Pragmatic Perl 10

pragmaticperl.com

Выпуск 10. Декабрь 2013

Другие выпуски и форматы журнала всегда можно загрузить с http://pragmaticperl.com. С вопросами и предложениями пишите на editor@pragmaticperl.com.

Комментарии к каждой статье есть в html-версии. Подписаться на новые выпуски можно по ссылке pragmaticperl.com/subscribe.

Авторы статей: Сергей Романов, Сергей Можайский, Владимир Леттиев

Корректор: Андрей Шитов

Выпускающий редактор: Вячеслав Тихановский (vti)

Ревизия: 2014-11-29 16:16

© «Pragmatic Perl»

Оглавление

1	От редактора	1
2	Воркшоп Saint Perl 5	3
3	Что такое cpanfile?	6
4	Обзор CPAN за ноябрь 2013 г.	22
5	Интервью с Marc Lehmann. Часть 2	31
6	Морской бой на Perl — решение Perl Golf 09	54
7	Perl Golf	67

1 От редактора

21 декабря в Санкт-Петербурге пройдет пятый воркшоп, посвященный языку Perl и его сообществу. Приезжайте пообщаться, послушать доклады и просто хорошо провести время! Анонс мероприятия читайте в этом номере.

С начала декабря и до католического рождества некоторые популярные Perl-проекты запускают так называемый Advent Calender (Рождественский календарь), где каждый день публикуется новая статья. В этом году ведутся следующие календари: Perl, Perl 6, Futures.

В предыдущие годы подобные календари были у Catalyst, Dancer, Plack.

Сложно предположить, когда именно выйдет следующий номер, поэтому всех читателей поздравляем с наступающими праздниками и желаем интересных проектов в новом году!

Мы продолжаем искать авторов для следующих номеров. Если у вас есть идеи или желание помочь, пожалуйста, свяжитесь с нами.

Приятного чтения.

■ *Вячеслав Тихановский*

2 Воркшоп Saint Perl 5

18 декабря 2013 года исполнится 26 лет с момента выхода первой версии языка программирования *Perl*. А уже через два дня, 21 декабря, в Санкт-Петербурге пройдёт ежегодный воркшоп *Saint Perl*. В этот раз *Saint Perl* пройдёт уже в пятый раз, отмечая таким образом своеобразный юбилей. (Первые три воркшопа нам удалось провести прямо 18 декабря!)

В этом году площадку для воркшопа любезно согласилась предоставить компания JetBrains — её основной центр разработок и исследований находится практически в самом центре Санкт-Петербурга, на Васильевском острове, в 10 минутах ходьбы от Университетской набережной.

В течение дня участники прослушают ряд докладов (прием докладов все ещё открыт!), получат возможность задать вопросы выступающим специалистам и пообщаться на тему современного состояния дел в разработке на одном из самых заслуженных скриптовых языков современности. Традиционная секция блиц-докладов завершит официальную часть мероприятия.

Организаторы выражают благодарность всем, кто помогает подготовиться к мероприятию и оказывает медийную поддержку.

Мы также рады сотрудничеству со спонсорами, которые обеспечат возможность проведения конференции на ещё более высоком уровне.

Сайт конференции: event.yapcrussia.org/saintpe

Электронная почта: saintperl_orgs@yapcrussia.

Увидимся в Петербурге!

■ *Сергей Романов*

3 Что такое cpanfile?

От создателя `cpanminus` и `Plack`, революционная инновация в процессе подготовки Perl-приложения для распространения — `cpanfile`. Начните использование сегодня, и установка ваших модулей будет мягкой и шелковистой даже из git.

cpanfile

`cpanfile` — это название файла, а также формата, который описывает CPAN-зависимости для Perl-приложений. Идея `cpanfile` пришла из мира Ruby, где для описания зависимостей используется `Gemfile`. На сам формат сильно повлиял DSL, используемый в системе сборки `Module::Install`, и, кстати, `cpanfile`

обратно-совместим с ним. Кроме того, описание зависимостей использует терминологию спецификации META при описании зависимостей: те же типы и фазы зависимостей.

Рассмотрим пример:

```
# требуется Perl 5.8.5 или старше
requires 'perl', 5.008_005

# рекомендуется Foo::Bar версии 1.0 или старше
# но меньше версии 2.00
recommends 'Foo::Bar', '>= 1.0, < 2.00';

# Зависимости фазы тестирования
on test => sub {
    # Test::More версии 0.88 или старше
    requires 'Test::More', '0.88';
```

```
};
```

Зачем нужен cpanfile?

Все существующие системы сборки, такие как `ExtUtils::MakeMaker`, `Module::Build`, `Module::Install` и другие имеют средства для описания зависимостей.

```
use ExtUtils::MakeMaker;
WriteMakefile(
    NAME => "Foo::Bar",
    PREREQ_PM => {
        'JSON::XS' => '3.00'
    }
);
```

Как правило, утилиты для установки CPAN-модулей используют файл `META.yml`, который автоматически формируется средства-

ми сборки и содержит информацию о зависимостях. Таким образом, типичный CPAN-модуль содержит всё необходимое для корректной сборки и установки в систему. Зачем нужно что-то ещё?

Прежде всего, `cpanfile` предлагает универсальный формат описания зависимостей, не перегруженный избыточным многословием, который может использоваться различными утилитами, начиная от инсталляторов, заканчивая статическими анализаторами или сервисами непрерывной интеграции.

На данный момент уже созданы плагины для многих систем сборок, которые позволяют использовать cpanfile для извлечения списка зависимостей. Например, для `ExtUtils::MakeMaker` есть модуль ExtUtils::MakeMaker::CPANfile, который

позволяет переписать указанный выше пример так:

```
# Makefile.PL
use ExtUtils::MakeMaker::CPANfile
    ;
WriteMakefile(
    NAME => "Foo::Bar",
);

# cpanfile
requires 'JSON::XS', '3.00
```

Для других систем сборки соответственно есть Module::Install::CPANfile, Dist::Zilla::Plugin::Prereqs::FromCPANfile, Module::Build::Pluggable::CPANfile. Таким образом, можно использовать `cpanfile` совместно с существующими системами сборки и избежать дублирования информации о зависимостях.

Кроме того, существует несколько ситуаций, когда применение `cpanfile` имеет явные преимущества. Рассмотрим такие примеры.

Веб-приложение или скрипт

Разрабатывается приложение, которое использует модули CPAN, но при этом нет задачи по выкладыванию самого приложения на CPAN. В этом случае для нормальной установки приложения требуется удобный способ описания зависимостей без необходимости создания полноценного CPAN-дистрибутива с `Makefile.PL` (выбора системы сборки и формальное следование всем её требованиям).

В такой ситуации после создания `cpanfile` все необходимые CPAN-модули можно

установить с помощью команд:

```
$ carton install
```

или

```
$ cpanm --installdeps .
```

Установка модуля из git-репозитория

Всё больше авторов используют git в качестве системы контроля версий и github как публичный хостинг проектов. В этом отношении всё большее значение приобретает возможность брать и использовать код непосредственно из git (или любой другой VCS).

Как правило, при разработке модуля в git-репозитории отсутствуют автоматически

генерируемые файлы. В случае использования `Dist::Zilla` это означает отсутствие `Makefile.PL` или `Build.PL` вообще, что требует от пользователя необходимость установки `dzil` и множества модулей этой среды разработки, которые не имеют никакого отношения к работе модуля, всё только ради возможности выяснить, какие он требует зависимости. В случае `Module::Install` потребуется знать, какие нужны модули в каталоге `inc` только для бутстрапа самого `Makefile.PL`.

В подобных ситуациях наличие `cpanfile` позволяет быстро и просто установить необходимые зависимости с помощью `cpanm` или `carton`, что существенно повышает доступность кода для использования непосредственно из git-репозитория.

Фиксация зависимостей

Существующие средства описания зависимостей фиксируют саму зависимость и минимальную необходимую версию. Таким образом, не исключена возможность установки более новой версии зависимого модуля, с которой работа приложения не тестировалась. С помощью `cpanfile` существует возможность точно указать требуемые версии модулей или диапазоны таких версий, а также явно указывать версии, которые использовать нельзя. Например,

```
# зафиксировать версию
requires 'JSON::XS', '== 3.01'

# диапазон версий с исключениями
requires 'Plack',
  '> 1.0000, != 1.0020, !=
     1.0021, <= 1.0030'
```

Это позволяет получать предсказуемые результаты установки и работоспособность программы.

Формат cpanfile

Рассмотрим пример cpanfile:

```
requires 'Catalyst', '5.8000';
requires 'Catalyst::View::JSON',
    '>= 0.30, < 0.40';

recommends 'JSON::XS', '2.0';
conflicts 'JSON', '< 1.0';

on 'test' => sub {
  requires 'Test::More', '>=
     0.96, < 2.0';
  recommends 'Test::TCP', '1.12';
};

```

```
12 on 'develop' => sub {
13   recommends 'Devel::NYTProf';
14 };
15
16 feature 'sqlite', 'SQLite support
     ' => sub {
17   recommends 'DBD::SQLite';
18 };
```

Ключевые слова `requires`, `recommends`, `conflicts`, а также `suggests` описывают зависимости модуля соответственно как обязательные, рекомендуемые, конфликтующие и опциональные. После ключевого слова идёт название модуля, затем через запятую указывается строка с диапазоном версий, формат которой соответствует спецификации `CPAN::Meta::Spec` Version Range:

- `undef`, `''`, `0` — любая версия

- `'1.00'` или `'>= 1.00'` — минимальная версия (1.00 или старше)
- `'> 1.00'` — любая версия старше 1.00
- `'<= 1.00'` — версия 1.00 или младше
- `'< 1.00'` — строго младше версии 1.00
- `'!= 1.00'` — любая версия, кроме 1.00
- `'== 1.00'` — только версия 1.00

С помощью ключевого слова `on` можно определять, к какой именно фазе относятся указанные зависимости:

1. `develop` — зависимости, необходимые для разработки модуля
2. `configure` — зависимости этапа конфигурации, т.е. необходимые для успешного запуска `perl Makefile.PL`

3. `build` — зависимости этапа сборки (`make`)
4. `test` — зависимости этапа тестирования (`make test`)
5. `runtime` — основные зависимости приложения, необходимые для выполнения приложения

Для совместимости с `Module::Install` `requires`-зависимости, указанные в перечисленных фазах, можно указывать с помощью соответствующих ключевых слов: `author_requires`, `configure_requires`, `build_requires`, `test_requires`.

После определения фазы идёт тело функции с описанием зависимостей фазы.

Ключевое слово `feature` описывает зависимости той или иной опциональной функциональности приложения, которая

может быть подключена или отключена. Вслед за ключевым словом идёт идентификатор функционала, через запятую следует строка его описания, которая может быть опущена. Последним параметром идёт функция, описывающая зависимости функционала.

Идеи развития cpanfile

Одно из интересных направлений развития cpanfile — это поддержка не только CPAN-модулей, но и произвольных репозиториев кода.

Например, на данный момент `cpanminus` поддерживает установку модулей непосредственно из git, например:

```
cpanm git://github.com/plack/
```

```
    Plack.git@1.0000 # тег
cpanm git://github.com/plack/
    Plack.git@devel  # бранч
```

Кажется разумным, чтобы и `cpanfile` поддерживал подобную возможность, например:

```
requires 'Plack', '1.001', at =>
    'git://github.com/tokuhirom/
    Plack.git'
```

Также открыт вопрос с тестированием git-репозиториев. На сегодняшний момент многие используют сервис Travis CI. Для этого в репозиторий помещают файл `.travis.yml` для описаний зависимостей и способа их установки. К сожалению, данный метод специфичен для данного сервиса. Кажется интересным идея о добавлении ключевого слова или команды в `cpanfile`, которая бы задавала способ

подготовки кода к стандартному виду CPAN-дистрибутива.

■ *Владимир Леттиев*

4 Обзор CPAN за ноябрь 2013 г.

Рубрика с обзором интересных новинок CPAN за прошедший месяц.

Статистика

- Новых дистрибутивов — 214
- Новых выпусков — 1004

Новые модули

- Encode::Detect::Upload Модуль, который пытается определить кодировку отправленных данных веб-клиента, исходя из его ip-адреса (геолокация), переменных окру-

жения HTTP_ACCEPT_LANGUAGE и HTTP_USER_AGENT. Несмотря на ориентированность на CGI, модуль можно использовать и в другом окружении.

- AnyEvent::Delay Ещё один модуль для управления выполнением асинхронных задач. Модуль является аналогом Mojo::IOLoop::Delay, созданным на основе Ev (логичным было бы название Ev::Delay).

- AnyEvent::Delay::Simple Модуль имеет то же назначение, что и AnyEvent::Delay, но имет более простой интерфейс и базируется на чистом AnyEvent.

- HTML::CallJS Модуль, который позволяет безопасно вставлять в создаваемый html-документ javascript-код со

сложными структурами данных, которые автоматически преобразуются в JSON:

```
call_js('foo', {x => 1});
# => <script class="call_js"
      type="text/javascript">
    foo({"x":1})</script>
```

- AnyEvent::Promises Ещё одна реализация спецификации Promises, требующая для работы запущенной петли событий AnyEvent. В отличие от модуля Promises AnyEvent::Promises более полно следует спецификации Promises.

- WL Данный модуль является Perl-обвязкой к протоколу Wayland — замена протокола системы X Window.

- Ariba Реализация PSGI веб-сервера на основе Starman с поддержкой прото-

кола SPDY.

- App::Fetchware Приложение является пакетными менеджером для управления сборкой и установкой программ из исходных кодов.

- Try::Tiny::Retry Небольшое расширение к Try::Tiny, которое вводит ключевое слово retry, которое в отличие от try пытается выполнить код повторно несколько раз в случае ошибки, количество повтров и время повторений может настраиваться.

- Pistachio Модуль позволяет конвертировать текст с исходным кодом в html-фрагмент с подсветкой синтаксиса и нумерацией строк. Пока поддерживаются только язык Perl и стиль оформления Github. Из недостатков можно отметить использование встроенных

стилей, что существенно увеличивает объём создаваемого html-кода.

Обновлённые модули

- Coro 6.33 В новом релизе `Coro` исправлена ошибка при работе модуля в Perl 5.18, которая могла приводить к разыменованию нулевого указателя и краху процесса.
- Mouse 2.0.0 Вышел новый мажорный релиз Mouse, который обусловлен появлением несовместимого изменения: раньше при подключении роли к классу конфликтующие методы заменяли существующие, теперь же конфликтующие методы роли игнорируются.

- Devel::MAT 0.10 Обновлён модуль для анализа использования памяти Perl-приложениями, исправлены проблемы с работой на 64-битных системах и множество других улучшений. Модуль также содержит набор утилит, в том числе графический обозреватель всех символов процесса.
- Padre 1.00 Наконец-то. Выпущен первый мажорный релиз Padre Perl IDE за более чем пятилетний период развития, который содержит небольшие исправления ошибок.
- BerkeleyDB 0.54 Новый релиз BerkeleyDB устраняет ошибку с утечкой памяти при использовании механизма блокировки CDS.
- IO::Socket::SSL 1.960 В ноябре вышло несколько новых версий модуля

IO::Socket::SSL. Начиная с версии 1.956 появилось множество изменений в поведении модуля по умолчанию. В частности, усилен список используемых методов шифрования в пользу более безопасных, исключены MD5, а также методы анононимной идетификации. Это может привести к отказу в соединении со старыми менее безопасными реализациями. Добавлена поддержка TLSv11 и TLSv12 как протоколов согласования. Также была переработана схема проверки имён хостов, появилась поддержка wildcards в секции CN для протоколов SMTP, IMAP, POP3 и других. Кроме того wildcards теперь всегда должен совпадать хотя бы с одним символом (т.е. * теперь совпадает по действию с + в терминах регулярных выражений) и не может

применяться для IDNA-имён.

- Amon2 6.00 Вышел новый мажорный релиз веб-фреймворка Amon2. В новой версии произошёл переход на новую библиотеку для поддержки сессий `HTTP::Session2`.

- lib::xi 1.02 lib::xi — это небольшой модуль, который позволяет автоматически загрузить и установить с помощью cpanm отсутствующие модули непосредственно во время исполнения программы. В новой версии исправлен вывод предупреждений на некоторых системах.

```
$ perl -Mlib::xi script.pl
```

- Plack 1.0030 В новой версии суперклея для веб-фреймворков и веб-серверов исправлены некоторые ошибки и выполнена оптимизация кода. `HTTP::`

`Server::PSGI` больше не завершает аварийно работу, если заголовки или тело http-ответа содержат «широкие» символы.

- Readonly 1.04 Старый, преданный забвению `Readonly` обрёл нового мейнтейнера *Sanko Robinson*. Обновился файл `TODO`, в котором предложена «дорожная карта» дальнейшего развития модуля.

■ *Владимир Леттиев*

5 Интервью с Marc Lehmann. Часть 2

Marc Lehmann — автор AnyEvent, Coro, common::sense, JSON::XS и многих других популярных модулей на CPAN. Продолжение интервью из предыдущего номера.

Где сейчас работаешь? Perl — основной язык?

Я работаю в nethype GmbH в Германии, в компании, которую я основал вместе со своим другом, когда мы вместе учились.

К счастью, Perl — это наш основной язык, который мы используем для всего, начиная от распределенного трейдинга, высокопроизводительных DHCP- или radius-серверов, и заканчивая обычными SMTP-пауками, веб-приложениями или

даже онлайн-играми (например, Deliantra). Также повезло с тем, что мы можем часто публиковать свои модули, написанные по работе, как свободный софт.

Несмотря на то, что C или C++ мы далеко не откладываем, Perl — прекрасный язык не только для написания высокоуровневой логики, но и из-за XS всегда можно получить доступ к C для реализации частей, требующих высокой скорости.

Нравится ли посещать Perl-конференции?

Конечно! В основном из-за возможности пообщаться с другими Perl-чудаками (ну, и нормальными посетителями :).

Однако, заставить себя куда-то поехать это совершенно другая история — я очень ленивый, поэтому сейчас я посещаю гораздо

меньше конференций, чем раньше. Все это планирование, перелет или переезд (и подготовка выступления)... слишком много усилий (также я стараюсь не ездить в США, из-за сложных политических причин).

Если я мог себя заставить, то никогда не жалел, но меня нужно сильно уговаривать.

Что думаешь о будущем Perl?

К 2050 году я и еще парочка людей все еще используем Perl, зарабатывая большие деньги на поддержке cobolˆWPerl-программ, от которых из-за борьбы с глобальным потеплением зависит большинство населения планеты. Я становлюсь богатым, остальной мир же голодает и умирает до тех пор, пока электричество не пропадет и мои деньги не обесценятся, и тогда приходит всему трындец.

Простите, кажется, я приплел сюда свое будущее.

В реальности, я думаю, что Perl 6 продолжит свою медленную и необходимую смерть, давая Perl больше пространства для существования и, возможно, роста.

Не думаю, что какие-либо улучшения (если это улучшения) в ядре Perl сильно повлияют на будущее языка — Perl определенно достаточно хорош, также как и CPAN.

Но я все-таки считаю, что Perl уже не “моден”. Это и еще тот факт, что Perl уже не один в своей категории, сильно уменьшит важность языка, потому что меньше людей будут учить Perl или относиться к нему как к “основному языку”.

Для меня это не плохо и вполне естесствен-

но. Я думаю, что сила Perl не в его инновационности, а в заслуженном доверии и надежности. Это не очень хорошо для роста, но Perl еще будет полезным, когда другие более модные языки будут забыты.

Однако, я не руковожу процессом разработки Perl, поэтому мои предположения могут быть ошибочными и мои желания могут не разделяться теми людьми, которые формируют будущее Perl.

Лично я бы хотел, чтобы приоритет был у обратной совместимости и сохранении CPAN в рабочем состоянии, без постоянного ломания модулей.

Для меня это гораздо важнее реализации новых возможностей в perl-интерпретаторе — perl настолько гибок, что новые возможности могут быть реализованы с помощью

модулей, без необходимости что либо ломать, и это лучший путь для инноваций.

Поэтому, я думаю, что Perl не умрет, а сохранит свое месте среди популярных языков, и, возможно, сможет расти, сохраняя написанные тонны кода рабочими.

Стоит ли сейчас советовать молодым людям изучать Perl?

Конечно — это великолепный и очень полезный язык. Некоторые могут поспорить, что Java лучше для Денежно-Ориентированного Программирования, а программистов C++ чаще ищут работодатели, но кроме немного сложного процесса поиска работы из-за своей экзотичности, нет ничего странного, когда Perl это первый или второй язык программирования, да и просто веселее писать на Perl, чем на

Java, что очень полезно для здоровья.

Поэтому, если кто-то хочет начать с ruby, python или “похожего” языка, то посоветовать Perl никак не повредит. Давать “молодым” людям больше возможностей выбора между языками позволяет им выбрать наиболее удобный для них самих, что очень ценно.

Представьте только, что если бы я не попробовал VI потому что мне сказали, что он старый и немодный, я был бы вечно несчастлив с GNU Emacs. Ужас.

Есть, правда, небольшая проблема: иногда меня спрашивают, как выучить Perl, и я обычно здесь теряюсь, так как сам выучил этот язык, читая документацию и исходный код, что вероятно не самый простой способ. Если смотреть на книги, то многие

из них не ориентированы на новичков, например в “Learning Perl” издательства O’Reilly четко сказано, что книга не научит вас программировать. Да и многие другие книги, на которые я смотрел, учат Perl в качестве второго языка (или вообще не очень хорошие). Но нашел одно исключение — это “Beginning Perl” от Simon Cozens (она также в сводобном доступе!).

Я думаю, что сложно рекомендовать Perl в качестве первого языка программирования, когда большинство книг относятся к нему как ко второму, поэтому найдите несколько книг, прежде чем советовать учить Perl.

Но у самого языка нет никаких проблем.

Вопросы от читателей

Почему до сих пор пользуешься CVS?

Короткий ответ: потому что работает.

Длинный ответ:

CVS до сих пор сносно справляется со всем, что мне требуется. Git, например, требует больше команд для выполнений тех же checkout/update/commit-действий в CVS, и со множеством сложностей при работе в небольшой команде (будучи больше ориентированной на распределенную разработку!). SVN в действительности еще хуже, чем CVS (те же самые команды, те же ограничения, но длиннее URL и эта странная модель “копирования” от которой пухнет голова). Если вдруг CVS перестанет мне походить, я, возможно, посмотрю на Mercurial, эта система видится мне не такой сумасшедшей как git.

Я совсем не религиозен по отношению к

CVS (но удивительно, как много людей религиозны по отношению к git и постоянно достают меня, чтобы я перешел на эту систему) — мне ее достаточно. Зачем что-то менять, если оно вам подходит: сэкономленное время я могу потратить где-нибудь еще.

Кроме того, современные идентификаторы комитов и такие утилиты как cvsps позволяют CVS быть на том же уровне, что и другие системы контроля версий, исправляя большинство ограничений, поэтому разница не настолько большая, как была раньше.

Ты широко известен за высказывание своего мнения в жесткой манере. Это происходит намеренно?

Да и нет — обычно у меня есть мнение насчет вещей или концепций, но не о людях,

поэтому я не очень понимаю, почему мне стоит сдерживаться — я не могу обидеть чувства вещи или концепции. Я часто могу обосновать свою точку зрения (даже если я неправ, я стараюсь в разумных пределах пытаться быть правым, вместо того, чтобы навыдумывать безосновательного бреда). Это все помогает мне иметь и высказывать устойчивое мнение и редко бывать неправым. Также это помогает сдерживать себя в темах, о которых ты мало что знаешь.

У меня есть впечатление, что я не сильно-то и известен по этому поводу — несколько людей, поднявших эту тему, сами узнали от кого-то, то есть не видели самостоятельно никаких доказательств. По факту, единственными людьми, кто публично высказывался по этой теме, были рассерженные фанаты, патчи которых я не принял…

Это подтверждается моим опытом в других “не-Perl сообществах”, где я совсем не известен таким поведением (у меня есть несколько популярных библиотек и программ, написаных не на Perl...).

Считаешь себя частью Perl-сообщества?

Это зависит от того, что понимать под “Perl-сообществом”.

На первом Perl-воркшопе в Германии я был потрясен, когда увидел такое разнообразие участников: менеджеры, гики, любители; у них не было ничего общего между собой, кроме Perl. Было ли это Perl-сообществом? Не знаю, но мне кажется, что нет.

Я выкладываю модули на CPAN и считаю себя частью этого “сообщества”. Я не считаю себя частью движения “modern perl”

(если таковое даже существует).

Поэтому я не знаю, каким должно быть “Perl-сообщество”, и я подозреваю, что нет единого сообщества, а, скорее всего, несколько разных.

Одно я могу сказать точно: большинство моих проектов не имеют вокруг себя “сообщества”. Я разрабатываю софт для себя, и это часто означает, что я не буду добавлять новые возможности, которые сам не буду использовать (или поддерживать). По этой причине я разрабатываю таким образом, чтобы другие могли расширять функционал — это позволяет мне отклонять запросы на внесение изменений.

Изменилась ли ситуация с rt.cpan.org?

Все стало намного хуже. Были некоторые

косметические изменения (например, уведомления на некоторых страницах), но основной эффект в том, что rt.cpan.org еще сложнее не использовать. В основе лежит та же проблема, что была вначале: этот сервис навязывается людям, и его нельзя отключить (последняя проверка: начало 2013).

Были попытки на стороне rt.cpan.org скрыть эту проблему, но никаких попыток для улучшения ситуации. Это похоже на игру, в которую я не хочу играть. Это сильно раздражает, похоже на принудительную подписку на рассылку microsoft или что-то в этом роде, только нельзя это игнорировать как спам, потому что есть другие люди, которые также являются жертвами этой системы.

Я хочу, чтобы была простая кнопка для

отключения rt.cpan.org от моих модулей, или возможность заменить его на пересылку почты. Но есть те, кто в этом не заинтересован, и поэтому это невозможно.

Насколько стабильна обвязка OpenCL для Perl? Какие преимущества в использовании этой библиотеки из Perl по сравнению с C/C++?

За последние полтора года у меня не было необходимости в новом релизе, и эта библиотека продолжает надежно работать. Предполагаю, что много предстоит работы, когда NVidia будет наконец поддерживать OpenCL 1.2 (или даже 2.0), и я смогу улучшить поддержку этой версии (у меня до сих пор есть только один драйвер для тестирования версии 1.1).

И преимущества использования OpenCL из

Perl, в отличие от C/C++, в том, что ее можно использовать из Perl, дурилка :->

Почему AnyEvent по-особенному обрабатывает IO::Async?

Короткий ответ: потому что другие бэкэнды не работают с IO::Async.

Длинный ответ: IO::Async использует свой собственный мультиплексор (вместо того, что быть фреймворком, который находится над ним). Для каждого несовместимого обработчика событий должен быть свой бэкэнд — также как и для EV или Event, должен быть и для IO::Async.

Если бы IO::Async использовал другие обработчики событий (например, через AnyEvent, или напрямую через EV), тогда такого бэкэнда не было бы нужно.

AnyEvent использовал бы тот же бэкэнд, что и IO::Async, и они бы мирно сосуществовали.

(Для любителей технических подробностей: в обычной ситуации было бы достаточно интерфейса к IO::Async::Loop, который бы реализовывал обработчик событий IO::Async, но этот модуль — редкий пример библиотеки, которая не поддерживает разделение ресурсов между несколькими пользователями, поэтому, чтобы работать с IO::Async, AnyEvent, к сожалению, приходится использовать сам IO::Async (и все равно требуются дополнительные настройки — приходится сообщать AnyEvent, какой экземпляр стоит использовать)).

Кроме этого нет никаких особенных обработок IO::Async, модуль обрабатыва-

ется специфично, но в обычном режиме AnyEvent.

Есть один неоднозначный участок кода, который отключает себя, когда работает с модулем IO::Async::Loop::AnyEvent. Этот модуль не нужен, чтобы работать с IO::Async, также он не является частью IO::Async, скорее всего, именно из-за этого модуля и возник сам вопрос.

История этого модуля в том, что AnyEvent сидит сверху IO::Async как обработчике событий, но этот модуль пытается перевернуть все с ног на голову и быть сверху AnyEvent. Это вызывает бесконечную рекурсию и ведет к другим очевидным или неуловимым проблемам, и, в конечном счете, приводит к баг-репортам в моем почтовом ящике.

Я написал автору IO::Async об этой проблеме и объяснил, как IO::Async может правильно использовать AnyEvent, но так и не получил ответа (позже автор публично признал, что проигнорировал мое письмо, так ему было все равно).

Поэтому вместо того, чтобы в дальнейшем создавать редко воспроизводимые и тяжело отлаживаемые ошибки, AnyEvent не работает, когда загружен этот модуль. Во-первых, я не хочу иметь дело с этими фиктивными ошибками, и, во-вторых, я хочу защитить моих пользователей от незаметной порчи данных или бесконечной отладки.

Самая неприятная и болезненная деталь в этой истории состоит в том, что автор POE использовал данный факт для начала клеветнической кампании против меня,

обманывая людей, которым не свойственно самостоятельно искать доказательства — ответственный за выпуск perl 5 даже попросил CPAN удалить мои модули (и позже извинился за то, что сам не разобрался). Я скажу четко и недвусмысленно: AnyEvent *не* обрабатывает и *никогда* не обрабатывал IO::Async как-то иначе по отношению к другим бэкэндам, вне случаев, когда это обуславливалось разницей в API.

Подытожим: несмотря на костыльность, AnyEvent продолжает поддерживать IO::Async насколько это возможно: два модуля, использующие IO::Async, не могут прозрачно сосуществовать без определенных предосторожностей, что приходится реализовавывать несчастному пользователю.

До тех пор, пока вы используете AnyEvent::Imp

AnyEvent будет работать.

Какой вид “Спасибо” предпочитаешь? Литры или бутылки?

У меня есть проблема со всеми видами “Спасибо”, чем бы это ни было. Я это делал не для вас, поэтому мне сложно принимать благодарности (наверное, запущенный синдром Аспергера).

Если вам хочется поблагодарить меня за модуль (или за что угодно, в общем-то), не говорите мне, скажите другим об этом модуле — если модуль был каким-то образом полезен вам, расскажите и другим о нем. И наоборот, если вам не нравится мой код (не путайте, если вам не нравлюсь я лично), критикуйте и не советуйте.

Конечно, мне нравится слышать, что люди

используют мои разработки (когда я выпустил JSON::XS 2.0, посыпалось много писем по типу “зачем ты изменил API такого широко используемого модуля”, когда на самом деле я и не думал, что им кто-то пользовался, так как никто мне об этом не говорил; может мне стоит добавить несколько багов, чтобы получить фидбэк?), так что черкните мне пару строчек, что именно вам пригодилось (можете сказать спасибо, если это действительно необходимо, но помните, что мне проще иметь дело к объективными вещами, чем с чувствами, поэтому не злитесь, если я проигнорирую часть, где вы меня благодарите — я просто не знаю, что сказать).

Но я гарантирую, что буду продолжать выкладывать свои модули, будут ли меня благодарить или нет. Хотя, если какие-то вещи полезны другим, то их я буду обновлять по-

чаще.

■ *Вячеслав Тихановский*

6 Морской бой на Perl — решение Perl Golf 09

Задача Perl Golf из 9 номера оказалась одновременно и проще, и сложнее предыдущей. Проще — потому что алгоритм был прост и понятен (ведь все играли в Морской бой хоть раз), а сложнее — потому что одной только оптимизации по размеру было недостаточно.

Итак, цель — поразить хотя бы одну палубу линкора (4-клеточного корабля) на заданном поле. Забегая вперед, скажу, что если бы условием задачи было бы поразить все палубы линкора, эта история сложилась бы совсем по-другому.

За каждое лишнее попадание в каждом тесткейсе дается пенальти в 10 символов,

затем по итогам всех тестов оно усредняется и прибавляется к длине скрипта. Тестовый скрипт генерирует несколько случайных полей с разной плотностью имеющихся попаданий, поэтому пенальти между запусками может отличаться в пределах 10-20 баллов — именно поэтому я дальше в статье указываю его приблизительное значение.

В качестве примера к заданию прилагался скрипт, который просто заменял все пробелы символами попадания — при длине 17 символов он дает пенальти 510, итого 527 баллов. Нужно написать что-то получше :)

Первый вариант решения я написал сравнительно быстро, взяв код для поворота поля из своего решения предыдущего гольфа. Пришлось, конечно, повозиться с регулярным выражением — использовать

выражение [^X] для отсеивания стоящих рядом кораблей я догадался далеко не сразу.

```
#!perl -n0
sub r {
    # поворот поля на 90 градусов
    my @e;
    $e["@-" % 11] .= $& while /[^
       E]/gs; # оставляем в
       повернутом поле

    @e
}
sub x {
    s/
```

```
11                                  # возле
                                       корабля
                                        не
                                       должно
                                        быть

                                       попадани
                                       :
12      (?<=[^X]{6}.{5}[^X]) # ни
           перед ним,
13      [ @]{4,}
14      (?=[^X].{5}[^X]{6})  # ни
           после него
15      /
16      '@'x length $&       #
           заменяем найденный "
           корабль" из 4 и более
17                                  #
                                       символо
                                        на
                                       соответ

                                       количес
```

```
                                    попадани
    /egsx
}
$_ = "E"x11 . $_ . "E"x20; # прицепляем лишние символы, чтобы в начале и конце
                                    # поля
                                        сраба
                                        prema
                                         и
                                        postma
s/\n/E/g;
x;                                  # делаем проход в поисках горизонтальных кораблей
$_ = "E"x11 . join ("E", r) . "E"x20; # поворачиваем поле на 90 градусов
```

```
25 x;

    # и ищем вертикальные корабли
26 $_ = join "E", r;              #
    поворачиваем поле "как было"
27 s/E/\n/g;
28 print "$_\n";                  #
    выводим результат
```

Длина 274 символа, пенальти порядка 170 — итог 444 балла.

Конечно, можно обойтись без замены переносов строки буквой “E”, что я сразу же и сделал. Процедуру `r` можно научить сразу “собирать” поле обратно, все равно нам нужно это делать каждый раз после поворота поля. Ну и ключ `-p` вместо `print` `$_` — замечательный хак, позволяющий сэкономить несколько символов.

Немного подчистим код, и вот у нас уже 173 символа с тем же пенальти. Итого 343 балла:

```
#!perl -p0
sub r {
    my @e;
    $e["@-"%11] .= $& while /[^\n]/gs;
    join "\n", @e
}
sub x {
    s/(?<=[^X]{6}.{5}[^X])[ @]{4,}(?=[^X].{5}[^X]{6})/'@' x length $&/egs
}
$z = "\n" x 11;
$_ = $z . $_ . $z;
x;
$_ = $z . r . $z;
x;
$_ = r
```

Нетрудно заметить, что в конце скрипта

у нас снова есть повторяющиеся операции. Логично будет действия процедур `r` и `x` объединить в одну, тогда размер скрипта станет еще меньше. Кроме того, чтобы уменьшить величину пенальти, мы будем стрелять только в первый из четырех квадратов возможного расположения линкора.

Получился код длиной 136 символов с пенальти около 65 — в сумме 201 балл:

```
#!perl -p0
sub r {
    $_ = $z . $_ . $z;
    s/(?<=[^X]{6}.{5}[^X]) (?=[^X].{5}[^X]{6})/@/gs;
    my @e;
    $e["@-"%11] .= $& while /[^\n]/g;
    $_ = join "\n", @e
}
$z = "\n" x 22;
```

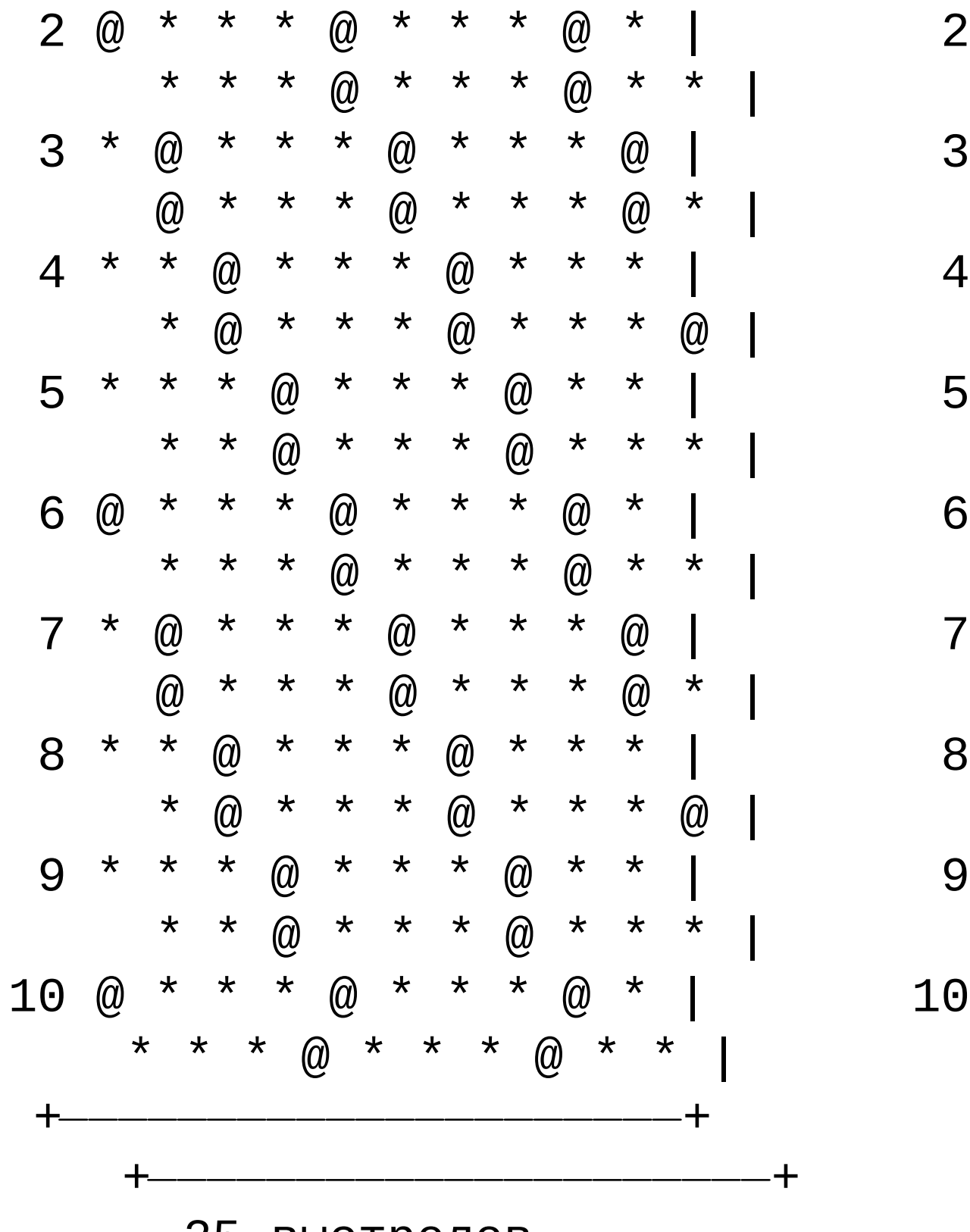
```
r;
r
```

Достаточно неплохой результат, и на нем вполне можно было бы остановиться.

Однако в один из вечеров, пока я ехал с работы, мне в голову пришла мысль о совершенно другом способе решения этой задачи. И этот способ — брутфорс :) Ведь не обязательно вычислять место нахождения линкора — чтобы гарантированно попасть хоть в одну его палубу, достаточно просто прострелять все поле по определенному алгоритму.

Есть несколько способов это сделать:

```
  +—а—б—в—г—д—е—ж—з—и—к—+
     +—а—б—в—г—д—е—ж—з—и—к—+
1 1 * * * @ * * * @ * * |         1
      * * @ * * * @ * * * |
```


2 @ * * * @ * * * @ * | 2
* * * @ * * * @ * * |
3 * @ * * * @ * * * @ | 3
@ * * * @ * * * @ * |
4 * * @ * * * @ * * * | 4
* @ * * * @ * * * @ |
5 * * * @ * * * @ * * | 5
* * @ * * * @ * * * |
6 @ * * * @ * * * @ * | 6
* * * @ * * * @ * * |
7 * @ * * * @ * * * @ | 7
@ * * * @ * * * @ * |
8 * * @ * * * @ * * * | 8
* @ * * * @ * * * @ |
9 * * * @ * * * @ * * | 9
* * @ * * * @ * * * |
10 @ * * * @ * * * @ * | 10
* * * @ * * * @ * * |
+——————————————————————+
+————————————————————————+
25 выстрелов
24
выстрела

Таким образом, достаточно поместить значки выстрела в заданные позиции при условии, что в них находится пробел.

В качестве эксперимента я попробовал считерить и просто сгенерировать пустое поле с попаданиями в нужных местах с помощью коротенького кода:

```
#!perl -p
$_='    @'x25
```

Но как и следовало ожидать, этот трюк не прокатил — скрипт получает пенальти более 2300, поскольку в тестах проверяется разница между оригинальным полем и тем, которое выдал проверяемый скрипт.

Ладно, попробуем первый вариант, он попроще:

```
#!perl -0p
s/(...)(.)/$2eq' '?"$1@":$&/egs
```

Всего 35 (!) символов с пенальти порядка 130. И по сумме баллов (165) эта версия выигрывает у “интеллектуального” варианта выше.

Конечно, если не стрелять в квадраты, рядом с которыми есть кусок подбитого корабля, пенальти будет куда меньше. Я попытался реализовать этот алгоритм, но увеличение длины кода не полностью компенсировалось уменьшением пенальти и приводило лишь к ухудшению итогового результата. Поэтому я бросил эту затею.

Для сравнения я реализовал второй вариант простреливания поля — код получился на 5 символов длиннее:

```
#!perl -0p
s/(..)(.)(.)/$2eq' '?"$1\@$3":$&/
    egs
```

Что интересно, на достаточно большом количестве полей эта версия дает чуть меньшее пенальти и в общем зачете выигрывает у предыдущего варианта с перевесом в 1 балл :) Видимо, потому, что она делает на один выстрел меньше.

Вот такая история о победе грубой силы над интеллектом. Спасибо за внимание!

■ *Сергей Можайский*

7 Perl Golf

Perl Golf — это соревнование по поиску Perl кода наименьшего размера (меньше всего символов), который решает заданную задачу. В этом выпуске будет сделан обзор решений предыдущего задания — «Морской бой», торжественно оглашено имя победителя и предложена новая головоломка.

«Морской бой»

Решить это задание было не очень сложно, поскольку даже элементарное заполнение карты символами @ выполняло задачу. Интереснее было бы найти решение, которое бы выполняло эту задачу более оптимально, учитывая границы подбитых кораблей и исключая места, в которых линкор поме-

ститься не мог. Именно для этих целей и была создана система штрафных балов, которая бы отсеяла решения в лоб.

На конкурс были представлены два решения, причём последнее подоспело за несколько минут до окончания срока приёма.

Автором первого решения является Сергей Можайский technix. Его первоначальный вариант предусматривал некоторый интеллектуальный анализ карты, но всё-таки в окончательном решении был выбран шаблонный обстрел полей, потому что, несмотря на пенальти, размер скрипта был небольшим и давал за счёт этого суммарный выигрыш.

```
#!perl -0p
s/(..)(.)(.)/$2eq' '?"$1\@$3":$&/
    egs
```

Скрипт обходит карту по четыре символа, и если третий символ совпадает с пробелом, замещает его символом @, таким образом в худшем случае на карте будут находиться 24 обстрелянные точки.

Иван Крылов aitap, автор второго решения, предложил следующий вариант (отформатированная версия):

```
#!perl -p0
sub z : lvalue {
    substr $_, $a + pos() - 1, 1
}
while (/X/g) {
    for $a ( -12 .. -10, -1, 1,
       10 .. 12 ) {
        z && z =~ s/ /*/
    }
}
s/(   ) /$1@/g;
{
    s/(( .{10}){3}) /$1@/sgc
```

```
13          && redo
14 }
```

Цикл `while` обходит карту в поиске полей с подбитой палубой. Как только такое поле найдено, запускается цикл, который обходит поля, находящиеся вокруг данного поля, функция `z` возвращает символ в данной позиции и если позиция существует, то производится замена пробела на символ `*`, таким образом на карте маскируются позиции, где линкор располагаться не может.

На следующем шаге, регулярное выражение ищет четыре пробельных поля подряд и замещает в последнем поле пробел на @. Следующий блок проводит такую же операцию, но для всех вертикальных полей.

К сожалению, в реализации содержится ошибка, связанная с тем, что `pos()` возвращает `undef`, каждый раз после того, как происходит успешное замещение пробела в `s/ /*/`. В итоге оказываются замещены позиции по углам карты, что выявляется при тестирование на большом количестве тестовых карт.

Тем не менее, идея отличная, реализация очень интересная. Очевидно, что ошибка возникла из-за спешки с отправкой решения и оно не было достаточно протестировано.

Финальное тестирование было проведено на 500 случайных тестовых картах, для которых выполнялся случайный обстрел в 60%, 50%, 40% и 30% картах (т.е. в общей сложности 2000 проверок для решения), что давало небольшую статистическую

погрешность результата.

```
example.pl: length= 17, penalty
    =509
technix.pl: length= 40, penalty
    =121
```

По итогам тестирования решение Сергея получало наименьшее значение — 161. Таким образом Сергей вновь становится победителем. Поздравляем Сергея с победой и выражаем благодарность Ивану за участие.

Как обычно, покажу и мой собственный вариант решения. Его размер составил 119 байт. Отформатированный вариант выглядит так

```
#!/usr/bin/perl -p0
while ($/) {
    $/  = s/ (.{9,11})?X/o$1X/s;
    $/ += s/X(.{9,11})?\K /o/s
```

```
}
s/ {3}\K /@/g;
while ($") {
    $" = s/( .{10}){3}\K /@/s
}
s/o/ /g
```

Алгоритм таков. Изначально карта сохраняется в переменной `$_`. Первый цикл `while` выполняет два регулярных выражения с заменой. Чтобы понять их смысл, обратимся к иллюстрации:

```
[1] [1] [1]
[1] [X] [2]
[2] [2] [2]
```

В центре квадрата из девяти полей находится палуба поражённого корабля. По правилам игры это значит, что поля вокруг этого квадрата не могут содержать палубы другого корабля. Первое регулярное выражение

заменяет все поля, помеченные цифрой `1`, символом `o`, если там находился символ пробела. Второе регулярное выражение замещает, соответственно, поля, помеченные цифрой `2`. Т.о. мы закрыли все поля, где не может быть линкора в принципе.

Следующим шагом, регулярное выражение ищет четыре пробельных поля подряд и замещает в последнем поле пробел на `@`.

Дальнейший цикл `while` проводит такую же операцию, но для всех вертикальных полей.

Ну и чтобы вернуть карту в тот вид, в котором оно было изначально — удаляем символы `o`, которые были расставлены на первом шаге.

По результатам тестирования такой алго-

ритм давал очень низкое пенальти ~ 34, т.е. в среднем всего 3-4 лишних символа на карте.

«Искусственный интеллект»

В то время как Google тестирует автомобили, которые управляются роботом, предлагаю не отставать и реализовать искусственный интеллект, который бы управлял Змейкой.

По условиям игры, необходимо управлять змейкой, которая двигается по плоскости и ищет еду. Движение змеи остановить невозможно, после поглощения еды змея увеличивается в длине. Змея не должна натыкаться на препятствия: границы экрана, а также саму себя, в противном случае игра

завершается.

В репозитории golf-10 реализован простейший вариант консольной игры с использованием библиотеки Curses. В простейшем варианте змейкой можно поуправлять вручную (snake.pl), а можно поиграть и против роботов (snake-fight.pl), но в этом случае надо реализовать хотя бы одного такого робота.

Как сделать своего робота? Для этого необходимо создать файл скрипта в каталоге `script` с именем `your_github_login.pl`. Ваш робот будет запущен в отдельном процессе и на `STDIN` получит строку с шестью полями, разделёнными пробелом, со следующими данными:

1. содержимое поля над головой змеи (`0` — пусто, # — препятствие, @ — ябло-

ко(пища))
2. содержимое поля справа от головы змеи
3. содержимое поля снизу от головы змеи
4. содержимое поля слева от змеи
5. смещение по оси Y до яблока
6. смещение по оси X до яблока

Например:

```
0 0 # 0 -10 2
```

На `STDOUT` должны будут быть переданы значения смещения по координатам `Y` и `X`, относительно текущего положения головы, например:

```
-1 0
```

Смещение по оси Y на 1 вниз, нет смещения по оси X. Необходимо учитывать, что змейка не может двигаться по диагонали, поэтому только одна координата может иметь ненулевое смещение. В нашем случае координаты точки `0,0` находятся в левом верхнем углу, ось Y растёт вниз, в ось X вправо.

Цель игры — написать не просто самый короткий вариант реализации управления, но и самый результативный в плане количества поглощённых яблок. И, как обычно, победителя определит `make test`.

Проверяться решения будут на последней стабильной версии Perl, т.е. 5.18.1 (или 5.18.2, если она выйдет в декабре). Приём решений закончится как только куранты начнут бить 12 ударов и начнётся Новый 2014 год. Поднимая бокал шампанского,

можете загадать, чтобы ваш вариант выиграл в конкурсе, ведь вдруг потом вашим роботом заинтересуются в Google.

■ *Владимир Леттиев*