ЗАДАНИЕ 13
Системы Линденмайера

Построение модели • Рассмотрим построение модели, реализующей D0L-системы с описанной выше черепашьей визуализацией. Для иллюстрации возможностей модели включим в нее три L-системы: квадратный остров Коха, кривую Гильберта и фрактальное дерево. Кривая Гильберта является примером непрерывной кривой, полностью заполняющей пространство квадрата, в котором она строится (рис. 13.1). Соответствующая этой кривой L-система задается следующим образом:

$$\begin{cases} A \to +BF - AFA - FB+, \\ B \to -AF + BFB + FA-, \\ \omega_0 = A,\ \delta = 90^\circ. \end{cases} \qquad (13.1)$$

Символы A и B здесь служат для управления ростом структуры, рисующим устройством они должны игнорироваться.

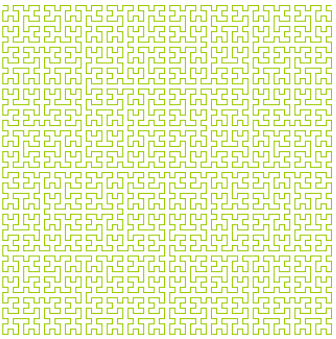

РИС. 13.1 Кривая Гильберта

РИС. 13.2 Давид Гильберт

A Создаем новую модель. Устанавливаем размер патча равным 20 пикселей.

B Правила L-систем будем хранить в черепахах вида `chars`, создаем такой вид, приписываем ему атрибуты: `v` — символ левой части правила, `alpha` — подстановка, `cmd` — команда рисующего устройства, ассоциированная с символом `v`.

C Создаем процедуру `add-rule [u al cm]`, которая добавляет в систему новое правило. В коде этой процедуры мы создаем одну новую черепаху вида `chars`, делаем ее невидимой, устанавливаем ее атрибуты `v`, `alpha` и `cmd` равными `u`, `al` и `cm` соответственно.

D Создаем функцию `get-alpha [u]`, которая по заданному символу `u` должна находить связанное с ним правило (черепаху вида `chars`) и возвращать значение атрибута `alpha` найденного правила. Если в системе для символа `u` нет правила, то функция должна возвращать символ `u` (тривиальная подстановка).

```
to-report get-alpha [u]
  let c one-of chars with [v = u]
  if c = nobody [report u]
  report [alpha] of c
end
```

E Создаем аналогичную функцию `get-cmd [u]`, которая возвращает команду (атрибут `cmd`), связанную с символом `u`, или сам этот символ, если в системе нет для него правила.

F Создаем глобальные переменные: `chain` — символьная цепочка, соответствующая текущему состоянию системы, `d` — текущий шаг для команд F и f, `delta` — угол поворота для команд $+$ и $-$. Кроме того, для рисования структуры, генерируемой L-системой, нам потребуются следующие пять параметров, которые также будем хранить в глобальных переменных:

- `x0` и `y0` — координаты стартовой точки рисования;
- `delta-0` — начальная ориентация рисующего устройства;
- `k-shorten` — коэффициент укорачивания параметра `d` на каждой последующей итерации эволюции L-системы[1].

Значения этих параметров следует подбирать экспериментально для каждой L-системы.

[1] Многие L-системы имеют тенденцию к экспоненциальному росту. Чтобы избежать соответствующего экспоненциального роста размера их изображения, параметр d следует постепенно уменьшать.

G Пишем код процедуры `setup-koch-island`, которая должна создать систему правил и настроить параметры L–системы, генерирующей квадратный остров Коха.

```
to setup-koch-island
  add-rule "F" "F-F+F+FF-F-F+F" "F"
  set chain "F-F-F-F"
  set d 16
  set delta 90
  set x0 -8
  set y0 8
```

```
  set delta-0 90
  set k-shorten 0.25
end
```

H Для настройки L-системы, генерирующей простое дерево, пишем код аналогично устроенной процедуры `setup-simple-tree`, в которой создаем одно правило для символа F:

```
add-rule "F" "F[+F]F[-F]" "F"
```

Значения параметров системы устанавливаем согласно левому столбцу таблицы 13.1.

I Создаем аналогичную процедуру для настройки L-системы, генерирующей кривую Гильберта, `setup-hilbert-curve`. В этой системе должно быть три правила для символов A, B и X:

```
add-rule "A" "+BX-AXA-XB+" "A"
add-rule "B" "-AX+BXB+XA-" "B"
add-rule "X" "X" "F"
```

Подстановка для символа X является тривиальной, но рисующим устройством этот символ интерпретируется как команда F — движение с прорисовкой следа. Значения параметров устанавливаем согласно правому столбцу таблицы 13.1.

J Добавляем к интерфейсу кнопку `setup` и выпадающий список `l-system` с тремя опциями: `"Koch island"`, `"Simple tree"`, `"Hilbert curve"`. Создаем процедуру `setup`, в которой: очищаем модель, окрашиваем все патчи в белый цвет[2], вызываем одну из трех описанных выше процедур настройки параметров L-системы согласно значению переменной `l-system`. Создаем одну черепаху (наше рисующее устройство), для которой вызываем процедуру `init-turtle`. Кроме того, запоминаем эту черепаху в глобальной переменной `turtle-1`: `set turtle-1 self`. Создаем еще одну глобальную переменную `pos`, в которой будет храниться текущая позиция в цепочке `chain`, определяющая символ — команду рисующей черепахе. Инициализируем эту переменную нулем. Последней командой процедуры `setup` традиционно указываем команду сброса таймера.

K Пишем код процедуры `init-turtle`, в котором: помещаем черепаху в точку с координатами `x0`, `y0`; ориентируем ее в направлении `delta-0`; устанавливаем форму `"turtle"`; размер — 2; толщину пера — 2; цвет — ярко-красный (`hsb 0 100 100`).

ТАБЛ. 13.1 Параметры рисования L-систем для простого дерева (Д) и кривой Гильберта (Г)

	Д	Г
`chain`	F	A
`d`	16	24
`delta`	25	90
`x0`	0	−12
`y0`	−14	−12
`delta-0`	0	90
`k-shorten`	0.5	0.5

[2] Попробуйте также черный цвет фона.

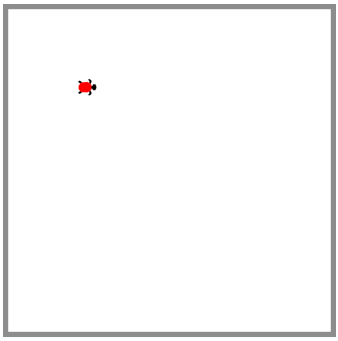

РИС. 13.3 Начальное состояние модели для острова Коха

L Проверяем работу кнопки setup, нажатие которой, помимо всего прочего, должно помещать черепаху в разные места модели согласно выбранному типу L-системы (рис. 13.3).

M Создаем кнопку go и пишем код соответствующей процедуры. За один такт работы модели будем выполнять ровно одну команду рисования согласно текущей позиции pos в текущей цепочке chain. Как только цепочка заканчивается, останавливаем процесс выполнения данной процедуры командой stop, чтобы пользователь мог спокойно рассмотреть построенную структуру. После повторного нажатия кнопки go обновляем цепочку по правилам L-системы, переустанавливаем параметры рисования и начинаем очередной процесс рисования.

```
to go
  if pos = length chain [restart]
  ask one-of turtles with-max [who] [run-cmd]
  set pos pos + 1
  if pos = length chain [stop]
  tick
end
```

Первый условный оператор отвечает за повторный старт, если текущая позиция pos вышла за пределы строки (нумерация символов начинается с нуля). Следующей командой просим черепаху с самым большим идентификатором who[3] выполнить текущую команду рисования, используя процедуру run-cmd. После этого увеличиваем значение указателя pos на 1, если достигнут конец текущей цепочки — останавливаем работу кнопки go. Последней командой обновляем таймер.

[3] Вместо организации стека состояний мы будем использовать стек черепах, вершина стека — это последняя созданная (текущая) черепаха, т.е. черепаха с максимальным значением who. Добавлению состояния в стек будет соответствовать создание копии текущей черепахи с помощью команды hatch, а удаление текущей черепахи будет приводить к переходу к предыдущему состоянию.

N Создаем процедуру restart, в которой: очищаем все нарисованное ранее (clear-drawing); обновляем цепочку командой update-chain; устанавливаем указатель pos равным нулю; умножаем d на k-shorten; просим черепаху turtle-1 выполнить команду init-turtle.

O Создаем вспомогательную процедуру to-list, которая будет преобразовывать заданную строку c в список символов, последовательно перенося их из начала строки в конец списка.

```
to-report to-list [c]
  let l []
```

```
  while [c != ""] [
    set l lput first c l
    set c but-first c
  ]
  report l
end
```

P Создаем процедуру `update-chain`, являющуюся ядром нашей модели. Сначала преобразуем текущую цепочку `chain` в список символов `l` с помощью функции `to-list`. Затем применяем к каждому элементу списка `l` соответствующее ему правило L-системы и получаем новый список `a`:

```
let a map get-alpha l.
```

Последней командой выполняем конкатенацию списка строк `a` в цепочку `chain`:

```
set chain reduce word a.
```

Q Последняя процедура — интерпретатор команд рисующим устройством (черепахой). Создаем процедуру `run-cmd`. Сначала определяем текущий цвет черепахи в формате HSB. Оттенок цвета будет изменяться от 0 до 360 градусов пропорционально значению `pos`. Яркость и насыщенность установим максимальными. После этого выясняем, какая команда `cm` связана с текущим символом. Далее пишем серию из шести условных операторов, по одному на каждую из шести описанных в главе команд[4]. Реализация первых четырех команд является очевидной. Команда [создает копию текущей черепахи, оригинал остается в текущей точке ждать, когда его копия выполнит свою часть программы и уничтожится командой], после чего продолжит выполнение своей части программы.

```
to run-cmd
  let h 360 * pos / (length chain)
  set color hsb h 100 100
  let cm get-cmd item pos chain
  if cm = "F" [pd fd d pu]
  if cm = "f" [fd d]
  if cm = "+" [lt delta]
  if cm = "-" [rt delta]
  if cm = "[" [hatch 1]
  if cm = "]" [die]
end
```

R Окончательный интерфейс модели показан на рис. 13.4. Проверяем работу модели для всех трех встроенных в нее L-систем (рис. 13.5).

[4] Таким образом, если текущая команда `cm` не входит в этот список из шести команд, то она будет просто проигнорирована.

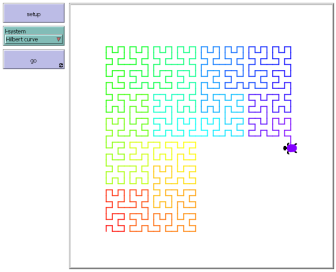

РИС. 13.4 Окончательный интерфейс модели

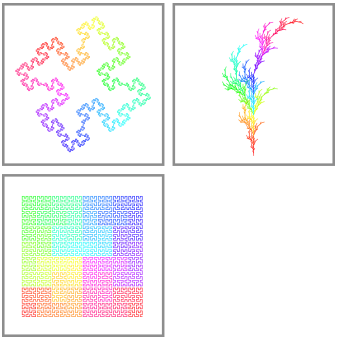

РИС. 13.5 Визуализация трех L-систем в модели

РИС. 13.6 Кривая дракона

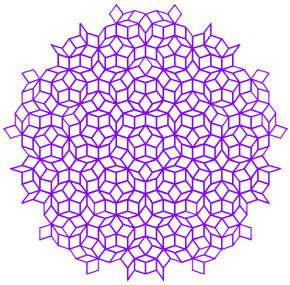

РИС. 13.7 Мозаика Пенроуза

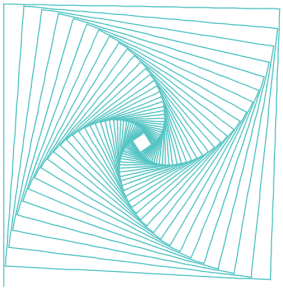

РИС. 13.8 L-система для тестирования команды $*$

УПРАЖНЕНИЯ

1 Включите в модель возможность загрузки настроек L-системы из внешнего файла. Содержимое файла может быть таким же по форме, как и тело процедуры типа `setup-koch-island`. При загрузке читаем файл построчно и сразу исполняем каждую считанную строку: `run file-read-line`. Проверка достижения конца файла выполняется командой `file-at-end?`.

2 Добавьте к модели L-систему для построения кривой дракона (рис. 13.6):

$$\begin{cases} L \to L + R+, \; R \to -L - R, \\ \omega_0 = L, \; \delta = 90^\circ. \end{cases} \tag{13.2}$$

Символам L и R соответствует команда рисования F. Особенностью этого фрактала является то, что на каждой следующей итерации построения он поворачивается на некоторый угол. Чтобы компенсировать этот эффект, введите еще один глобальный параметр `d-delta` и установите его значение в настройках L-системы равным 45°. В процедуре `restart` добавьте команду увеличения значения `delta-0` на величину `d-delta`.

3 Мозаика Пенроуза является примером *апериодического* заполнения плоскости ромбовидными плитками двух типов (рис. 13.7). Такая мозаика реализуется с помощью L-системы следующего вида (рисуется только символ F):

$$\begin{cases} W \to YF{+}{+}ZF{-}{-}{-}{-}XF[{-}YF{-}{-}{-}{-}WF]{+}{+}, \\ X \to {+}YF{-}{-}ZF[{-}{-}WF{-}{-}XF]{+}, \\ Y \to {-}WF{+}{+}XF[{+}{+}{+}YF{+}{+}ZF]{-}, \\ Z \to {-}{-}YF{+}{+}{+}{+}WF[{+}ZF{+}{+}{+}{+}XF]{-}{-}XF, \\ F \to \epsilon, \; \delta = 36^\circ, \\ \omega_0 = [X]{+}{+}[X]{+}{+}[X]{+}{+}[X]{+}{+}[X]. \end{cases}$$

4 Включите в модель другие виды L-систем, рассмотренные в первой главе книги *The Algorithmic Beauty of Plants*.

5 Добавьте черепахам атрибут `my-d`, выполните его инициализацию значением `d` в процедуре `init-turtle`. Включите в список команд рисующего устройства дополнительную команду $*$, которая умножает `my-d` на заданное число k. Протестируйте эту команду на L-системе вида (рис. 13.8):

$$\begin{cases} x \to - * Fx, \\ \omega_0 = Fx, \; \delta = 92^\circ, \; k = 0.98. \end{cases}$$

6 Разработайте L-систему для построения еще одного фрактала — *дерева Мандельброта* (рис. 13.9). Для визуализации этой системы добавьте к модели команду уменьшения толщины линии, рисуемой черепахой (атрибут `pen-size` в NetLogo).

7 Добавьте к списку команд команду S, выполнение которой заключается в оставлении отпечатка (`stamp`) черепахи в той точке и в той ориентации, где она находится в этот момент времени. Разработайте и реализуйте L-систему для построения дерева Пифагора (рис. 13.10) с использованием новой команды и формы `"square"` (предварительно через редактор форм включите ее вращение).

8 В *параметрических* L-системах отдельным символам приписываются числовые параметры, влияющие на выполнение соответствующих графических команд. Например, символу F можно приписать параметр d, соответствующий длине шага черепахи при выполнении ею команды F. Правила в параметрических системах определяют не только замену символа, они также устанавливают (в виде выражений в круглых скобках) значения параметров символов в правой части подстановки. Примером может служить следующая L-система (рис. 13.11):

$$\begin{cases} F(x) \to F(px) + F(hx) - -F(hx) + F(qx), \\ \omega_0 = F(1),\ \delta = 86^\circ,\ p = 0.7,\ q = 0.3,\ h = \sqrt{pq}. \end{cases}$$

9 Правила *контекстно-зависимых* систем включают в себя контекст, т.е. ближайшее окружение символов. Правило $L\langle v\rangle R \to \alpha$ применимо к символу v, только если он окружен правильным контекстом (слева L, справа R)[5]. В качестве примера контекстно-зависимой системы Линденмайера можно рассмотреть простую систему, моделирующую распространение сигнала вдоль цепочки символов:

$$\begin{cases} b\langle a \to b \\ b \to a. \end{cases}$$

Придумайте схему реализации таких систем в NetLogo, например ограничив себя контекстом длины 1.

10 В *стохастических* (недетерминированных) L-системах для каждого символа может быть определено несколько правил, каждому из которых приписывается вероятность его применения (аналогично марковским

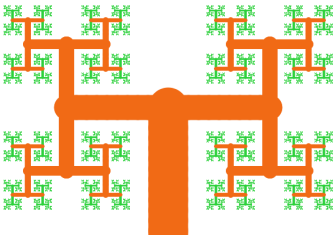

РИС. 13.9 Дерево Мандельброта

РИС. 13.10 Дерево Пифагора

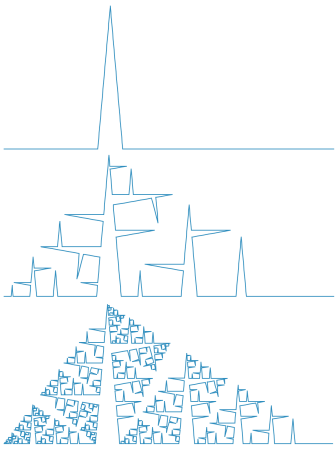

РИС. 13.11 Параметрическая L-система

[5] С этой зрения, кстати, одномерные клеточные автоматы можно считать частным случаем контекстно-зависимых L-систем.

системам). Реализуйте такой тип L-систем и протестируйте его, например, на системе вида (рис. 13.12):

$$\begin{cases} x \xrightarrow{0.5} *[--Fx]+FFx, \\ x \xrightarrow{0.5} *[++Fx]-FFx, \\ \omega_0 = Fx. \end{cases}$$

РИС. 13.12 Генерация нескольких деревьев одной стохастической L-системой