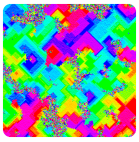
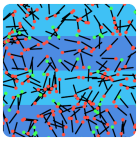
Оглавление



Введение 7

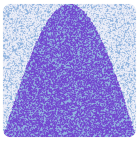
Естественные вычисления (7) Среда моделирования NetLogo (7) Структура книги (9)



ГЛАВА 1

Задача Бюффона 11

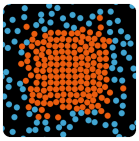
Методы Монте-Карло (11) Задача Бюффона (12) Построение модели (13) Упражнения (19)



ГЛАВА 2

Интегрирование методом Монте-Карло 23

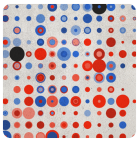
Площади и интегралы (23) Метод Монте-Карло (24) Геометрический подход (25) Построение модели (26) Упражнения (29)



ГЛАВА 3

Броуновское движение 33

Открытие явления (33) Случайные блуждания (34) Построение модели (35) Упражнения (41)

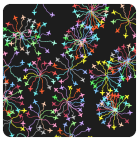


ГЛАВА 4

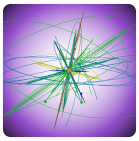
Задача о разорении игрока . . . 45

Постановка задачи (45) Марковские цепи (47) Построение модели (48) Упражнения (52)

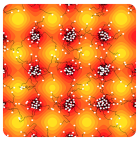
ГЛАВА 5
Метод имитации отжига 57
Отжиг металлов (57) Алгоритм Метрополиса (58) Метод имитации отжига (59) Построение модели (60) Упражнения (65)



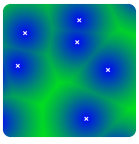
ГЛАВА 6
Модель Рейнолдса 69
Роевой интеллект (69) Модель Рейнолдса (70) Построение модели (71) Упражнения (75)



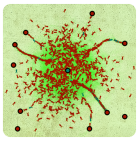
ГЛАВА 7
Метод роя частиц 79
Роевая оптимизация (79) Метод роя частиц (80) Построение модели (81) Упражнения (86)



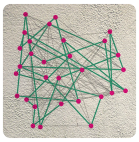
ГЛАВА 8
Алгоритм бактериального поиска 91
Хемотаксис бактерий (91) Алгоритм бактериального поиска (92) Роевое поведение бактерий (93) Построение модели (94) Упражнения (98)



ГЛАВА 9
Пчелиный алгоритм 101
Поведение пчел в природе (101) Алгоритм пчелиного поиска (102) Построение модели (103) Упражнения (107)

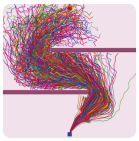


ГЛАВА 10
Модель муравьиной колонии 111
Коллективное поведение муравьев (111) Стигмергия (111) Эксперимент с двумя мостами (112) Построение модели (113) Упражнения (118)

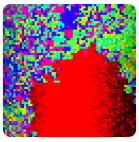


ГЛАВА 11
Муравьиные алгоритмы 121
Основная идея (121) Задача коммивояжера (122) Построение модели (123) Упражнения (130)

ГЛАВА 12
Роевая робототехника 133
Принципы организации (133) Сортировка предметов (134) Построение модели (135) Упражнения (140)

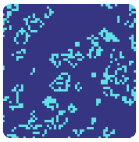


ГЛАВА 13
Генетические алгоритмы . . . 143
Понятие генетического алгоритма (143) Генетические операторы (144) Построение модели (145) Упражнения (151)



ГЛАВА 14
Клеточные генетические алгоритмы 155
Проблема потери разнообразия (155) Островная модель (156) Клеточная модель (156) Построение модели (158) Упражнения (163)

ГЛАВА 15
Элементарные клеточные автоматы 167
Клеточные автоматы (167) Элементарные автоматы (168) Классификация элементарных автоматов (169) Построение модели (170) Упражнения (173)

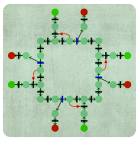


ГЛАВА 16
Игра «Жизнь» 177
Двумерные клеточные автоматы (177) Игра «Жизнь» (178) Построение модели (179) Упражнения (183)

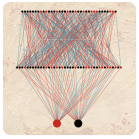
ГЛАВА 17
Блочные клеточные автоматы 187
Законы сохранения (187) Блочные автоматы (188) Клеточный газ (188) Построение модели (190) Упражнения (195)

ГЛАВА 18
Марковские системы 199
Одномерный случай (199) Двумерные М-системы (200) Построение модели (201) Упражнения (208)

ГЛАВА 19
Системы Линденмайера 211
D0L-системы (211) Графическая интерпретация (212) Построение модели (213) Упражнения (218)



ГЛАВА 20
Сети Петри 221
Понятие сети Петри (221) Параллелизм в сетях Петри (222) Моделирование трафика (222) Построение модели (224) Упражнения (230)



ГЛАВА 21
Перцептрон Розенблатта . . . 233
Искусственный нейрон (233) Нейронные сети (234) Перцептрон Розенблатта (235) Построение модели (236) Упражнения (243)

ПРИЛОЖЕНИЕ
Словарь NetLogo 247

Введение

Естественные вычисления • Тематика настоящего издания связана с проблемами построения и визуализации так называемых естественных вычислительных моделей. *Естественные вычисления*[1] — это относительно новый раздел прикладной науки, образовавшийся на стыке математики, информатики и естественных наук (прежде всего биологии). Под естественными вычислениями принято понимать различные математические и компьютерные модели, прообразом которых являются разнообразные природные системы и процессы. Сюда входят как классические модели, например клеточные автоматы, нейронные сети и генетические алгоритмы, так и относительно новые, в частности модели роевого интеллекта.

[1] Англ. — Natural Computing.

Отличительной особенностью практически всех такого рода моделей является их *распределенность*, или *многоагентность*. Каждая модель состоит, как правило, из большого числа относительно просто устроенных объектов — агентов. Сложное нетривиальное поведение моделей обеспечивается взаимодействием этих простых объектов, это называется системным эффектом (поведение системы сложнее поведения ее частей, рис. 1), или *эмерджентностью*. Исследование эмерджентных свойств систем до сих пор остается фундаментальной проблемой в разных областях науки.

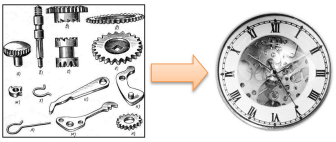


РИС. 1 Системный эффект

Среда моделирования NetLogo • Методы аналитического исследования естественных вычислительных моделей, как правило, являются очень скудными. Основным методом их изучения оказывается компьютерное моделирование. Отметим, что во многих случаях интересующий исследователей

РИС. 2 Ури Виленски

[2] U. Wilensky, W. Rand, *An introduction to agent-based modeling: Modeling natural, social and engineered complex systems with NetLogo*, Cambridge: MIT Press, 2015.

[3] По состоянию на июнь 2010 года насчитывалось порядка 250 реализаций Logo с момента создания языка!

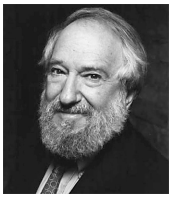


РИС. 3 Сеймур Пейперт

модели эффект достигается только при очень большом числе входящих в нее объектов. Это, в свою очередь, определяет серьезные требования к программному и аппаратному обеспечению процесса моделирования. Однако для первоначального ознакомления с данной тематикой вполне достаточно иметь какую-нибудь простую систему прототипирования многоагентных моделей.

В качестве такой системы в настоящем учебнике предлагается использовать среду многоагентного моделирования NetLogo, разработанную в конце 1990-х годов американским ученым и педагогом Ури Виленски[2]. В основу этой свободно распространяемой среды положен язык программирования NetLogo — отдаленный потомок языка черепашьей графики Logo, разработанного с образовательными целями еще в 1960-х годах одним из классиков искусственного интеллекта Сеймуром Пейпертом[3].

Вселенная в модели NetLogo состоит из двумерного поля, поделенного на отдельные клетки — патчи, по которым перемещаются разного рода агенты, называемые черепахами (рис. 4). Последние могут взаимодействовать как между собой, так и с окружающей их средой.

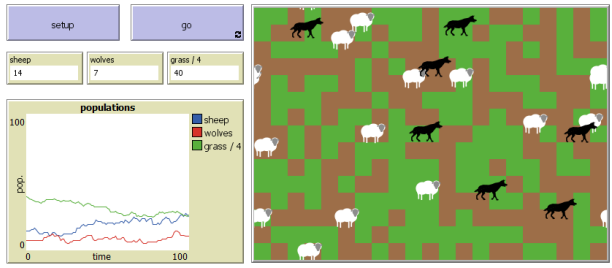


РИС. 4 Модель NetLogo «Wolf Sheep Predation»

Неотъемлемой чертой процесса построения моделей в NetLogo является их визуализация, разработчик модели может управлять в динамике большим количеством визуальных атрибутов как черепах, так и патчей. Кроме того, среда NetLogo поддерживает построение разного рода графиков, что делает ее полноценной средой численного исследования поведения разрабатываемых моделей.

Структура книги • Каждая глава издания посвящена рассмотрению какой-нибудь одной модели (либо семейства моделей). В первых параграфах главы дается теоретическое описание модели, после чего идет подробное, фактически пошаговое описание процесса построения этой модели в среде NetLogo[4].

В конце каждой главы приводится список заданий для самостоятельной проработки изложенного в главе материала: технические упражнения по доработке построенной модели, задания по численному исследованию поведения данной модели и «творческие» задания по разработке и визуализации аналогичных моделей.

Изложение материала учебника сопровождается большим числом иллюстраций, таблиц, графиков и библиографических ссылок. В приложении к учебнику дается краткое описание всех конструкций языка NetLogo, использовавшихся при построении моделей. Более полная документация доступна через систему помощи среды или на официальном сайте NetLogo. Крайне полезным также будет изучение моделей из большой библиотеки готовых моделей NetLogo, включающей в себя, в частности, примеры использования многих конструкций языка и интерфейса NetLogo.

Автор с благодарностью примет конструктивные или доброжелательные комментарии, отзывы и пожелания, которые читатель сможет оставить по адресу netlogo.tutorial@gmail.com.

[4] Для разработки моделей автором использовалась версия 6.0 среды NetLogo.

ГЛАВА 1
Задача Бюффона

Методы Монте-Карло • Семейство методов Монте-Карло представляет собой собрание разнообразных методов и моделей, объединенных идеей проведения большой серии стохастических вычислительных экспериментов с последующей статистической обработкой их результатов. Причем чаще всего под статистической обработкой понимается вычисление математического ожидания и, возможно, среднеквадратичного отклонения. Автором метода Монте-Карло (как *вычислительного алгоритма*) является американский математик Станислав Улам.

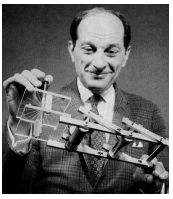


РИС. 1.1 Станислав Улам

Идея алгоритма пришла Уламу в то время, когда он лежал в больнице и занимался раскладыванием карточного пасьянса Кэнфилд (аналог нашей Косынки). Он попытался найти вероятность того, что этот пасьянс сойдется. Однако эта задача оказалась слишком сложной для методов теории вероятностей. Тогда ему и пришла в голову мысль применить следующий подход — надо разложить пасьянс n раз и определить количество m тех случаев, когда пасьянс сошелся. Тогда отношение m/n (так называемое выборочное среднее) и даст приближенную оценку искомой вероятности. Этот алгоритм был практически сразу применен к задаче вычисления многомерных определенных интегралов; заметим, что эта задача до сих пор остается одним из важнейших применений метода Монте-Карло, мы рассмотрим ее подробнее в следующей главе.

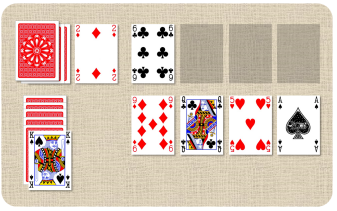


РИС. 1.2 Начальная конфигурация в пасьянсе Кэнфилд

РИС. 1.3 Николас Метрополис

Название метода было придумано Николасом Метрополисом, коллегой Улама, как утверждает-

ся, в честь его дяди — страстного игрока в рулетку. В 1949 году они с Уламом опубликовали совместную статью, которая так и называлась — «Метод Монте-Карло»[1].

[1] N. Metropolis, S. Ulam, *The Monte Carlo Method*, Journal of the American Statistical Association, Vol. 44, No. 247, 1949, p. 335–341.

Задачи, решаемые методами Монте-Карло, могут быть как вероятностными, так и детерминированными. В первом случае метод Монте-Карло применяется, например, для того, чтобы численно определить те или иные характеристики распределения интересующей нас случайной величины, когда аналитически это сделать или очень сложно, или даже невозможно. Именно такую задачу и рассматривал Улам, когда придумал данный метод.

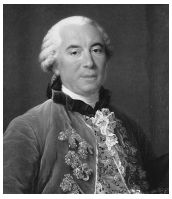


РИС. 1.4 Жорж-Луи Леклерк, граф де Бюффон

Тем не менее методы Монте-Карло применяются и для решения детерминированных задач. Идея состоит в том, что нужная нам (детерминированная) величина связывается с некоторыми статистическими параметрами, найденными при проведении серии случайных экспериментов. К этому классу задач как раз и относится вычисление определенных интегралов (т.е. площадей, объемов и т.д.). Классической же иллюстрацией такого подхода является *задача Бюффона*[2], которая была поставлена и решена задолго до официального появления метода Монте-Карло.

[2] Buffon's needle problem.

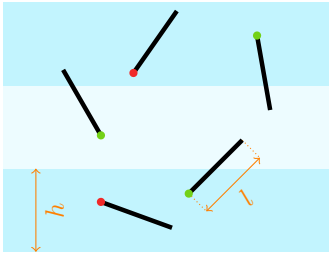


РИС. 1.5 Задача Бюффона

Задача Бюффона • В этой задаче у нас имеется плоская поверхность, на которой проведены параллельные прямые, отстоящие друг от друга на заданное расстояние h (рис. 1.5). Простым примером такой модели может служить дощатый пол или разлинованный лист бумаги.

На эту поверхность бросаются иголки (мы, однако, будем бросать спички), длина которых l меньше расстояния h. Каждая спичка может пересечь какую-нибудь прямую (зеленая спичка) или не пересечь ни одну из них (красная спичка). Вероятность p того, что спичка пересечет некоторую прямую, может быть вычислена аналитически[3]:

[3] В нахождении этой вероятности, собственно, и заключалась оригинальная задача Бюффона.

$$p = \frac{2l}{\pi h}. \tag{1.1}$$

Если теперь провести большую серию таких экспериментов по бросанию спичек и подсчитать частоту (выборочное среднее) $\hat{p} = n/m$ тех случаев,

когда спички пересекают какую-нибудь линию, то с помощью формулы 1.1 оказывается возможным найти приближенное значение числа π[4]:

$$\pi \approx \hat{\pi} = \frac{2l}{h\hat{p}} = \frac{2ln}{hm}. \qquad (1.2)$$

Особенно удобно пользоваться этой формулой, когда длина спички ровно в два раза меньше расстояния h. В этом случае число π приблизительно оценивается как n/m.

В таблице 1.1 приведены результаты простого компьютерного эксперимента по решению задачи Бюффона методом Монте-Карло. Эта таблица хорошо демонстрирует тот факт, что методы Монте-Карло, при всей их универсальности, обладают существенным недостатком — скорость их сходимости к точному решению, как правило, очень низкая. В частности, погрешность между выборочным средним и математическим ожиданием с ростом n ведет себя как $1/\sqrt{n}$ (рис. 1.6). Это означает, что для получения решений с высокой точностью требуется проведение *очень* большого числа экспериментов.

При этом отметим, что методы Монте-Карло легко и эффективно *распараллеливаются*, что отчасти решает проблему их медленной сходимости. Например, в задаче Бюффона все бросания можно выполнять одновременно, т.к. они являются взаимно независимыми. Именно этот прием мы и применим при построении нашей первой распределенной модели в среде NetLogo.

Построение модели • Наша цель — смоделировать задачу Бюффона, выполняя броски заданного числа спичек на разлинованную горизонтальными линиями поверхность. Модель должна показывать результат каждого броска, раскрашивая головку спички в красный или зеленый цвет (рис. 1.5). По результатам бросков спичек должна выполняться оценка числа π. Кроме того, интерфейс модели должен содержать графики сходимости метода (рис. 1.6).

A Создаем новую модель (меню File → New).

B Через диалог модели (по клику правой кнопки мыши на черном поле) настраиваем ее параметры:

[4] Хотя сам Буффон такой опыт не проводил (см. E. Behrends, *Buffon — Did he actually throw sticks?* Newsletter of the EMS, 2014, June, p. 47–49), опыт с бросанием иголок в XIX веке неоднократно проводился. Например, в 1864 году некий капитан О.С. Фокс, выполнив около 500 бросаний иглы, нашел приближенное значение $\pi \approx 3.1423$ (см. A. Hall, *On an experimental determination of Pi*, The Messenger of Mathematics, Vol. 2, 1872, p. 113–114).

ТАБЛ. 1.1 Сходимость метода Монте-Карло в задаче Бюффона

n	$\hat{\pi}$
10	4.00000
100	3.01886
1000	3.00751
10 000	3.15768
100 000	3.14545
1000 000	3.14233
10000 000	3.14041
100000 000	3.14157

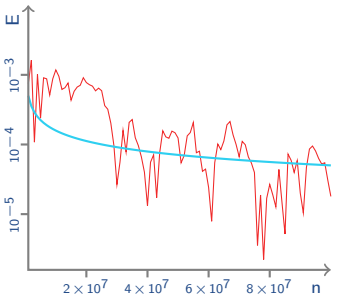


РИС. 1.6 Сравнение экспериментальной (голубая линия) и аналитической (красная линия) погрешности метода Монте-Карло в задаче Бюффона

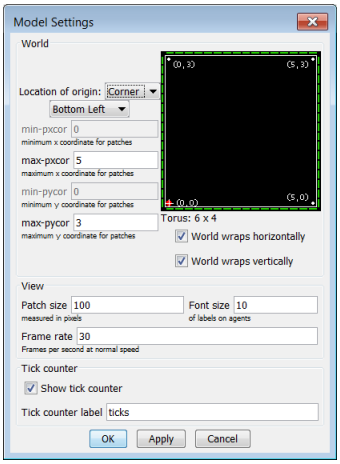


РИС. 1.7 Диалог с настройкой параметров модели

[5] Можно явным образом указать текст на кнопке, если ввести его в поле Display name в диалоге настройки.

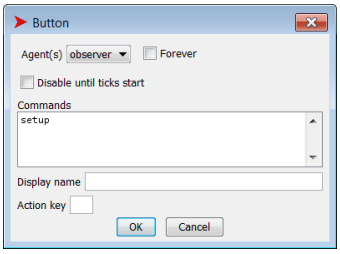


РИС. 1.8 Настройка параметров кнопки

[6] В частности, при этом происходит обнуление всех переменных.

начало координат помещаем в левый нижний угол (Location of origin → Corner, Bottom Left), размеры max-pxcor и max-pycor полагаем равными 3 и 5 соответственно, размер патча Patch size — 100 пикселям (рис. 1.7).

C Добавляем к интерфейсу кнопку setup, для этого нажимаем на кнопку Add на панели инструментов при выбранном типе элемента Button (справа от кнопки Add) и указываем место расположения новой кнопки. Сразу же открывается диалог с настройкой параметров новой кнопки (рис. 1.8). В поле Commands указываем команду `setup`, которая будет вызываться при нажатии этой кнопки. Эта же строка будет по умолчанию и текстом кнопки[5]. В результате в нашем интерфейсе появится новая кнопка, но цвет ее текста будет красным — это значит, что связанная с ней команда пока нами не определена.

D Переходим во вкладку Code и пишем код процедуры `setup` для настройки начальной конфигурации модели.

```
1 to setup
2   clear-all
3   ask patches [
4     ifelse pycor mod 2 = 0
5     [set pcolor blue]
6     [set pcolor sky]
7   ]
8   reset-ticks
9 end
```

В первой строке мы объявляем новую процедуру `setup` с помощью ключевого слова `to`. Первой командой процедуры `setup` практически всегда является команда `clear-all` (строка 2), которая очищает модель от результатов предыдущих запусков[6]. После этого происходит настройка всех патчей с помощью команды `ask patches` — при выполнении этой команды происходит перебор в некотором *случайном* порядке всех патчей модели и для каждого патча выполняются действия, указанные в квадратных скобках команды. В нашем случае мы проверяем (команда `ifelse`, строка 5), в какой строке располагается данный патч (с четной или нечетной y-координатой, `pycor mod 2 = 0`), и в зависимости от этого просим установить (команда `set`,

строки 6 и 7) цвет текущего патча (`pcolor`) либо синим (`blue`), либо голубым (`sky`)[7]. Последней командой процедуры `setup`, как правило, идет команда `reset-ticks`, которая сбрасывает таймер модели и вызывает ее прорисовку. Код любой процедуры или функции должен заканчиваться ключевым словом `end`.

E Переходим во вкладку Interface и проверяем работу кнопки setup — после нажатия на нее модель должна принять вид, показанный на рис. 1.10.

F Добавляем к интерфейсу слайдер (slider), используя кнопку Add. В диалоге настройки связываем слайдер с переменной `turtles-number`[8], которая будет обозначать число выбрасываемых за один раз спичек, в качестве минимального значения указываем 1 (рис. 1.11). Теперь в любом месте кода нам будет доступна переменная с указанным именем.

G Добавляем к интерфейсу кнопку go, связанную с командой `go`. При ее настройке включаем переключатели Disable until ticks start (блокировка кнопки до инициализации таймера, которую мы выполняем в процедуре `setup`) и Forever (автоматическое нажатие кнопки после завершения работы связанной с ней команды).

H Создаем процедуру `go` и пишем ее код, в котором моделируется бросание заданного количества спичек.

```
1 to go
2   create-turtles turtles-number
3   [setup-turtle]
4   tick
5 end
```

В строке 2 мы с помощью команды `create-turtles` создаем заданное количество `turtles-number` черепах, каждую из которых просим выполнить код, указанный в скобках, в данном случае это процедура `setup-turtle`, код которой мы определим на следующем шаге. Команда `tick` служит для обновления таймера и вызывает перерисовку модели[9].

I Создаем процедуру `setup-turtle`, в начале которой определим визуальные характеристики черепахи:

[7] Цвета в NetLogo представляются числами от 0 до 140, подобрать нужный цвет можно с помощью инструмента Color Swatches (меню Tools, рис. 1.9).

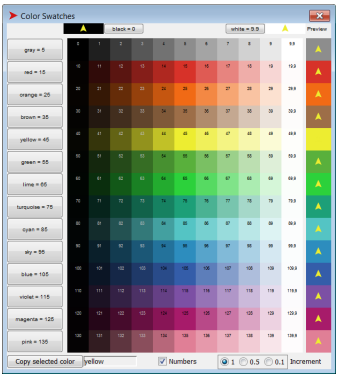


РИС. 1.9 Цвета в NetLogo

РИС. 1.10 Начальная конфигурация модели

[8] Имена переменных в NetLogo могут содержать в себе практически любые символы (кроме скобок).

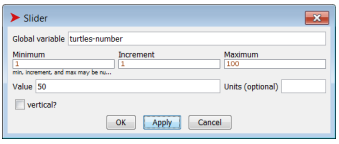


РИС. 1.11 Настройка слайдера

[9] При выбранном режиме показа on ticks справа в панели инструментов на вкладке с интерфейсом. Для всех рассматриваемых нами моделей рекомендуется использовать именно этот режим.

```
1 set shape "circle"
2 set size 0.1
3 set color black
4 set pen-size 4
```

В первой строке задаем форму черепахи (`shape`) — круг, затем ее размер (`size`[10]), цвет (`color`[11]) и толщину следа (`pen-size`[12]).

J Саму спичку мы будем рисовать в два приема, сначала поместим черепаху в случайную точку модели, потом передвинем в случайном направлении на заданное расстояние с прорисовкой следа. Таким образом получим отрезок, на одном конце которого и будет располагаться черепаха (визуально это будет головка спички, рис. 1.12). Цвет спички мы должны определить из условия пересечения черепахой горизонтальной линии между патчами. Это мы можем сделать, просто сравнив цвета патчей в начальной и конечной точках: если они различаются, то пересечение есть. Таким образом, вторая часть процедуры `setup-turtle` будет выглядеть следующим образом.

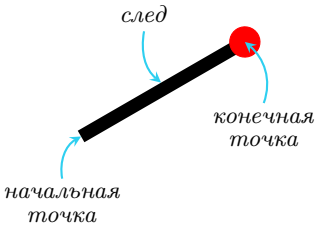


РИС. 1.12 Схема рисования одной спички

```
1 setxy random-xcor random-ycor
2 pendown
3 let c1 pcolor
4 fd 0.4
5 let c2 pcolor
6 ifelse c1 = c2
7 [set color red]
8 [set color lime]
```

При создании черепахи она автоматически помещается в начало координат и ориентируется в случайном направлении, переместить ее в заданную точку можно с помощью команды `setxy` (строка 1), два параметра которой задают координаты точки. Случайные координаты точки[13] можно получить с помощью команд `random-xcor` и `random-ycor`. После этого командой `pendown` включаем режим прорисовки следа[14]. Цвет патча, в котором располагается в данный момент черепаха, определяется командой `pcolor`. В строке 2 мы запоминаем в переменной `c1` цвет в начальной точке, команда `let` создает новую *локальную* переменную. В третьей строке командой `fd`[15] мы передвигаем черепаху на расстояние 0.4 патча в ее текущем (случайном) направлении. Запоминаем цвет нового патча в переменной `c2`, после чего сравниваем два цвета (стро-

[10] Измеряется в патчах.

[11] Это также цвет следа, который оставляет черепаха при своем перемещении.

[12] Измеряется в пикселях.

[13] С учетом реальных размеров модели.

[14] Этот режим можно отключить командой `penup`. По умолчанию режим прорисовки следа отключен.

[15] Сокращение от `forward`, можно использовать и такой вариант команды.

ка 5). Если цвет патчей одинаковый, то окрашиваем черепаху в красный цвет (`red`), иначе — в светло-зеленый (`lime`).

K Переходим в интерфейс модели и проверяем ее работу, нажимая сначала на кнопку setup, а затем на кнопку go. Чтобы остановить работу модели, надо еще раз нажать на кнопку go. После выполнения нескольких итераций модель должна выглядеть, как на рис. 1.13. Заметим, что по умолчанию в NetLogo применяется так называемое *циклическое замыкание* границ, когда правая граница фактически оказывается склеенной с левой, а верхняя — с нижней. Поэтому в нашей модели некоторые черепахи при движении за одну из границ поля появляются с его другой стороны[16].

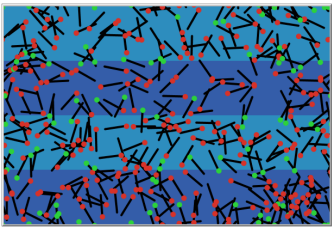


РИС. 1.13 Вид модели после нескольких итераций

[16] Этот режим можно поменять в настройках модели с помощью двух переключателей World wraps horizontally и World wraps vertically.

L Теперь займемся приближенной оценкой числа π согласно формуле (1.2). Для этого нам потребуются два счетчика — число всех черепах (переменная n) и число черепах, которые при своем движении пересекли горизонтальную границу между патчами (переменная m). Создадим соответствующие глобальные переменные в *начале* нашего кода с помощью следующей команды.

```
globals [n m]
```

Аргументом этой команды является список (в квадратных скобках) имен глобальных переменных.

M Инициализацию этих переменных выполним в процедуре `setup` с помощью следующих двух команд.

```
set n 0
set m 0
```

Их надо вставить в любом месте процедуры между командами `clear-all` и `reset-ticks`[17], например после команды (D:2).

[17] В принципе, эта инициализация не является обязательной в нашем случае, т.к. команда `clear-all` и так обнуляет все глобальные переменные.

N Наконец, в процедуре `setup-turtle`, перед командой (J:6) с проверкой на равенство двух цветов, увеличиваем `n` на 1.

```
set n n + 1
```

А перед командой (J:8) смены цвета черепахи на светло-зеленый (*внутри* квадратных скобок) увеличиваем на 1 аналогичным образом и переменную `m`.

```
set m m + 1
```

O Определим три вспомогательные функции (в терминах NetLogo функции называются *репортерами*). Функция `approx-pi` будет вычислять приближенное значение числа π согласно (1.2).

```
1 to-report approx-pi
2   report 2 * 0.4 * n / m
3 end
```

Функции, в отличие от процедур, в NetLogo определяются командой `to-report`. Значение, возвращаемое функцией, определяется командой `report`. В нашем случае (строка 2) возвращаемое значение задается формулой (1.2) с учетом того, что $h = 1$ — расстояние между полосами у нас равно одному патчу, а $l = 0.4$ — такой шаг черепахи мы установили в (J:4).

P Вторая функция `model-error` вычисляет экспериментальную погрешность, т.е. модуль разности между точным значением числа π и приближенной оценкой.

```
1 to-report model-error
2   report abs (approx-pi - pi)
3 end
```

Модуль числа в NetLogo находится функцией `abs`.

Q Наконец, функция `theory-error` будет вычислять теоретическую оценку погрешности $1/\sqrt{n}$.

```
1 to-report theory-error
2   report 1 / sqrt n
3 end
```

Квадратный корень из числа в NetLogo определяется с помощью функции `sqrt`.

R Перейдем на вкладку с интерфейсом и добавим к нему *монитор* (monitor), используя кнопку Add. В открывшемся окне настройки указываем следующие параметры (рис. 1.14). В поле reporter указываем выражение `approx-pi`. В поле Display Name указываем строку pi[18]. Проверяем работу модели (кнопки setup и go) — в новом мониторе будет отображаться приближенное значение числа π, которое постепенно должно становиться все более близким к точному значению[19].

S Последним пунктом построения нашей модели является включение в нее графика сходимости, показывающего, как изменяется погрешность в зависимости от числа n черепах (т.е. брошенных спи-

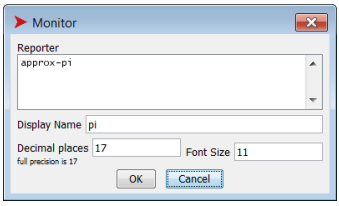


РИС. 1.14 Настройка монитора

[18] Если оставить это поле пустым, то заголовок монитора будет содержать выражение из поля reporter.

[19] 3.1415926535897932384 — заметим, однако, что такая точность в нашей модели является недостижимой.

чек). Выбираем тип элемента Plot, нажимаем кнопку Add и выполняем настройку графика (рис. 1.15). В поле Name (заголовок графика) указываем Error plot, в поле X axis label (подпись оси горизонтальной оси) — n, в поле Y axis label (подпись оси вертикальной оси) — error, в поле Y max (начальное максимальное значение по вертикальной оси) — 1. Включаем переключатели Auto scale? (автоматическое масштабирование графика) и Show legend? (показывает названия кривых). Далее настраиваем сами графики (группа элементов управления Plot pens), у нас их два — экспериментальная погрешность и теоретическая погрешность. Выбираем цвет первой кривой (например, красный). В поле Pen name указываем имя model, в поле Pen update commands записывается команда рисования:

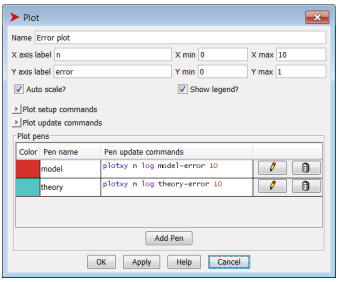


РИС. 1.15 Настройка графика

```
plotxy n log model-error 10.
```

Команда `plotxy` определяет график зависимости одной величины от другой. Первый ее аргумент (`n`) соответствует величине, откладываемой по оси абсцисс, второй (`log model-error 10`, десятичный логарифм ошибки) — по оси ординат. Если нажать на кнопку с изображением карандаша, то можно получить доступ к дополнительным настройкам кривой (рис. 1.16). Например, можно определить тип кривой mode как Point. Аналогичным образом настраиваем вторую кривую (предварительно нажав на кнопку Add pen). Цвет — голубой, имя кривой — theory, команда рисования:

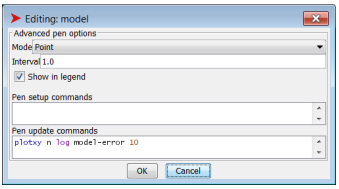


РИС. 1.16 Дополнительные настройки кривой

```
plotxy n log theory-error 10.
```

T Проверяем работу модели. Окончательный интерфейс модели должен иметь вид, как на рис. 1.17.

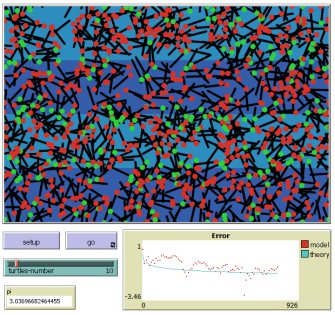


РИС. 1.17 Окончательный интерфейс модели

УПРАЖНЕНИЯ

1 Выполните вручную опыт Бюффона. Вместо иголок можно использовать любые линейные объекты — спички, зубочистки, булавки. В Интернете можно найти описания даже таких экзотических опытов, как бросание замороженных сосисок для хот-догов.

2 Если после 10 итераций погрешность метода Монте-Карло в задаче Бюффона составила 10^{-1}, то сколь-

ко итераций надо выполнить, чтобы достичь точности а) 10^{-2}; б) 10^{-4}; в) 10^{-8}?

3 Проведите серию численных экспериментов с различными размерами модели — числом патчей по вертикали и горизонтали[20]. Влияет ли изменение размеров на скорость сходимости метода?

[20] Поля max-pycor и max-pycor в настройках модели.

4 Убедитесь экспериментально, что любое замыкание границ модели существенным образом влияет на приближенную оценку числа π. Причиной этого явления является особенность поведения черепах в NetLogo, которые оказались перед одной из границ и следующим шагом должны пересечь эту границу. Если границы замкнуты, то черепаха по правилам NetLogo остается на месте, и по нашему методу определения факта пересечения горизонтальной линии она эту линию гарантированно не пересечет. Поэтому число m оказывается заниженным, а соответствующая оценка числа π — завышенной. Замыкание каких границ сильнее ухудшает оценку?

5 Наш вариант вычисления приближенного числа π в (O:2) может приводить к ошибке деления на нуль. Это будет происходить на первых итерациях, при условии что первые броски спички оказываются неудачными (т.е. не пересекающими горизонтальных линий). Тогда значение `m` будет оставаться нулевым, а в коде процедуры `approx-pi` мы выполняем деление на это значение. Т.к. этот код вызывается только из графика, то ошибка в его выполнении не будет приводить к остановке всей модели. Но в любом случае, это ошибочная ситуация, и ее следует исправить. Простейшим способом сделать это (с учетом специфики нашей задачи) является инициализация значений `n` и `m` в процедуре `setup` единицами, а не нулями, что будет выглядеть так, как будто первый бросок спички был успешным, что никак не должно повлиять на оценку числа π при больших значениях `n`.

6 Длина спички в коде нашей модели фиксирована и равна 0.4. Такого рода константы в программировании называются *магическими*[21]. Одним из способов решения этой проблемы является полное комментирование кода (комментарии в NetLogo начинаются с символа `;` (точка с запятой) и продолжаются до конца строки). Однако лучшим решением будет вообще избегать такого рода констант в коде, а использовать именованные константы или переменные. В нашем случае нужно ввести дополнительную глобальную переменную `match-length`, например с помощью команды

[21] Если наш код будет смотреть кто-то незнакомый с описанием всей задачи (или мы сами спустя некоторое время), то он, скорее всего, не сможет уловить связь между двумя использованиями этого значения в (J:4) и (O:2), особенно если в последнем случае мы заменим `2 * 0.4` на `0.8`.

`globals`, настроить ее в процедуре `setup` и затем использовать там, где встречается данная константа. Более предпочтительный вариант — добавить к интерфейсу еще один слайдер с именем `match-length`, настроить его пределы и значение по умолчанию, что автоматически даст нам глобальную переменную с таким именем и сделает ненужным ее инициализацию в процедуре `setup`.

7 Формула (1.1) корректно описывает вероятность пересечения одной из параллельных линий только при условии $l < h$ (проверьте экспериментально). Если же длина иглы больше расстояния между прямыми, то эта формула будет верной, только если при подсчете пересечений учитывать полное их количество (двойное пересечение — плюс два к счетчику m и т.д., см. рис. 1.18). Для реализации такой модели надо работать не с цветом патча, на котором располагается черепаха, а с ее y-координатой (`ycor`). Дополнительно надо учесть, что при движении за верхнюю границу черепаха оказывается снизу, и наоборот. Для этого будет полезным атрибут `dy`, указывающий, в частности, на направление движения черепахи (положительное значение — движение вверх, отрицательное — вниз).

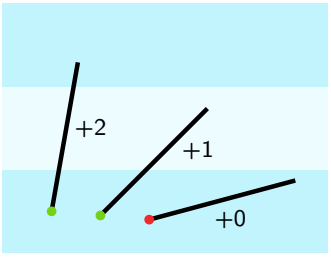


РИС. 1.18 Учет числа пересечений для случая длинной иглы

8 В обобщении Лапласа задачи Бюффона[22] поверхность разлинована горизонтальными и вертикальными линиями, расстояния между которыми равны a и b соответственно. В предположении, что игла короче обеих длин a и b, вероятность пересечения ею при случайном броске хотя бы одной линии выражается формулой

$$p = \frac{2l(a+b) - l^2}{\pi ab}. \qquad (1.3)$$

Адаптируйте построенную нами модель задачи Бюффона под задачу Бюффона-Лапласа. Для этого, например, надо раскрасить патчи в шахматном порядке с использованием четырех цветов (почему?) и учесть, что в этом случае $a = b = 1$ (рис. 1.19).

[22] B. J. Arnow, *On Laplace's Extension of the Buffon Needle Problem*, The College Mathematics Journal, Vol. 25, No. 1, Jan. 1994, p. 40–43.

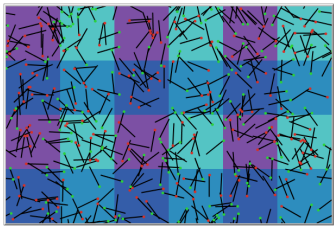


РИС. 1.19 Модель Бюффона-Лапласа

9 Еще одним интересным обобщением задачи Бюффона об игле (needle) является задача Бюффона о лапше (noodle)[23]. Оказывается, что формула (1.1) остается корректной даже для случая одной иглы длины l *произвольной* формы. Т.е. для приближенного определения числа π достаточно бросить на разлинованную поверхность *одну* ($n = 1$) длинную кривую (ту самую лапшу) и подсчитать количество m ее пересечений всех заданных линий. Реализуйте описанную вариацию задачи Бюффона. За основу можно взять классический

[23] J. F. Ramaley, *Buffon's Noodle Problem* // The American Mathematical Monthly, Vol. 76, No. 8 (Oct., 1969), p. 916–918.