ЗАДАНИЕ 2
Задача Бюффона

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. По результатам бросков спичек должна выполняться оценка числа π. Кроме того, интерфейс модели должен содержать графики сходимости метода.

A Создаем новую модель (меню File → New).

B Через диалог модели (по клику правой кнопки мыши на черном поле) настраиваем ее параметры: 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. 3.1).

C Добавляем к интерфейсу кнопку setup, для этого нажимаем на кнопку Add на панеле 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. 3.2). В поле Commands указываем команду `setup`, которая будет вызываться при нажатии этой кнопки. Эта же строка будет по умолчанию и текстом кнопки[1]. 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.

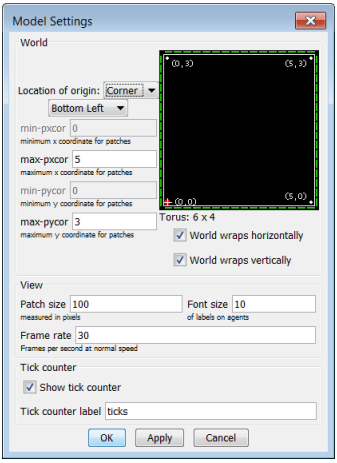

РИС. 2.1 Диалог с настройкой параметров модели

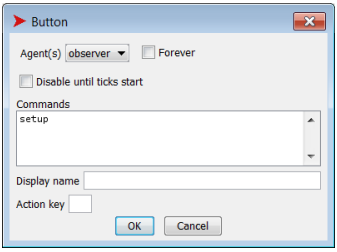

РИС. 2.2 Настройка параметров кнопки

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

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[2]. 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)[3]. 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.

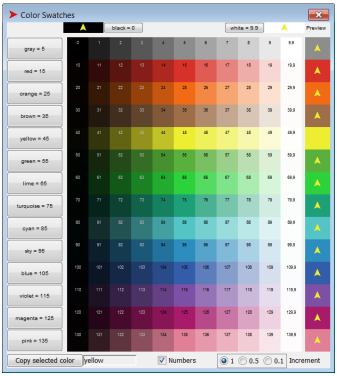

РИС. 2.3 Цвета в NetLogo

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

[3] Цвета в NetLogo представляются числами от 0 до 140, подобрать нужный цвет можно с помощью инструмента Color Swatches (меню Tools, рис. 3.3).

РИС. 2.4 Начальная конфигурация модели

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

E Переходим во вкладку Interface и проверяем работу кнопки setup — после нажатия на нее модель должна принять вид, показанный на рис. 3.4.

F Добавляем к интерфейсу слайдер (slider), используя кнопку Add. В диалоге настройки связываем слайдер с переменной turtles-number[4], которая будет обозначать число выбрасываемых за один раз спичек, в качестве минимального значения указываем 1 (рис. 3.5). 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[5].

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

```
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[6]), цвет (color[7]) и толщину следа (pen-size[8]).

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. 3.6). 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, вто-

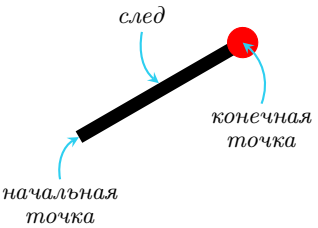

РИС. 2.6 Схема рисования одной спички

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

[6] Измеряется в патчах.

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

[8] Измеряется в пикселях.

рая часть процедуры `setup-turtle` будет выглядеть следующим образом.

```
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[9] можно получить с помощью команд `random-xcor` и `random-ycor`. После этого командой `pendown` включаем режим прорисовки следа[10]. 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`[11] 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 (строка 5). Если цвет патчей одинаковый, то окрашиваем черепаху в красный цвет (`red`), иначе — в светло-зеленый (`lime`).

[9] С учетом реальных размеров модели.

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

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

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. 3.7. 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[12].

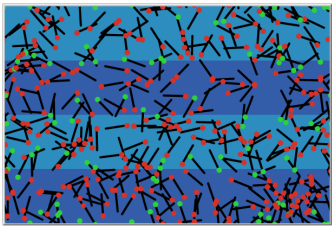

РИС. 2.7 Вид модели после нескольких итераций

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

L Теперь займемся приближенной оценкой чис-

ла π согласно формуле (3.1):

$$\pi \approx \frac{2ln}{hm}, \tag{2.1}$$

где h — ширина полос, l — длина одной иголки. 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`[13], например после команды (D:2).

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 (3.1).

```
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

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

задается формулой (3.1) 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.

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

Модуль числа в NetLogo находится функцией `abs`.

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

```
to-report theory-error
  report 1 / sqrt n
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. 3.8). В поле reporter указываем выражение `approx-pi`. В поле Display Name указываем строку pi[14]. 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[15].

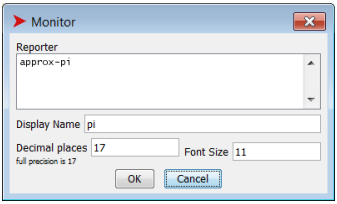

РИС. 2.8 Настройка монитора

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

[15] 3.1415926535897932384 — заметим, однако, что такая точность в нашей модели является недостижимой.

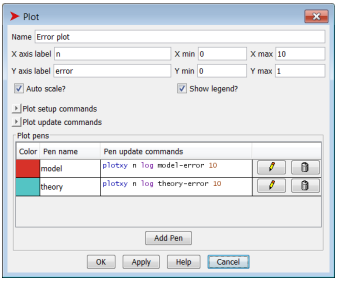

РИС. 2.9 Настройка графика

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 спичек). Выбираем тип элемента Plot, нажимаем кнопку Add и выполняем настройку графика (рис. 3.9). 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:

```
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. 3.10). 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:

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

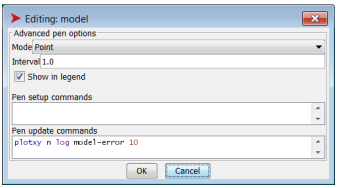

РИС. 2.10 Дополнительные настройки кривой

T Проверяем работу модели. Окончательный интерфейс модели должен иметь вид, как на рис. 3.11.

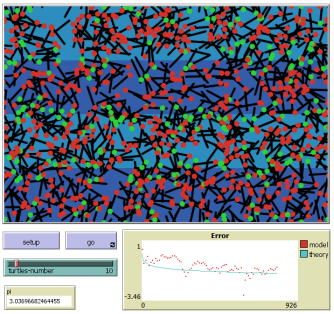

РИС. 2.11 Окончательный интерфейс модели

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 сколько 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[16]. Влияет ли изменение размеров на скорость сходимости метода?

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

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

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*[17]. 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 `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`.

[17] 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`.

7 Формула

$$p = \frac{2l}{\pi h}. \tag{2.2}$$

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. 3.12). 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).

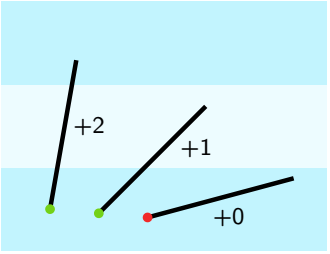

РИС. 2.12 Учет числа пересечений для случая длинной иглы

8 В обобщении Лапласа задачи Бюффона[18] 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

$$p = \frac{2l(a+b) - l^2}{\pi ab}. \qquad (2.3)$$

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. 3.13).

[18] 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.

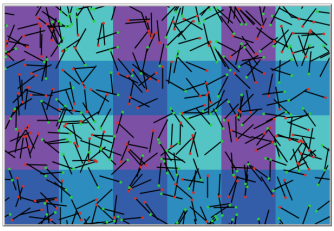

РИС. 2.13 Модель Бюффона-Лапласа

9 Еще одним интересным обобщением задачи Бюффона об игле (needle) является задача Бюффона о лапше (noodle)[19]. Оказывается, что формула (3.2) 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 вариант, рассмотренный в главе, и произвести в ней следующие изменения. К списку глобальных переменных добавить переменные `dl` (шаг черепахи, инициализировать его небольшим значением, например 0.1, в процедуре `setup`) и `l` (полная длина пути, пройденного черепахами, инициализировать нулем), переменную `n` из кода удалить. Создание черепах перенести в `setup`, в процедуре `setup-turtle` оставить только первую ее часть, отвечающую за определение визуальных характеристик черепахи. В процедуре `go` сделать вызов про-

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

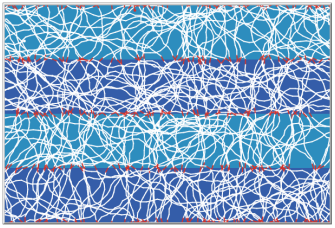

РИС. 2.14 Внешний вид модели Buffon's noodle

цедуры `move-turtle`, за основу кода которой взять код второй части «старой» процедуры `setup-turtle`. При этом надо изменить команду движения черепахи вперед на `dl`, обновить значение `l` и после этого выполнить случайный поворот вправо на угол от $-30°$ до $30°$ с помощью команды `right`[20] `random`[21] `60 - 30`. Чтобы корректно отображать цвет при пересечении горизонтальной линии (рис. 3.14), определение цвета `c2` и смену цвета черепахи нужно производить *до* ее движения. Цвет `c2` можно определить с помощью команды

```
let c2 [pcolor] of patch-ahead dl.
```

Функция `patch-ahead dl` возвращает патч, куда должна попасть черепаха, пройдя расстояние `dl` в ее текущем направлении. Конструкция `[a] of X` используется для получения значения атрибута `a` агента `X`. Последним шагом надо удалить кривую theory из графика модели, а для кривой model изменить команду рисования на `plot log model-error 10`.

10 Интересное приложение задачи Бюффона описано в работе[22], в которой авторы экспериментально показали, что муравьи вида *Leptothorax* при поиске места для нового гнезда оценивают площадь помещений с помощью приближенного решения модифицированной задачи Бюффона. Оказывается, что площадь S плоской фигуры можно оценить путем подсчета числа n пересечений двух нитей длины a и b, расположенных случайным образом в границах этой фигуры:

$$S \approx \frac{2ab}{\pi n}. \tag{2.4}$$

Мы вернемся к реализации этой модели позже при рассмотрении моделей поведения муравьев, основанных на применении ими феромонов.

[20] Вместо `right` можно использовать короткий вариант `rt`.

[21] Функция `random num` возвращает случайное целое число от 0 до `num` (не включительно).

[22] E. B. Mallon, N. R. Franks, *Ants Estimate Area Using Buffon's Needle*, Proceedings of the Royal Society B: Biological Sciences 267.1445, 2000, p. 765–770.