принимать регулярные рейсы в сияющем космопорте.

Однако из-за дефекта в системе обучения, вызванного крайне халатной культурой программирования у местного населения, в полночь 5 ноября роботы-пылесосы решили, что их главным врагом являются производители мусора — другие роботы этой планеты. Всего один час потребовался им, чтобы остановить всех роботов других моделей, взорвав их источник питания — металлическое ядро планеты.

Теперь роботы-пылесосы принялись догонять и убирать друг друга. Поскольку все они совершенно одинаковы, в схватке между двумя роботами всегда побеждает тот, который больше, причем он после победы присоединяет к себе все детали убитого робота и за счет этого увеличивается. Разумеется, растет при этом и масса, что заставляет робота двигаться медленнее.

В результате взрыва геометрия планеты изменилась: во-первых, она стала плоской, во-вторых, камни на краю начали обваливаться в космическую бездну, так что радиус планеты уменьшается (равномерно по тактам). Роботам приходится все теснее прижиматься к суперпрочному центру планеты.

Вы управляете серией из 10 роботов-пылесосов, и ваша задача — разработать стратегию для них.

Игра состоит из 5000 тактов, на каждом из которых вызывается ваша стратегия. В начале игры планета является кругом радиуса 2000.0, к концу радиус уменьшается до 1.0.

Все роботы-пылесосы являются окружностями и движутся с постоянной скоростью, определяемой их радиусом по формуле `speed = 0.5 + sqrt(1 / (r + 3.0))`. Движущий крепежный механизм робота находится ровно в центре; центр не может выйти за границы планеты (а при сужении планеты ее границы подталкивают роботов внутрь). В то же время, тело робота может «свисать» с границ планеты без каких-либо последствий.

На планете разбросан металлолом — мелкие куски старых разбившихся роботов и космических кораблей. Этот металлолом роботы могут поглощать, поэтому он также называется *едой* (*food*). Принцип распределения этих осколков не изменяется между раундами игры, и его можно увидеть в визуализаторе. Металлолом, оказавшийся за пределами планеты, исчезает.

Каждый такт происходит так:

1. Радиус планеты уменьшается.
2. Вызываются все стратегии, которые отдают команды своим роботам.
3. Все роботы передвигаются на величину своей скорости.

4. Координаты роботов, оказавшихся за границами планеты, изменяются так, чтобы они вернулись на игровое поле, а полярный угол оставался неизменным. Металлолом, оказавшийся за границами планеты, падает в бездну и исчезает.

5. Каждый робот засасывает металлолом и других роботов меньших радиусов, если круг этого робота и круг поглощаемого объекта пересекаются. Процесс засасывания повторяется, пока это возможно (например, если в результате засасывания робот увеличился и стал пересекаться еще с чем-то).

В результате засасывания детали разобранного робота или металлолом присоединяются к разобравшему их роботу. От этого площадь робота-пожирателя увеличивается на площадь засосанного объекта. Центр при этом остается неизменным.

У каждого робота есть небольшой бак с высокоэффективным горючим, получаемым из толстого слоя пыли на поверхности планеты. Этот бак изначально пуст и заполняется по мере движения робота каждые 50 тактов. Бак можно использовать, только если он полностью заполнен; это дает двукратное увеличение скорости на протяжении 20 тактов. Бак начинает заполняться заново сразу после начала использования.

Среда исполнения

Роботы программируются на языке Lua (версии 5.1). Все десять ваших роботов управляются одной программой. Программа запускается один раз и должна определить функцию `tick`, которая затем вызывается на каждом такте. Значения глобальных переменных сохраняются между тактами. Таким образом, наименьшее допустимое решение выглядит так:

```
function tick()
end
```

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 игрой вам доступны следующие функции:

- `game.params()` возвращает три числа: ваш номер игрока (от 1 до числа игроков на поле), количество тактов в игре (всегда 5000, если только вы не изменили его локально опцией –time), и `N` — число роботов в вашей серии (всегда 10).
- `game.world()` возвращает состояние мира, два числа: текущий номер такта (от 0 до 5000; он равен 0 до первого вызова tick) и текущий радиус мира (уменьшается от 2000 до 1).
- `game.move(i, vx, vy)` задает вектор скорости перемещения вашего робота номер i (число от 1 до N); имеет значение только направление заданного вектора; вы можете использовать вектор (0, 0), чтобы оставаться на месте.
- `game.status(i)` возвращает состояние робота вашего робота номер i (число от 1 до N) — 5 чисел: alive (0 или 1, определяющее, жив ли робот), радиус, координаты x, y, скорость.

- `game.find(x, y, r, d, filter)` возвращает роботов или еду на поле в порядке их удаления от заданной точки (x, y), где

`r` — радиус объекта в точке (x, y); он учитывается при подсчете расстояний, а также используется для фильтрации при задании фильтров "e" или "p";

`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)
  - "`e`" — возвращать врагов (enemies) — роботов, которые больше вас (имеет смысл, только если указана r)
  - "`p`" — возвращать добычу (prey) — роботов, которые меньше вас (имеет смысл, только если указана r)

"`f`" и "`r`" можно указывать вместе; "`m`" и "`o`" взаимно исключающие; "`e`" и "`p`" взаимно исключающие. (Если не указано ни "`m`", ни "`o`", то функция не фильтрует по игроку. Если не указано ни "`e`", ни "`p`", то функция не фильтрует по радиусу.)

Функция возвращает количество найденных объектов (роботов и еды). Чтобы получить сами объекты, используйте game.get.

Время выполнения этой функции тем больше, чем больше заданный радиус поиска `d`.

- `game.get(i)` — отдает информацию об i-том объекте из результатов последнего вызова find. Возвращает 7 чисел: x, y, radius, номер игрока (от 1 до числа игроков или 0 для еды), порядковый номер объекта у игрока (от 1 до N, или, для еды, от 1 до количества еды на поле), расстояние до данного объекта, скорость данного объекта.

- `game.turbo(i)` — включает функцию турбо у вашего робота номер i (число от 1 до N), если он накопил достаточно высокоэффективного горючего; в противном случае не делает ничего. Турбо-режим продлится 20 тактов с момента включения.

- `game.turbostatus(i)` — возвращает состояние турбо-режима вашего робота номер i (число от 1 до N) — два числа: количество тактов, оставшихся до того, как будет доступен турбо-режим (0, если он уже доступен), и количество оставшихся ускоренных тактов при включенном турбо-режиме (0, если в данный момент турбо-режим выключен).

- `log(...)` пишет переданные значения в текстовый 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()` возвращает случайное вещественной число в полуоткрытом диапазоне [0; 1), 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` или `-D`, иначе не делает ничего (в т.ч. во время тестирования на компьютерах жюри).
- `watchquota`, `watch` — см. ниже раздел про отладку.

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

Ограничения

Время работы вашей стратегии на одном такте ограничено 500 миллисекундами. При превышении этого времени стратегия удаляется из тура, а робот продолжает двигаться в последнем заданном направлении.

Кроме того, есть мягкое ограничение на суммарное время работы стратегии на всех тактах: изначально это 500 миллисекунд, к которым каждый такт добавляются 2 миллисекунды. Если после очередного такта ваша стратегия суммарно использовала больше текущего мягкого ограничения, она будет пропускать дальнейшие такты, пока ограничение не вырастет достаточно, чтобы покрыть использованное время.

Память ограничена 10 МБ. При превышении этого ограничения ваша стратегия будет удалена из тура, а робот продолжит двигаться в последнем заданном направлении. Заметитье, что, если вы генерируете много мусора, вы можете быстро выйти за пределы памяти. Вы можете захотеть изменить параметры сборки мусора в Lua функцией collectgarbage.

Общее количество логов, выдаваемых функцией `log`, ограничено 1 МБ за всю игру. При превышении этого размера дальнейшие вызовы `log` ничего не делают.

Вы можете использовать опцию `-N` при запуске решений локально, чтобы снять все эти ограничения.

Отправка решений и система оценки

Решение в виде одного файла с исходным кодом на языке Lua можно посылать в тур «Первая номинация» в системе тестирования nsuts. Сразу после отправки ваше решение будет передано на тестирующую машину, заменяя более старое ваше решение. При этом вердикт в системе тестирования изменится с «Queued» на «Testing».

На тестирующей машине в бесконечном цикле выполняются раунды игры. От каждой команды в игре всегда участвует последнее посланное в nsuts решение. Результаты раунда кладутся на общий сервер по адресу `http://parallels.nsu.ru/shelezyaka/main/`. Все раунды нумеруются по порядку, номер раунда вставляется в имена выкладываемых файлов. Подробнее о том, как скачать истории игр, описано далее.

Система оценки такова. Назовем *территорией* игрока на данном такте суммарную площадь всех его роботов. *Очками* игрока за одну конкретную игру является максимальная его территория по всем тактам игры. Игроки ранжируются по набранным очкам: чем больше очков, тем лучше место. Распределение очков и мест по ним за каждый раунд доступно участникам в файлах `scoresXXXXX.txt`.

На финальное тестирование будет взято *последнее посланное в nsuts решение* от каждой команды. Игра будет запущена много раз, и итоговое место в первой номинации будет определено по *среднему значению очков* каждой команды. Результаты раундов, произошедших во время тура, *никак НЕ учитываются* в финальном тестировании.

Архив материалов

Из системы тестирования вы можете скачать архив материалов. Он включает в себя:

1. `Runner.exe`: консольный игровой сервер *как в системе тестирования*.
2. `VisRunner.exe`: игровой сервер с визуализатором для просмотра.
3. `default_player.lua`: простое решение задачи с демонстрацией всех возможностей игрового API.
4. `download.bat`: скрипт для скачивания историй и логов игр с сервера (можно просмотреть их сразу).
5. `teams.txt`: список идентификаторов команд-участников (нужен для настройки `download.bat`).
6. `docs`: директория с документацией по языку Lua.
7. `libs`: директория с набором простых библиотек на языке Lua (можно копировать в решение).

Все материалы рекомендуется распаковать в `D:\tools`. Чтобы материалы работали, нужно положить их в одну директорию и запускать из нее.

Запуск решений

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

- `Runner` запускает раунд игры в консольном режиме

- `VisRunner` запускает игру в интерактивном графическом визуализаторе (нажмите h, чтобы получить описание клавиш управления)

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, вы написали одно решение sol.lua и хотите запустить его в консоли:

```
VisRunner sol.lua
```

Если вы хотите запустить 60 его копий, нужно выполнить:

```
VisRunner sol.lua 60
```

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

```
VisRunner good.lua bad.lua 59
```

Можно запустить игру на всех файлах с расширением `.lua` из заданной директории:

```
VisRunner testsols\
```

Например, можно запустить все решения из текущей директории (она обозначается точкой):

```
VisRunner .
```

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

```
VisRunner sol.lua 60 -—seed 123
```

В целях отладки и экспериментирования вы можете отключить контроль жесткого ограничения времени и памяти, передав опцию -N (–non-strict). Мягкое ограничение времени при этом все равно действует.

```
VisRunner -N sol.lua bad.lua null.lua
```

Чтобы подсвечивать в визуализаторе нужного вам игрока по номеру, можно указать ключ *-m*:

```
VisRunner -m 2 sol.lua bad.lua null.lua
```

Вы также можете захотеть изменить длительность игры опцией –time.

Консольный вариант игры `Runner` в основном аналогичен варианту `VisRunner` с визуализатором.

В конце работы в рабочей директории создаются следующие файлы:

- `log.txt`: текстовый лог игры. Содержит сообщения, которые вы выдали при помощи команды log из решения. Особое внимание следует уделить критическим сообщениям (имеют префикс WARN, FATAL). Они включают ошибки компиляции, ошибки времени исполнения, превышение жесткого ограничения по времени, превышение ограничения по памяти.
- `run.txt`: история игры, пригодная для просмотра. Посмотреть ее можно, запустив `VisRunner.exe -r` или `VisRunner -R run.txt`. Подробнее описано далее.
- `scores.txt`: результаты прогона решений в текстовом виде. Этот файл создается только если игра корректно завершается. Здесь видно, сколько очков набрало каждое решение.

Обратите внимание, что критические сообщения всегда дублируются в стандартный поток вывода (на консоль) в том числе ошибки компиляции, ошибки времени исполнения, превышение жесткого ограничения по времени и ограничения по памяти.

Просмотр истории игры

История игры сохраняется в файле `run.txt`, а отладочные сообщения — в файле `log.txt` (самые важные из них также дублируются на экран).

Просмотреть историю из заданного файла можно при помощи визуализатора:

```
VisRunner -R run00001.txt
```

Кроме того, для запуска файла с фиксированным именем `run.txt` есть удобная короткая форма этой опции:

```
VisRunner -r
```

Скачать истории игр из системы тестирования можно в браузере из директории:

```
http://parallels.nsu.ru/shelezyaka/main/
```

Здесь лежат файлы `runXXXXX.txt`, `logXXXXX.txt` и `scoresXXXXX.txt`, у которых вместо `XXXXX` стоит номер запуска игры в системе. Можно скачивать и смотреть любые из них.

Для просмотра скачанного файла истории рекомендуется указать ключом *–me* идентификатор вашей команды в системе тестирования НГУ:

```
VisRunner -R run00123.txt --me 21865
```

Идентификатор команды можно найти в файле `teams.txt` по адресу `http://parallels.nsu.ru/shelezyaka/main/teams.txt`.

Для удобства скачивания и просмотра историй и логов можно использовать скрипт `download.bat`. Чтобы визуализатор подсвечивал вашу команду, рекомендуется прописать в него идентификатор вашей команды. Нужно изменить его в этой строке файла `download.bat`:

```
set TEAMID=21865
```

Скачать и проиграть в визуализаторе историю последней игры можно:

```
download.bat last play
```

Просто скачать последнюю игру можно без команды `play`:

```
download.bat last
```

Можно указать номер игры для скачивания вручную (`play` можно добавить для проигрывания):

```
download.bat 127
```

Отладка в ZeroBrane

Можно отлаживать решение с использованием среды разработки ZeroBrane Studio.

Чтобы начать отладку:

1. Запустите ZeroBrane, выберите команду «Project | Start Debugger Server».

2. Откройте в ZeroBrane свой lua-файл.

3. 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».

4. При вызове Runner или VisRunner добавьте опцию `-d`, чтобы разрешить отладку.

5. 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.

Замечание: отладка доступна только для первого из решений, перечисленных в командной строке.

Отладка: watch

Функция watch добавляет просматриваемое значение в визуализаторе. В системе тестирования эта команда не делает ничего.

Формат запуска функции watch такой.

Чтобы добавить глобальный watch:

```
watch("g", подпись, выражение)
```

Чтобы добавить watch около своего робота с заданным номером:

```
watch("r", подпись, выражение, номер робота)
```

Чтобы добавить watch около заданной точки:

```
watch("p", подпись, выражение, x, y)
```

При этом:

- `подпись` — текст в начале
- `выражение` — строка, значение которой нужно вычислить в lua-программе
- `номер робота` — порядковый номер робота от 1 до N
- `x`, `y` — координаты точки (вы можете задать каждую в виде фиксированного числа или в виде строки, которая будет регулярно вычисляться аналогично *выражению*)

В любой момент хранится не более, чем некоторое фиксированное количество watch-ей. Это ограничение можно устанавливать командой watchquota:

```
watchquota("g", количество)
watchquota("r", количество, номер робота)
watchquota("p", количество)
```

Пример решения

Жюри предлагает базовый шаблон решения данной задачи в файле `default_player.lua`. В нем, в частности, демонстрируется простой способ сохранения информации про каждого робота между тактами.