ЗАДАНИЕ 8

Алгоритм бактериального поиска

Построение модели • Наша цель — реализовать двумерную модель бактериального поиска на основе хемотаксиса и роевого поведения бактерий. Целью поиска должен быть выделенный патч, являющийся источником некоторого химического сигнала, распространяющегося диффузионным образом. Тогда роль функции качества будет играть уровень этого сигнала в данной точке модели. Кроме того, модель должна включать поддержку разного рода препятствий, мешающих прямолинейному движению бактерий.

A Создаем новую модель, устанавливаем размеры max-pxcor и max-pycor равными 20, размер патча — 10 пикселям. Убираем циклическое замыкание границ.

B Начинаем с генерации карты препятствий. Для этого добавляем к интерфейсу модели выпадающий список map-type с тремя опциями (рис. 8.1):

- `"free"` — без препятствий;
- `"blackhole"` — одно большое препятствие в форме квадрата посередине модели;
- `"bricks"` — отдельные небольшие препятствия по всему полю.

C Добавляем патчам атрибут `free?`. К интерфейсу добавляем кнопку setup-map и создаем код соответствующей процедуры, в которой сначала производим очистку мира и просим все патчи установить значение атрибута `free?` равным `true` (нет ни одного препятствия).

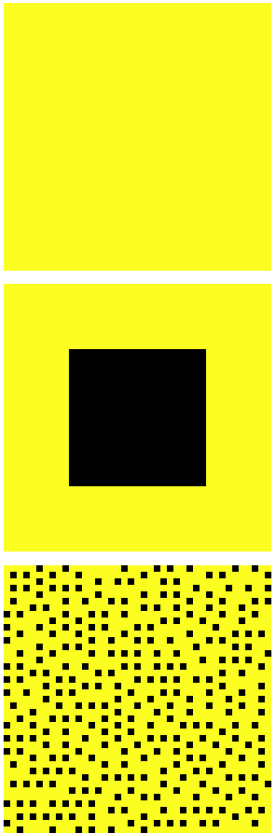

РИС. 8.1 Три типа карт с препятствиями

D Если выбран тип карты `"blackhole"`, то просим все патчи, лежащие в области $[-10, 10]$, по каждой координате установить атрибут `free?` равным `false`.

E Если выбран режим `"bricks"`, то просим все патчи выполнить следующие две команды. Сначала определяем количество соседних патчей с препятствиями:

```
set n count neighbors with [not free?].
```

Если `n = 0`, то объявляем данный патч препятствием: `set free? false`.

F Т.к. в дальнейшем мы будем работать только со свободным патчами, то создаем глобальную переменную `free-patches` и запоминаем в ней набор всех имеющихся свободных патчей:

```
set free-patches patches with [free?].
```

G Раскрашиваем патчи: препятствия — в черный цвет, свободные патчи — в желтый. Проверяем работу кнопки setup-map.

H Переходим к созданию и настройке цели. Добавляем патчам два новых атрибута: `p-val` — целевая функция, уровень сигнала в данном патче; `d-val` — вспомогательный атрибут для выполнения диффузии. Для визуализации цели поиска создаем новый вид черепах `targets`.

I Создаем кнопку setup-target и соответствующую процедуру, в начале кода которой удаляем все старые цели: `ask targets [die]`. Выбираем новый целевой патч:

```
let target-patch one-of free-patches,
```

в центре которого создаем новую цель (можно использовать команду `sprout-targets`), настраиваем визуальные параметры цели: цвет — желтый, форма — крестик, размер — 2.

J Реализуем процесс диффузии[1]. Сначала устанавливаем значение `p-val` для всех свободных патчей равным нулю. Определяем и инициализируем нулем две локальные переменные `minv` и `maxv`, в которых мы будем хранить минимальное и максимальное значения целевого сигнала. Дальше организуем цикл `while`, который будет работать до тех

[1] Использовать встроенную команду диффузии мы не можем, т.к. она работает со всеми патчами, а нам необходимо распространить сигнал из целевого патча только по свободным патчам.

пор, пока значение `minv` не станет больше порогового значения 10^{-7}. На каждой итерации этого цикла будем выполнять один шаг диффузии. Сначала к значению `p-val` целевого патча прибавляем 0.1. Затем просим все свободные патчи определить количество сигнала, которое они передадут всем своим соседям (`set d-val p-val / 20`), вычисляем количество `n` свободных соседей и уменьшаем свое значение `p-val` на величину `n * d-val`. Следующей командой просим все патчи увеличить свое значение `p-val` на величину, равную сумме значений `d-val` всех их свободных соседей. Далее находим минимальное и максимальное значения атрибутов `p-val` и просим все свободные патчи выполнить нормировку этого атрибута, так чтобы минимальное значение стало нулевым, а максимальное — единичным:

```
set p-val (p-val - minv) / (maxv - minv).
```

Последним действием процедуры вычисляем цвет каждого свободного патча, используя команду

```
my-scale-color p-val ^ 0.3,
```

после чего вызываем принудительную перерисовку модели с помощью команды `display`.

K Пишем код функции `my-scale-color`, в которой аргумент `x` из диапазона от 0 до 1 преобразуется в цвет в формате RGB: от желтого (`x = 0`) до красного (`x = 1`):

```
to-report my-scale-color [x]
  report rgb 255 (255 - x * 255) 0
end
```

Команда `rgb` формирует цвет по трем заданным компонентам — красной, зеленой и синей. Наши оттенки (от желтого до красного) получаются изменением зеленой (второй) компоненты от 255 до 0.

L Проверяем работу кнопки setup-target (рис. 8.2).

M Создаем новый вид черепах — бактерии[2]. Для них определяем атрибут `val`, для хранения последнего «считанного» значения целевой функции. Добавляем к интерфейсу слайдер colony-size — размер колонии бактерий, минимальное значение делаем равным 1. Создаем кнопку setup-colony и соответствующую ей процедуру. В коде этой процедуры сначала удаляем имеющуюся колонию бак-

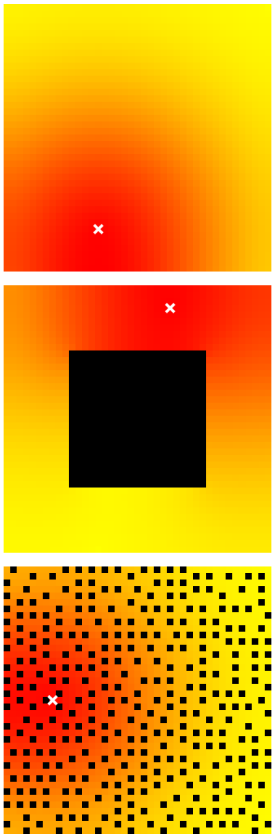

РИС. 8.2 Вид модели после настройки цели

[2] *Bacteria* во множественном числе и *bacterium* в единственном числе.

терий. Затем выбираем один из свободных патчей для образования новой колонии. В центре этого патча создаем новую колонию из заданного числа `colony-size` бактерий. Для каждой новой бактерии устанавливаем размер (0.8), форму (`"circle"`) и определяем значение атрибута `val` с помощью вызова функции `eval-me`. Последней командой в процедуре `setup-colony` сбрасываем таймер.

N Пишем код функции `eval-me`, которая пока в качестве результата возвращает значение атрибута `p-val` того патча, где располагается в данный момент бактерия.

O Создаем кнопку `go`. Пишем код процедуры `go`, в которой сначала просим все бактерии выполнить команду `move`, а затем обновляем таймер[3].

P К интерфейсу добавим слайдер `vel` (минимальное значение 0.1, максимальное — 1), соответствующий скорости бактерий.

Q Создаем процедуру `move`, которая, собственно, и будет моделировать хемотаксис бактерий. Сначала проверяем, может ли бактерия передвинуться вперед на расстояние `vel`, если не может, то выполняем ее случайный поворот и завершаем процедуру:

```
let p patch-ahead vel
if p = nobody or [not free?] of p [
  rt random 360 stop
]
```

Функция `patch-ahead vel` возвращает патч, в который попадает бактерия при перемещении на расстояние `vel`, если же такого патча нет (бактерия находится у границы модели), то эта функция возвращает значение `nobody`. Если же перемещение вперед возможно, то совершаем его: `fd vel`. Проверяем работу модели на разных картах. Бактерии должны совершать прямолинейные движения, при столкновении с препятствиями и границами менять случайным образом направление движения (рис. 8.3).

R Сам хемотаксис моделируется следующим кодом:

```
let new-val eval-me
if new-val < val [rt random 360]
set val new-val
```

[3] Добавьте по аналогии с рассмотренными ранее моделями поддержку трассировки бактерий для визуализации их траекторий движения.

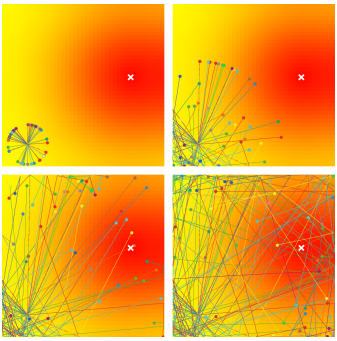

РИС. 8.3 Простое (без хемотаксиса) движение бактерий

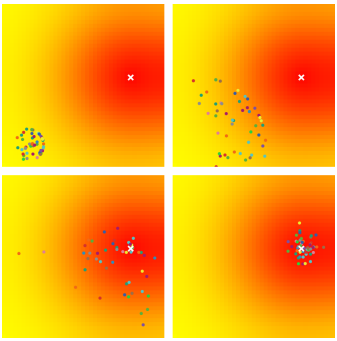

РИС. 8.4 Хемотаксис бактерий

Сначала вычисляем новое значение `new-val` целевого сигнала и сравниваем его с предыдущим значением. Если новое значение слабее предыдущего, значит, мы движемся в неправильную сторону, поэтому нужно выполнить случайный разворот. После этого обновляем значение `val` для следующего шага хемотаксиса.

S Проверяем работу модели. Теперь все бактерии должны достаточно быстро собираться в окрестности цели (рис. 8.4).

T Наконец, включим в модель поддержку роевого движения бактерий, до сих пор они двигались у нас абсолютно независимо друг от друга. Для этого достаточно модифицировать вычисление целевого сигнала в функции `eval-me`. Добавим к интерфейсу переключатель `swarm?`, который будет включать и выключать режим роения. Кроме того, добавим слайдер `d` (значения от 1 до 20) для представления параметра d в формуле (8.1).

$$\tilde{f}(x) = f(x) + \mu \sum_{k=1}^{m} h(|x_k - x|), \qquad (8.1)$$

Параметр μ в этой формуле зафиксируем равным 10^{-4}. Переписываем код функции `eval-me`. Если режим роения выключен, то возвращаем, как и раньше, значение `p-val` текущего патча. Если же этот режим включен, то сначала вычисляем дополнительное слагаемое в (8.1):

```
let a mean [h distance myself] of bacteria,
```

после чего возвращаем полное значение сигнала:

```
report p-val + 0.0001 * a.
```

U Функцию `h [r]`, вычисляющую сигнал, показанный на рис. 8.5, можно реализовать в виде суммы двух гауссианов следующим образом:

```
1 to-report h [r]
2   let x r / d
3   report 2 * exp(- x * x) - 3 * exp(-4 * x * x)
4 end
```

V Проверяем работу модели, сравнивая различные режимы ее работы, например без роения бактерий и с роением (рис. 8.6). Окончательный интерфейс модели показан на рис. 8.7.

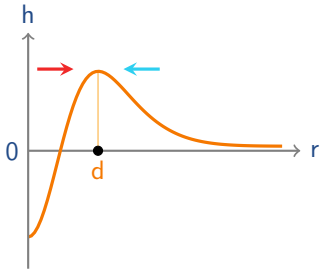

РИС. 8.5 Форма сигнала $h(r)$

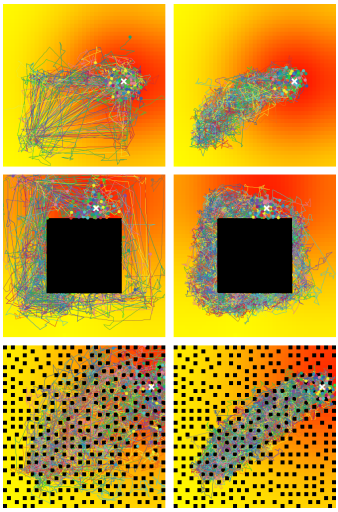

РИС. 8.6 Два режима работы модели: без роения бактерий и с роением

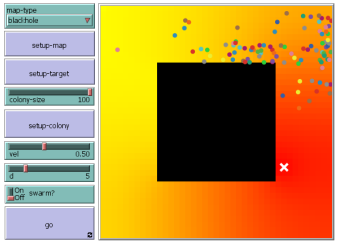

РИС. 8.7 Окончательный интерфейс модели

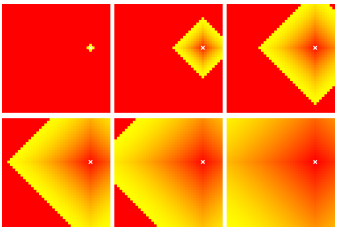

РИС. 8.8 Работа волнового алгоритма

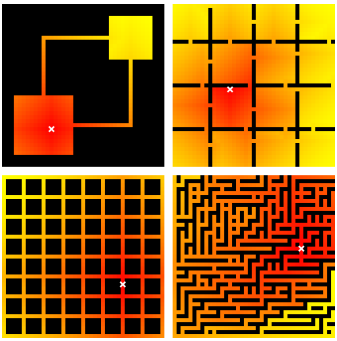

РИС. 8.9 Другие типы карт

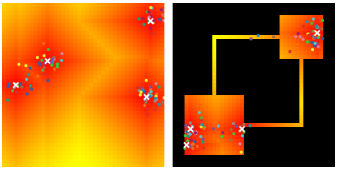

РИС. 8.10 Случай нескольких целей

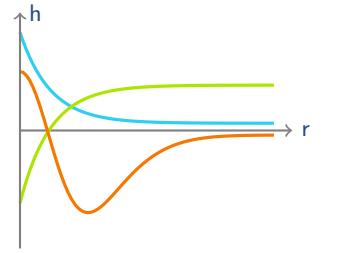

РИС. 8.11 Разные типы сигнальной функции $h(r)$

УПРАЖНЕНИЯ

1 Реализованный в нашей модели алгоритм диффузии, целью работы которого является формирование градиента сигнала в направлении целевого патча, работает достаточно медленно, особенно для сложных карт. Можно использовать более быструю схему формирования такого градиента, основанную на волновом алгоритме из теории графов (рис. 8.8): сначала для каждого патча вычисляем расстояние d от него до целевого патча, затем устанавливаем значение атрибута `p-val` равным $1 - d/d_{\max}$, где $d_{\max}$ — максимальное расстояние до цели среди всех патчей.

2 Добавьте в модель другие типы карт (рис. 8.9):

- две комнаты, соединенные коридорами;
- несколько комнат, соединенных «дверями»;
- квартальную схему;
- лабиринт (алгоритмы автоматической генерации лабиринтов можно посмотреть, например на англоязычной Википедии по запросу *Maze generation algorithm*).

Исследуйте поведение колонии бактерий для этих карт в разных режимах.

3 Рассмотрите вариант инициализации бактериального алгоритма, в котором начальная популяция не сосредоточена в отдельном патче, а распределяется случайным образом по всему полю модели.

4 Добавьте в модель возможность пользователю задавать сразу несколько целевых патчей. Рассмотрите поведение колонии бактерий при начальном случайном распределении бактерий (рис. 8.10).

5 Рассмотрите случай постоянной целевой функции $f(x) = 0$, например при отсутствии целевого патча. Проанализируйте поведение модели с включенным режимом роения бактерий. Поэкспериментируйте с другими типами функции $h(r)$ (рис. 8.11)[4]. Интересное поведение демонстрируют большие колонии бактерий для сигнала-репеллента (зеленая кривая на рис. 8.11). Если циклическое замыкание границ включено, то бактерии показывают ожидаемое поведение — они стараются отдалиться друг от друга, в результате заполняя равномерно все пространство модели (рис. 8.12, слева). Если же замыкание снять, то бактерии начинают прижиматься к краям модели, но места им всем там не хватает, поэтому часть бактерий вынуждена остаться в средней части модели (рис. 8.12, справа).

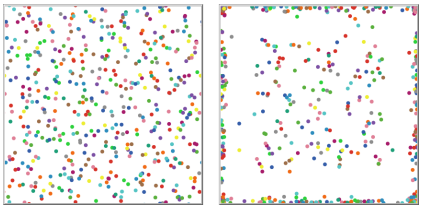

РИС. 8.12 Поведение бактерий для сигнальной функции-репеллента

6 Включите в модель оператор отбора и размножения. На каждом шаге алгоритма каждая бактерия с небольшой вероятностью выбирает «соперника», с которым устраивает турнир, побеждает та бактерия, у которой выше значение функции качества. Проигравшая бактерия удаляется из популяции, победившая — создает свою копию (команда `hatch`).

7 Включите в модель оператор рассеяния, когда каждая бактерия с малой вероятностью переносится в случайное место пространства.

8 Рассмотрите поведение модели при наличии шума в источнике сигнала. Для этого добавьте к целевой функции еще одно слагаемое — случайную величину с нормальным распределением вида `random-normal 0 0.1`.

9 Примените алгоритм бактериального поиска к задаче одномерной минимизации, взяв за основу модель для метода имитации отжига.

10 Примените алгоритм бактериального поиска к задачам двумерной оптимизации, взяв за основу модель из предыдущей главы для метода роя частиц.

11 Интересной формой коммуникации бактерий является так называемое *чувство кворума*[5], которое заключается в резкой смене типа поведения *всей* колонии при увеличении ее плотности (число бактерий на единицу объема) выше некоторого порогового значения[6]. Механизм этого явления заключается в выделении и распознавании бактериями некоторого сигнального вещества. Как только концентрация этого вещества превышает заданный пороговый уровень, бактерии переключаются на другой тип поведения. Реализуйте такую модель, включив в нее два типа агентов — бактерий и сигналов. Бактерии могут находиться в двух состояниях — синем и красном. Бактерии в синем состоянии делятся и движутся с небольшой скоростью. Сигналы движутся быстрее и с некоторой вероятностью исчезают. Как только количество сигналов в окрестности

[5] B. Bassler, *How bacteria talk to each other: regulation of gene expression by quorum sensing*, Current Opinion in Microbiology, Vol. 2, Issue 6, 1999, p. 582–587.

[6] Аналогичный эффект наблюдается также у насекомых (например у саранчи) и рыб.

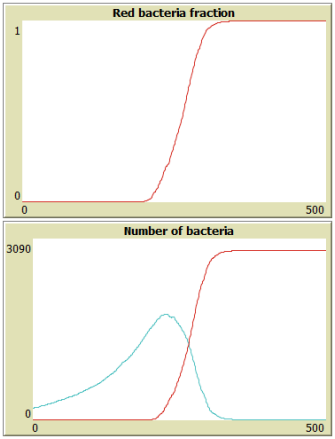

РИС. 8.13 Фазовый переход в модели чувства кворума

синей бактерии превысит заданное значение, она переходит в красное состояние и перестает делиться. На рис. 8.13 показаны зависимости числа бактерий в обоих состояниях и доля красных бактерий от времени. Хорошо виден быстрый фазовый переход всей колонии от синего состояния к красному. Пример работы такой модели приведен на рис. 8.14.

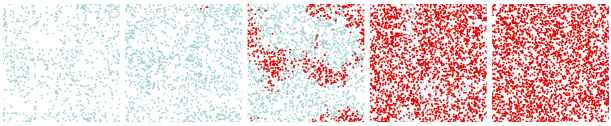

РИС. 8.14 Моделирование чувства кворума