А. С. Банников, Л. С. Чиркова

ВИЗУАЛИЗАЦИЯ ДИФФЕРЕНЦИАЛЬНЫХ ИГР ПРОСТОГО ПРЕСЛЕДОВАНИЯ

ФГБОУ ВПО «Удмуртский государственный университет»

Кафедра дифференциальных уравнений

А. С. БАННИКОВ, Л. С. ЧИРКОВА

ВИЗУАЛИЗАЦИЯ ДИФФЕРЕНЦИАЛЬНЫХ ИГР ПРОСТОГО ПРЕСЛЕДОВАНИЯ

Учебно-методическое пособие

Ижевск 2015

УДК 519.8(075)
ББК 22.183.2я 7
Б 232
Рекомендовано к изданию УМС УдГУ

Рецензент: д. ф.-м. н. Родина Л. И.

Банников А. С., Чиркова Л. С.
Б 232. Визуализация дифференциальных игр простого преследования: учебно-методическое пособие. Ижевск: ИЦ «Удмуртский университет» 2015. 48 с.

Пособие посвящено методам визуализации игр простого преследования и предназначено для студентов, обучающихся по направлениям подготовки магистров «Математическое моделирование», «Математическая кибернетика», «Математические основы компьютерных наук».

УДК 519.8(075)
ББК 22.162

ОГЛАВЛЕНИЕ

Предисловие 5

§ 1. Постановка задачи 6

§ 2. Простое движение на плоскости 7

§ 3. Решение задачи 11

§ 4. Объектная модель документа 12

§ 5. Формат SVG 14

§ 6. Рисование фигур 15

§ 7. Библиотека D3 20

§ 8. Пример визуализации игры 31

Задания для самостоятельного выполнения 38

Приложение 39

Список используемой литературы 44

ПРЕДИСЛОВИЕ

Пособие посвящено методам визуализации игр простого преследования. Простым движением называется движение участников, которые имеют возможность в каждый момент времени выбирать скорость своего движения. Более подробно с теорией дифференциальных игр можно познакомиться в [1–36].

Поставленную в первом параграфе задачу простого преследования решим методом параллельного сближения в третьем параграфе. Технологии, которые будем использовать для получения анимированного изображения в параграфе 8, описаны в параграфах 4-7.

Для освоения материала, изложенного в данном учебном пособии, необходимы знания математических дисциплин и основ программирования на уровне выпускника(бакалавра или специалиста) математических специальностей, в частности необходимы знания HTML, CSS, JS, SVG, DOM. Изучение библиотеки **D3** позволит углубить и систематизировать знания и представления о веб-стандартах.

Учебное пособие позволяет расширить свои знания в области конфликтно-управляемых процессов, приобрести знания о способах визуализации, познакомить студентов с библиотеками и инструментами, которые в дальнейшем можно использовать для web-программирования, создания анимированных информационных ресурсов.

Авторы выражают благодарность выпускникам математического факультета УдГУ 2015 года за интерес, проявленный к материалу, изложенному в данном учебном пособии.

§ 1. Постановка задачи

Рассмотрим игру, в которой участвуют один убегающий и один преследователь. Игра происходит на всей плоскости. Уравнение движения каждого из участников – дифференциальное уравнение первого порядка. Будем играть на стороне преследователя и управлять его скоростью. Поставим задачу поймать убегающего, если скорость убегающего в два раза меньше скорости преследователя. Запишем постановку формально.

В R^2 рассматрим дифференциальную игру двух лиц: преследователя P и убегающего E. Закон движения P имеет вид

$$\dot{x} = u, \quad x(0) = x_0, \quad u \in U = \big\{(u_1, u_2) : |u_1| + |u_2| \leqslant \alpha\big\}.$$

Движение E описывается уравнением вида

$$\dot{y} = v, \quad y(0) = y_0, \quad v \in V = \big\{(v_1, v_2) : |v_1| + |v_2| \leqslant \beta\big\}.$$

Игрок E выбирает программное управление $v(t)$ вида

$$v(t) = \begin{cases} v_0, & \text{если } \ t \in [0, \tau_1], \\ v_1, & \text{если } \ t \in (\tau_1, \tau_2], \\ v_2, & \text{если } \ t \in (\tau_2, \infty), \end{cases}$$

и в момент $t = 0$ сообщает P о выборе управления v_0, и момента τ_1, в момент τ_1 – о выборе v_1, τ_2, в момент τ_2 – о выборе v_2, если к соответствующему моменту τ_j встреча еще не произошла.

Игрок P использует стратегию параллельного сближения, двигаясь с максимальной по норме скоростью. Найти траекторию P и наименьший момент встречи. Момент встречи — момент T такой, что $x(T) = y(T)$. Пусть $\alpha = 4$, $\beta = 2$, $x_0 = (0, 0)$.

Решим эту задачу в третьем параграфе способом, который опишем во втором параграфе, а затем нарисуем, как будет происходить поимка.

§ 2. Простое преследование на плоскости

В данном параграфе рассматрим наиболее простые модели задач преследования, а именно: дифференциальные игры на плоскости с двумя участниками: преследователем P и убегающим E.

Местоположение преследователя в момент t будем обозначать $x(t)$, а местоположение убегающего в момент t через $y(t)$.

При этом будем считать местоположение участников геометрическими точками. Предполагается, что закон движения преследователя P имеет вид

$$\dot{x} = u, \ \|u\| \leqslant \alpha. \tag{2.1}$$

Закон движения убегающего E имеет вид

$$\dot{y} = v, \ \|v\| \leqslant \beta. \tag{2.2}$$

Здесь $x = (x_1, x_2), y = (y_1, y_2), u = (u_1, u_2), v = (v_1, v_2) \in R^2$.

Из законов движения следует, что игроки движутся с ограниченной по величине скоростью, при этом направление движения может меняться произвольным образом. Числа α, β показывают максимальные скорости игроков. Такие движения игроков называют **простым движением**.

Точки $y(t)$ при $t \in [0, T]$ описывают некоторую кривую, которую называют траекторией убегающего E на отрезке $[0, T]$. Аналогично, геометрическое место точек $x(t)$ для всех $t \in [0, T]$ называют траекторией преследователя P на отрезке $[0, T]$.

При простом движении траектории $x(t)$, $y(t)$ удовлетворяют условиям

$$\|x(t_2) - x(t_1)\| \leqslant \alpha(t_2 - t_1), \|y(t_2) - y(t_1)\| \leqslant \beta(t_2 - t_1)$$

для всех $0 \leqslant t_1 \leqslant t_2$. Если преследователь P выбирает некоторую вектор-функцию $u(t)$, $t \in [0, T]$, функцию u называют управлением преследователя, то положение P в момент t опре-

деляется следующим образом

$$x(t) = x(0) + \int_0^t u(s)ds.$$

Аналогично, если убегающий E выбирает некоторую вектор-функцию $v(t)$, $t \in [0, T]$ (функцию $v(t)$ называют управлением убегающего), то положение игрока E в момент t будет определяться следующим образом

$$y(t) = y(0) + \int_0^t v(s)ds,$$

где $x(0)$, $y(0)$ — положения игроков в начальный момент времени.

Пример 2.1. Пусть $x(0) = x_0 = (0, 1)$, $u(s) = (1, 1)$.

$$x(t) = \begin{pmatrix} 0 \\ 1 \end{pmatrix} + \int_0^t \begin{pmatrix} 1 \\ 1 \end{pmatrix} ds = \begin{pmatrix} t \\ 1 + t \end{pmatrix}$$

Таким образом, игрок P движется по прямой (рис. 1).

Пример 2.2. Пусть $y(0) = y_0 = (1, 0)$,

$$v(s) = \begin{cases} (-1, 0), \text{ если } s \in [0, 1], \\ (0, 1), \text{ если } s \in (1, \infty). \end{cases}$$

Тогда для $t \in [0, 1]$

$$y(t) = \begin{pmatrix} 1 \\ 0 \end{pmatrix} + \int_0^t \begin{pmatrix} -1 \\ 0 \end{pmatrix} ds = \begin{pmatrix} 1 - t \\ 0 \end{pmatrix}$$

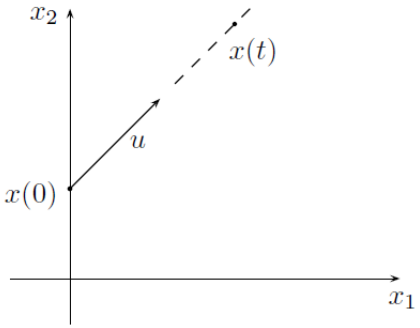


Рис. 1. Траектория игрока P

и в момент $t = 1$ убегающий E будет находиться в начале координат. Для $t > 1$

$$y(t) = \begin{pmatrix} 1 \\ 0 \end{pmatrix} + \int_0^1 \begin{pmatrix} -1 \\ 0 \end{pmatrix} ds + \int_1^t \begin{pmatrix} 0 \\ 1 \end{pmatrix} ds = \begin{pmatrix} 0 \\ t-1 \end{pmatrix}$$

Получаем траекторию игрока E при $t > 1$ (рис. 2)

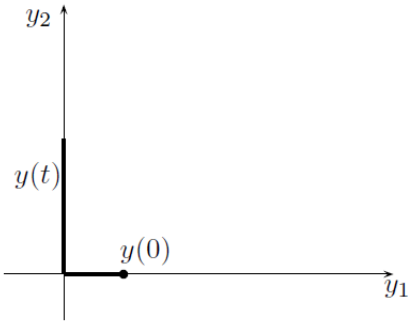


Рис. 2. Траектория игрока E

Пусть на плоскости преследователь P и убегающий E перемещаются в соответствии с простым движением с линейными

скоростями α и β, причем $\alpha \geqslant \beta$. Будем предполагать, что игроки движутся по ломаным с максимальными скоростями, причем на любом конечном отрезке времени они могут менять направление движения конечное число раз. Предположим, кроме того, что в любой момент времени t преследователь P знает свое местоположение $x(t)$, местоположение убегающего $y(t)$ и направление движения убегающего в этот момент t.

О п р е д е л е н и е 2.1. Стратегией параллельного сближения называется следующий способ преследования. Пока убегающий движется по лучу $y(0)A$, преследователь – по лучу $x(0)B$, причем для всех t выполнены соотношения

а) отрезок $x(t)y(t)$ параллелен отрезку $x(0)y(0)$;

б) $\|x(t_2) - y(t_2)\| \leqslant \|x(t_1) - y(t_1)\|$ для всех $t_2 \geqslant t_1$.

Пусть $\alpha > \beta$. Тогда преследователь P перемещается по лучу $x(0)B_0$, где B_0 — точка, определяемая условиями

1) B_0 лежит на луче $y(0)A$;

2) $\frac{\|x(0)-B_0\|}{\alpha} = \frac{\|y(0)-B_0\|}{\beta}$.

Таким образом, B_0 — точка на луче $y(0)A$, в которой преследователь P и убегающий E оказываются в один и тот же момент времени t (рис. 3).

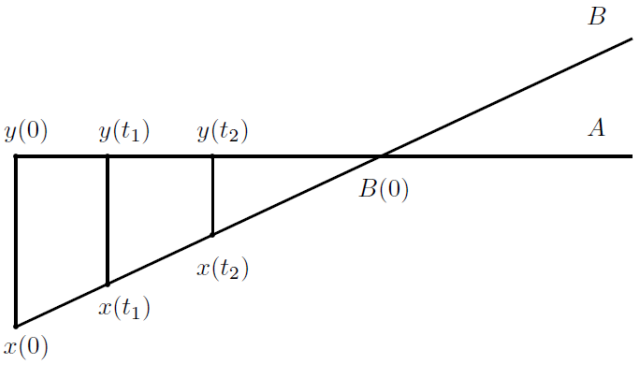


Рис. 3. Метод параллельного сближения

Предположим, что в момент τ убегающий E меняет направление своего движения и в течении некоторого времени перемещается по лучу $y(\tau)A_1$. Тогда преследователь P движется по лучу $x(\tau)B_1$, где точка B_1 — точка перехвата на луче $X(\tau)A_1$ такая, что

$$\frac{\|x(\tau) - B_1\|}{\alpha} = \frac{\|y(\tau) - B_1\|}{\beta}$$

и так далее (рис. 4)

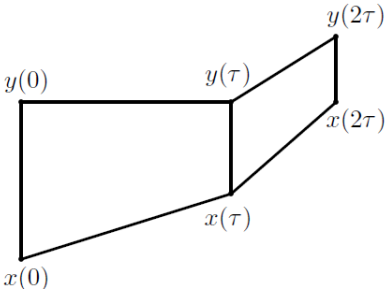


Рис. 4. Переключение управления

§ 3. Решение задачи

Пусть начальное положение убегающего $y_0 = (-12, -13)$, моменты переключения управления $\tau_1 = 2$, $\tau_2 = 5$.

Управление убегающего $v_0 = (-2, 0)$ на полуинтервале $[0, 2)$, $v_1 = (0, 2)$ на $[2, 5]$ и $v_2 = (1, 1)$, если $t > 5$.

Построим траекторию движения убегающего. При $t \in [0, 2]$

$$y(t) = \begin{pmatrix} -12 \\ -13 \end{pmatrix} + \int_0^t \begin{pmatrix} -2 \\ 0 \end{pmatrix} ds = \begin{pmatrix} -12 - 2t \\ -13 \end{pmatrix}$$

Тогда $y(2) = (-16, -13)$, $B_0 = (-12 - 2T_0, -13)$. При $t \in (2, 5]$

$$y(t) = \begin{pmatrix} -16 \\ -13 \end{pmatrix} + \int\limits_2^t \begin{pmatrix} 0 \\ 2 \end{pmatrix} ds = \begin{pmatrix} -16 \\ -17 + 2t \end{pmatrix}$$

Тогда $y(5) = (-16, -7)$, $B_1 = (-16, -17 + 2T_1)$. При $t > 5$

$$y(t) = \begin{pmatrix} -16 \\ -7 \end{pmatrix} + \int\limits_5^t \begin{pmatrix} 1 \\ 1 \end{pmatrix} ds = \begin{pmatrix} -21 + t \\ -12 + t \end{pmatrix}$$

Теперь построим траекторию движения преследователя. Подставим в равенство $\frac{\|x_0 - B_0\|}{\alpha} = \frac{\|y_0 - B_0\|}{\beta}$ свои значения: $\frac{25 + 2T_0}{4} = \frac{2T_0}{2}$, тогда $T_0 = \frac{25}{2}$, $B_0 = (-37, -13)$, $x(2) = (-\frac{148}{25}, -\frac{52}{25})$. Здесь T_0 – момент поимки, если бы не было переключения управления убегающего E.

Далее подставляя $x(2)$ и $y(2)$ в аналогичное равенство, получим $T_1 = \frac{11}{2}$, $B_1 = (-16, -6)$ – момент и координата встречи игроков, если бы не было второго переключения управления, $x(5) = (-\frac{364}{25}, -\frac{136}{25})$.

И подставив $x(5)$ и $y(5)$, найдем момент времени, когда происходит поимка $T_2 = \frac{11}{2}$, и точку, в которой игроки P и E встретятся $B_2 = (-15,5; -6,5)$.

Запомним следующие точки: начальные позиции, координаты игроков в моменты переключения управления и момент поимки: $y_0 = (-12, -13)$, $y(2) = (-16, -13)$, $y(5) = (-16, -7)$, $x_0 = (0, 0)$, $x(2) = (-\frac{148}{25}, -\frac{52}{25})$, $x(5) = (-\frac{364}{25}, -\frac{136}{25})$, $B_2 = (-15,5; -6,5)$.

В последнем параграфе запишем в два массива, и будем рисовать по ним траектории игроков.

§ 4. Объектная модель документа

DOM (Document Object Model) – это интерфейс прикладного программирования для представления документа и обеспечения доступа к его элементам и интерактивного изменения их свойств.

DOM предоставляет механизмы для изменения самой структуры документа: добавление и удаление элементов, изменение их содержимого.

Модель DOM манипулирует объектами в полном соответствии с традиционными объектно-ориентированными технологиями: все элементы документа представляются в виде объектов. В узлах структурной логической схемы находятся объекты, а не данные, со всеми присущими объектам свойствами и поведением.

Например, для HTML-документа

```
<html>
<head>
     <title>Пример представления HTML-документа в виде дерева</title>
</head>
<body>
     <h1>Представление документа в виде дерева</h1>
     <p>Абзац 1</p>
     <p>Абзац 2</p>
</body>
</html>
```

представление в виде древовидной структуры (рис. 5):

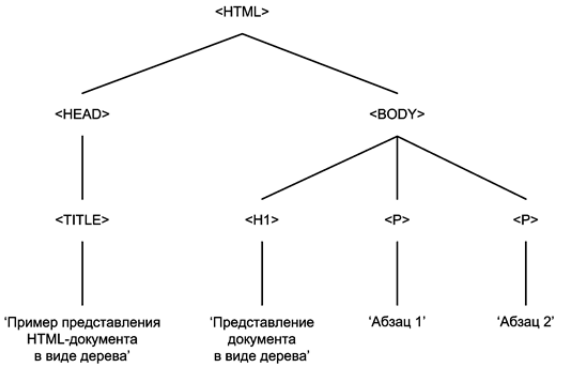


Рис. 5. Структура документа

§ 5. Формат SVG

В данном параграфе приведем краткую информацию по использованию svg-формата, необходимую для решения поставленной задачи.

Формат SVG позволяет описывать векторные изображения, форматированный текст, а также работать с растровыми изображениями: масштабировать и трансформировать их подобно векторным объектам.

Кроме того, можно создавать анимационную графику, интерактивные приложения, управлять поведением и внешним видом с помощью JavaScript и CSS. Ниже приведен типичный шаблон для SVG-файла.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg
xmlns="http://www.w3.org/2000/svg" version="1.1"
x="горизонт. координата" y="вертик. координата"
width="ширина" height="высота">
... здесь находится собственно SVG-код
</svg>
```

В первой строке находится стандартное объявление XML-документа с указанием кодировки содержимого UTF-8.

Файл с SVG-кодом нужно сохранить в той кодировке, которая указана в дескрипторе <**?xml...**> с помощью атрибута **encoding**.

Далее следует дескриптор <**!DOCTYPE...**> определения типа документа (в данном случае указан SVG версии 1.1), затем – контейнерный тег <**svg**>, внутри которого размещаются специфические для SVG дескрипторы (теги).

С помощью атрибутов **xmlns** и **version** указывают пространство имен и версию SVG, а посредством атрибутов **x**, **y**, **width**, **height** – координаты и размеры с указанием единиц измерения (**px, cm, mm, in, %**) прямоугольной области, в которой следует отобразить содержимое, заданное последующими тегами. Содержимое, выходящее за указанные рамки, при отображении будет

усечено. Если атрибуты **width** и **height** опущены, то показывается все содержимое файла SVG.

Тег <**svg**> может иметь необязательный атрибут **viewBox**, с помощью которого можно масштабировать содержимое SVG-документа. Атрибут **viewBox** принимает в качестве значения строку, содержащую четыре параметра, указанные через пробел или запятую с пробелом: горизонтальная координата, вертикальная координата, ширина и высота. Это параметры прямоугольной области в пикселах, относительно которой устанавливаются размеры всех элементов SVG-документа.

Если значения ширины и высоты в **viewBox** равны значениям атрибутов **width** и **height**, то эффект масштабирования не возникнет, а если нет – содержимое SVG-документа масштабируется.

§ 6. Рисование фигур

Фигуры могут быть замкнутыми, например, прямоугольник, круг, и разомкнутыми, например, прямые и кривые линии. Как объекты векторной графики, замкнутые фигуры состоят из внутренней части, называемой еще областью заливки, и внешнего контура (обводки). Линии можно представить себе как фигуры, состоящие только из контура. Все графические объекты задаются посредством специальных тегов с атрибутами.

Для объектов, имеющих область заливки, с помощью атрибута **fill** можно назначить ее цвет. Параметры обводки определяют посредством атрибутов **stroke** (цвет) и **stroke-width** (толщина).

Цвет задается как шестнадцатеричное значение, названием или как **rgb(r,g,b)**, где параметры – десятичные значения яркостей красной, зеленой и синей составляющих цвета в диапазоне от 0 до 255. Вот три способа задания красного цвета заливки:

```
fill="#ff0000", fill="red" и fill="rgb(255,0,0)"
```

Значение **none** применяется для области без цвета. Если атрибут **fill** опустить, то заливка будет черной. Если не указывать атрибут **stroke**, то обводки не будет.

6.1 Прямоугольник

Прямоугольник задается тегом <**rect**> с атрибутами:

- **x, y** – горизонтальная и вертикальная координаты верхнего левого угла;
- **width, height** – ширина и высота сооветственно;
- **rx, ry** – радиусы скругления углов; значения не должны превышать половины соответственно ширины и высоты прямоугольника; если указан только один из данных атрибутов, другой принимает такое же значение; если данные атрибуты не указаны, то скругление будет отсутствовать.

Параметры области заливки и обводки задают атрибуты **fill**, **fill-opacity** и **stroke**, **stroke-width**, **stroke-opacity** соответственно.

На рис. 6 показан пример создания прямоугольника с закругленными углами.

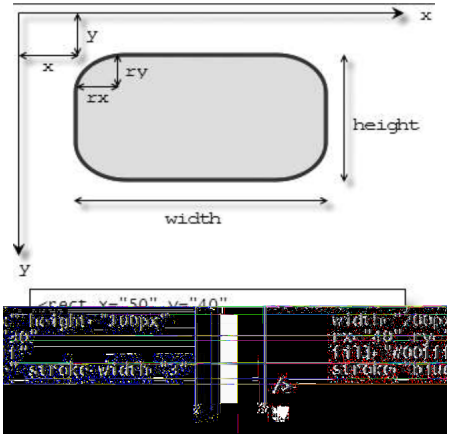


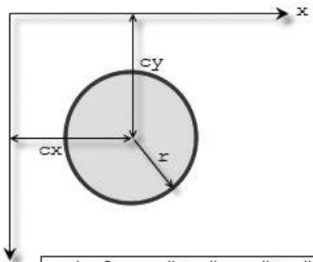
Рис. 6. Прямоугольник

Если задать значения атрибутов **rx** и **ry** равными половинам значений атрибутов соответственно **width** и **height**, то получится

овал. Если при этом значения **width** и **height** равны, то получается круг.

6.2 Круг

Круг задается тегом <**circle**> с атрибутами: **cx**, **cy** – горизонтальная и вертикальная координаты центра круга; **r** – радиус круга. На рис. 7 показан пример создания круга.




```
<circle cx="100" cy="100"
        r="50"
        fill="#00ffff"
        stroke="blue" stroke-width="3"
/>
```

Рис. 7. Круг

6.3 Многоугольник

Многоугольник задается тегом <**polygon**> с атрибутом **points**, принимающим в качестве параметра строку, которая содержит координаты вершин. Каждой вершине соответствует пара из горизонтальной и вертикальной координат, которые разделяются пробелом или запятой. Пары координат соседних вершин разделяют запятой. Как и все рассмотренные ранее замкнутые фигуры, многоугольник имеет область заливки и обводку. Пример задания пяти- и четырехугольника.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
```

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.1"
width="300" height="360" >
<title>Многоугольник</title>
<polygon points="20,10 200,20 180,90 120,150 50,100"
fill="#00ffff"
stroke="blue" stroke-width="5"
/>
<polygon points="20,210 200,220 30,290 120,350"
fill="none"
stroke="blue" stroke-width="5"
/>
</svg>
```

6.4 Эллипс

Эллипс задается тегом <**ellipse**> с атрибутами: **cx**, **cy** – горизонтальная и вертикальная координаты центра эллипса; **rx**, **ry** – длины горизонтальной и вертикальной полуосей эллипса.

На рис. 8 показан пример создания эллипса. При равенстве значений **rx** и **ry** получается круг.

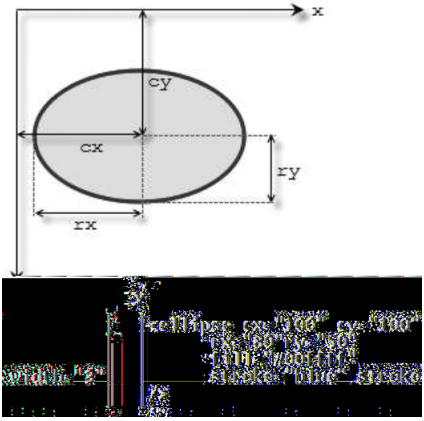


Рис. 8. Эллипс

6.5 Линии

В SVG, если линия непрямая, то область, ограниченная данной линией и отрезком прямой, соединяющим ее концы, по умолчанию залита черным цветом. Если надо нарисовать только линию, следует указать для нее отсутствие заливки. Цвет и толщину линии задают атрибуты **stroke** и **stroke-width**.

Отрезок прямой линии задается тегом <**line**> с указанием координат начала и конца с помощью атрибутов: **x1, y1** – горизонтальная и вертикальная координаты начальной точки отрезка; **x2, y2** – горизонтальная и вертикальная координаты конечной точки отрезка. Если не задавать цвет атрибутом **stroke**, то линия не будет видна.

На рис. 9 показан пример создания отрезка прямой линии. Задав достаточно большую толщину отрезка, можно получить прямоугольник.

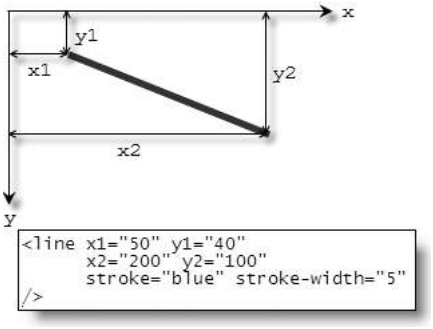


Рис. 9. Отрезок прямой линии

Атрибут **stroke-dasharray** – определяет стиль линии (пунктирная, штриховая и штрих-пунктирная); принимает в качестве значения одно, два или три числа, разделенных запятыми; единственное число указывает размер точек пунктирной (точечной) линии, пара чисел определяет соответственно размер штрихов и интервал между ними, а тройка чисел – размер точек, интервал между ними и размер штрихов между точками.

```
<line x1="10" y1="10" x2="100" y2="10"
stroke="black" stroke-width="2"
stroke-dasharray="3, 10"/>
<rect x="" y="" width="" height=""
fill="none" stroke="black"
stroke-dasharray="3, 10,15"/>
```

Атрибут **dashoffset** – определяет отступ в процентах при задании линии с использованием атрибута **stroke-dasharray**; значение по умолчанию 0.

```
<line x1="10" y1="10" x2="100" y2="10"
stroke="black" stroke-width="2"
stroke-dasharray="3, 10"
stroke-dashoffset="25%"/>
```

Атрибут **stroke-linecap** – определяет вид концов линии; возможны значения **butt** (обычный, принятый по умолчанию), **round** (округленный) и **squarel**(с выступом)

```
<line x1="10" y1="10" x2="100" y2="10"
stroke="black" stroke-width="2"
stroke-dasharray="3, 10"
stroke-linecap="round"/>
```

Атрибут **stroke-linejoin** – определяет вид соединений концов линий; возможны значения **mitre**(обычный, принятый по умолчанию), **round** (скругленный), **bevel**(рубленый).

```
<polyline points="20,80 50,20 80,80"
stroke="black" stroke-width="10"
stroke- linejoin="bevel"/>
```

Атрибут **stroke-mitrelimit** – определяет ограничение на длину соединения двух линий, заданного с помощью атрибута **stroke-linejoin** = «mitre»; значение по умолчанию 4.

§ 7. Библиотека D3

Данный параграф посвящен библиотеке **D3**, которую мы будем использовать для создания анимированного изображения.

D3(Data-DrivenDocuments) – это библиотека JavaScript, использующая цифровые данные для создания и контроля динамических и интерактивных графических элементов, которые могут отображаться в веб-браузере. Находит свое применение в разных областях: от построения диаграмм и отчетов до картографии. Кроме того, **D3** как инструмент визуализации данных поддерживает технологии, предусмотренные стандартом W3C – SVG, JavaScript, HTML5 и CSS3.

В библиотеке используется стандартная DOM-модель (Document Object Model) – объектная модель документа. Любой элемент, например, HTML-документ, таким образом, может быть представлен в виде дерева узлов, каждый из которых представляет собой образец данных – текстовый, графический или любой другой объект.

Узлы связаны между собой отношением «родитель-потомок», что позволяет программам и скриптам получать доступ к вебдокументам, изменять их структуру и содержимое, а также оформление. Программный интерфейс DOM является полностью универсальным, то есть не зависит ни от языка, ни от платформы. Пример объектной модели документа приведен в параграфе 4.

Библиотека **D3** обладает возможностью подключения к любой HTML-странице с использованием JavaScript. **D3** использует стандартные функции языка для выделения элементов, создания SVG-объектов и стилей, а также динамических эффектов.

Разберемся с основными техническими принципами библиотеки и структурой программного интерфейса. Четыре основополагающих принципа работы **D3**:

- **Выделение.** Центральный принцип **D3** – возможность выбора (выделения) необходимого набора DOM-узлов, чтобы затем использовать операторы преобразования этих данных.

- **Переход.** Еще одна функция применяется к объектам для постепенного перехода одного значения заданного атрибута к другому.

- **Привязка данных.** Для более сложных задач можно создать ситуацию, при которой загрузка данных провоцирует создание элементов. Допустим, библиотека загружает некий набор данных, а затем для каждого элемента создает SVG-объект с соответствующими параметрами (форма, цвет, значение) и поведением (переходы, события).

- **Добавление узлов.** После того как созданы все объекты, библиотека добавляет на страницу DOM-узлы, обеспечивая странице или приложению желанный внешний вид. Для визуализации больших данных это означает не только полностью «прозрачное» представление информации, но и полноценный контроль над внутренними механизмами работы программного интерфейса.

7.1 Привязка данных

Для решения поставленной задачи изучим более подробно механизм присоединения данных к элементам (data join). В качестве примера возьмем следующий HTML-документ:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <script type="application/javascript" src="http://d3js.org/d3.v3.min.js">
    </script>
  </head>
  <body>
    <script type="application/javascript">
      d3.select('body').selectAll('p')
        .data([16, 23, 42])
        .enter()
        .append('p')
        .text("Hello");
    </script>
  </body>
</html>
```

Мы видим, что библиотека **D3** подключена в пятой строке, а используется в теле документа. Выделим интересующий нас фрагмент и рассмотрим его пошагово — опишем, что происходит при выполнении данного кода в каждой строке:

```
d3.select('body').selectAll('p')
  .data([16, 23, 42])
  .enter()
  .append('p')
  .text("Hello");
```

Вызываем последовательно цепочку методов (chain syntax).

d3.select('body').selectAll('p') — этой командой мы сделали пустую выборку параграфов (элемент **p**). Такой подход позволяет не думать есть у нас эти элементы или нет, а просто сказать чего мы хотим.

.data([16, 23, 42])(рис. 10) — **data()** принимает массив объектов и объявляет отношение между выборкой и данными. Другими словами, пустая выборка связывается с массивом данных. Результатом **data()** является резервирование d3 мест под данные. Количество новых мест равно количеству мест под новые данные. Кроме того, **data()** возвращает три состояния выборки: **enter, update, exit.**

.enter() вернёт зарезервированные новые места для элементов с новыми, уже привязанными, данными (данные пишутся в __data__) и вернёт ссылку на них (это наша новая выборка зарезервированных элементов с привязанными к ним данными).

[__data__: 16, __data__: 23, __data__: 42]

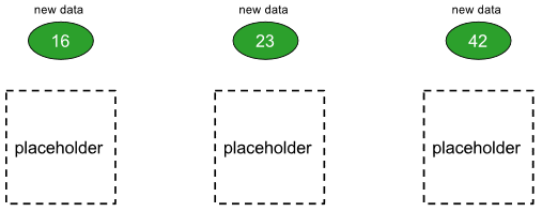


Рис. 10. Резервирование мест под данные

.append() для каждого несуществующего элемента создаётся элемент и добавляется в **dom** на ранее зарезервированные d3 места(рис. 11).

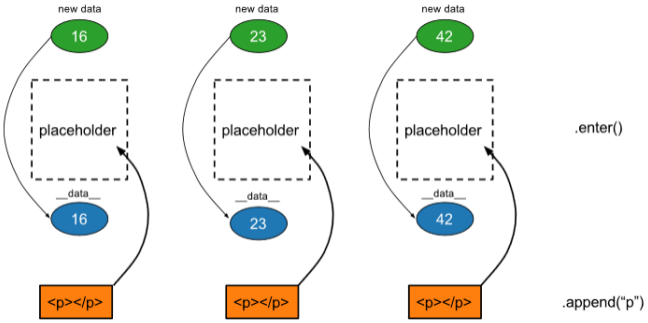


Рис. 11. Добавление элементов

.text("Hello") добавление текста в каждый элемент (рис. 12).

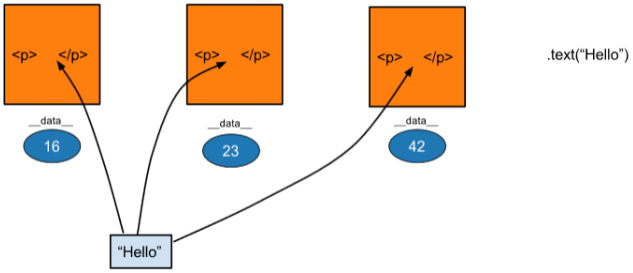


Рис. 12. Добавление текста

Данные хранятся в атрибуте __data__ элемента. Обратиться к нему можно через анонимную функцию. Например, вместо текста «Hello» можно вставить наши данные (рис. 13):

```
.text(function(d) { return d; })
```

Анонимная функция вторым параметром может принимать текущий индекс элемента. А так же внутри себя имеет объект **this**, который ссылается на текущий элемент, в данном случае – элемент <**p**>.

```
.text(function(d, i) { console.log(this);
                       console.log(i, "-", d);
                       return (d + 2 * i);})
```

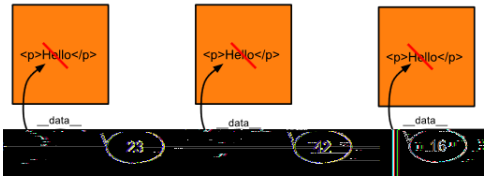


Рис. 13

Существуют два способа привязки данных: привязка по индексу и по ключу. Рассмотрим сначала привязку по индексу.

Для этого сделаем новую привязку с другими данными. Новую привязку мы выделим в отдельную переменную:

```
var p = d3.select('body').selectAll('p')
    .data([45, 16, 87, 108]);
```

Теперь в переменной **p** каждому существующему элементу было присвоено новое значение: первое значение записано в массив под индексом 0, второй под индексом 1 и т.д., и зарезервировано место для нового элемента (рис. 14).

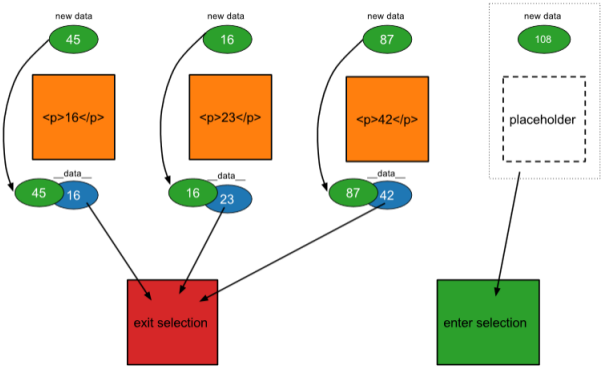


Рис. 14

В **p** хранятся три состояния выборки: enter, update, exit. В выборку **enter()** попал один элемент. Добавим его (рис. 15).

```
p.enter().append('p');
```

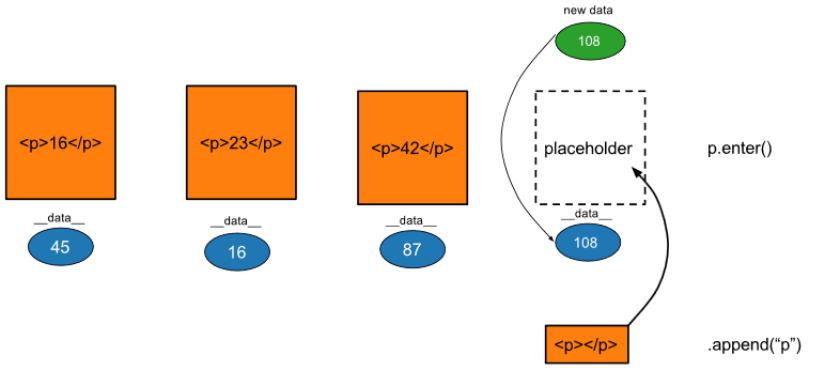


Рис. 15

На данный момент у нас есть новые данные для каждого параграфа. Осталось, используя их, обновить содержимое параграфов:

```
.text(function(d) { return d; });
```

Старые данные ушли в выборку **exit()** и их можно удалить:

```
p.exit().remove();
```

Теперь рассмотрим привязку данных по ключу. Когда вторым аргументом **data()** является анонимная функция, то привязка происходит по ключу:

```
var p = d3.select('body').selectAll('p')
        .data([45, 16, 87, 108], function(d) { return d; });
```

В данном случае мы возвращаем само значение. Важно понимать, что совпадающие по значению объекты игнорируются, например, если в данных уже есть объект 16, то в новых данных при добавлении он не будет учитываться (рис. 16).

Как правило, данные менее тривиальны. Рассмотрим более сложный пример с привязкой по ключу:

```
var data1 = [
    {"name": "Ringo", "height": 175},
    {"name": "Paul", "height": 190}];
```

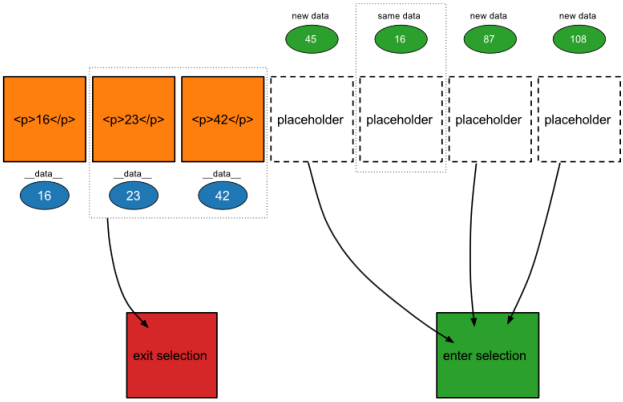


Рис. 16

```
var data2 = [
    {"name": "Paul", "height": 190},
    {"name": "Sam", "height": 170}];

d3.select('body').selectAll('p')
  .data(data1)
  .enter()
  .append('p')
  .text(function(d) { return d.height; });

var p = d3.select('body').selectAll('p')
          .data(data2, function(d) { return d.name; });
p.enter().append('p');
p.exit().remove();
p.text(function(d) { return d.height; });
```

Привязка происходит по уникальному ключу, в данном случае по ключу «name» (рис. 17).

```
var p = d3.select('body').selectAll('p')
          .data(data2, function(d) { return d.name; });
```

Сейчас в **exit** один элемент на удаление и в **enter** один элемент на добавление. Получается, что **data()** предоставляет три выборки для реализации отношений между элементами и данными. Выбор состояния зависит от количества данных и элементов.

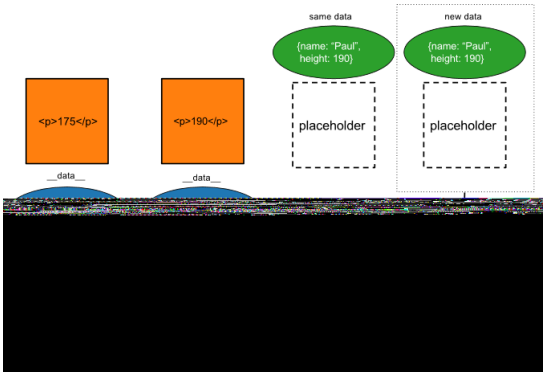


Рис. 17

Если данных больше чем элементов, то новые элементы попадают в **enter()** – выборку для добавления.

В случае, если у нас данных меньше, чем есть сейчас элементов, лишние элементы попадают в **exit()**–выборку и удаляются.

Если количество данных совпадает с количеством элементов, то происходит обновление, данных с последующим удалением старых данных, находившихся в элементах.

7.2 Вычисление значений и функторы

Для работы с DOM **D3** использует схожий API для всех вызовов. Разберём его на примере добавления или удаления класса элемента. Для этого понадобятся следующие методы:

- **selection.classed(name, value)** добавляет или удаляет класс **name** в зависимости от булевого значения **value**;
- **selection.on(event, callback)** используется для обработки событий, передавая название события **event** и функцию-обработчик **callback**.

Функции-обработчики вызываются с текущим элементом в **this**, а также **data** и **index** в аргументах. Событие — в перемен-

ной **d3.event**. Повторная установка обработчика заменит предыдущий.

```
var pressed = false
var button = d3.select('button')     //выбор кнопки
    .on('click', function (data, index) { //обработчик события <<click>>
        button.classed('pressed', pressed = !pressed)
        // в обработчике меняем значение переменной и вычисляем класс
    })
```

Аргумент функции **index** – это номер элемента в выборке, а **data** – заданный для него элемент данных.

7.3 Работа с выборкой

Рассмотрим пример, демонстрирующий работу с DOM-узлами документа через выборку:

```
var svg = d3.select('body').append('svg')

svg
    .append('text')
    .text('click somewhere')
    .attr('x', 50)
    .attr('y', 50)

var events = []
svg.on('click', function () {
    events.push(d3.event)
    if (events.length > 5) events.shift()
    var circles = svg.selectAll('circle')
        .data(events, function (e) { return e.timeStamp })
        .attr('fill', 'gray')
    circles
        .enter()
        .append('circle')
        .attr('cx', function (d) { return d.x || d.pageX })
        .attr('cy', function (d) { return d.y || d.pageY })
        .attr('fill', 'red')
        .attr('r', 10)
    circles
        .exit()
        .remove()
})
```

- Методы **selection.html()** и **selection.text()** задают или возвращают содержимое элементов в виде HTML или текста.

- **selection.style()**, **selection.attr()**, **selection.property()** задают или возвращают CSS-свойства элемента, его аттрибуты и свойства. Чаще всего используются **style** и **attr**, особенно при описании свойств новых элементов.

- Метод **selection.remove()** удаляет элементы текущей выборки.

- Метод **selection.append()** добавляет потомка к каждому элементу текущей выборки.

- Переданные в **selection.data()** данные сохраняются в поле **data** DOM-элемента, при вызове методов на выборке происходит их извлечение из элемента.

- Получить или записать данные в один элемент можно, используя **selection.datum()**.

7.4 Анимация

Наша задача – создать анамированное изображение, а именно: показать траектории движения игроков с момента начала игры до момента поимки. Поэтому будем изучать, как создать средствами библиотеки **D3** анимированные объекты.

Добавим анимацию при добавлении и удалении элементов в предыдущий пример:

```
var svg = d3.select('body').append('svg')

svg
    .append('text')
    .text('click here')
    .attr('x', 50)
    .attr('y', 50)
```

```
var events = []
svg.on('click', function () {
    events.push(d3.event)
    if (events.length > 5) events.shift()
    var circles = svg.selectAll('circle')
        .data(events, function (e) { return e.timeStamp })
        .attr('fill', 'gray')
    circles
        .enter()
        .append('circle')
        .attr('cx', function (d) { return d.x || d.pageX })
        .attr('cy', function (d) { return d.y || d.pageY })
        .attr('fill', 'red')
        .attr('r', 0) // Начальное значение
        .transition()
        .duration(1000) //Длительность перехода от нач. значения к конечному
        .attr('r', 10) // Конечное значение
    circles
        .exit()
        .transition()
        .attr('r', 0)
        .remove()
})
```

Заметим, что появились методы **.transition()** и **.duration()**.

§ 8. Пример визуализации игры

В данном параграфе напишем скрипт, который будет подгружать библиотеки **D3** из сети Интернет, а затем в несколько этапов будет рисовать ход нашей игры. Каждому этапу будет предшествовать краткое описание. Кроме того, в тексте самого скрипта для лучшего понимания комментируются строки. Полный текст html-файла без комментариев вынесен в Приложение.

Создадим новый файл с именем **index.html**. Для того, чтобы создать файл, можно воспользоваться, например, приложением **Far** в операционной системе семейства **Windows** или приложением **mc** в операционной системе семейства **Unix**.

Запустим приложение, одновременно нажмем клавиши **shift, F4**. В открывшемся окне встроенного редактора можно набрать скрипт. Укажем тип документа, кодировку, подключим библиотеки:

```
<!DOCTYPE html>
<html>
    <head>
      <title>D3JS</title>
      <meta charset="UTF-8">
      <meta name="viewport" content="width=device-width">
      <script src="http://d3js.org/d3.v3.min.js" charset="utf-8"></script>
      <style>
          body {
            font: 10px sans-serif;
            margin: 0;
          }

          .axis {
            shape-rendering: crispEdges;
          }
          .x.axis line, .x.axis path  {
            fill: none;
            stroke: #000;
          }
          .y.axis line, .y.axis path  {
            fill: none;
            stroke: #000;
          }

.rule line {
  stroke: #eee;
  shape-rendering: crispEdges;
}
.main {
overflow:hidden;
}
.sidebar {
width:860px;
float:right;
}
.content{
    margin-top:30px;
margin-bottom:30px;
margin-right:860px;
    padding-left: 30px;
    box-sizing: border-box;
}
    .content img {
      width: 100%;
    }
        </style>
```

Зададим размеры области, в которой будет отображаться ход игры, а также начало координат и число зарубок на каждой оси:

```
        <script>
            window.onload = function()
            {
              var w = 860 // svg width - ширина контейнера SVG
                  ,h = 860 // svg height - высота контейнера SVG
                  ,p = 30  // svg padding - отступы от границ SVG
                  ,x_max = 0 //x data max - максимальная координата x модельная
                  ,y_max = 0  //y data max - максимальная координата y модельная
                  ,xTicks = 20 // число зарубок на оси x
                  ,yTicks = 20 // число зарубок на оси y
  ,x_min =-20 // минимальная координата x модельная
  ,y_min =-20; // минимальная координата y модельная
```

Укажем данные для отрисовки траектории каждого игрока. В нашем случае это начальные условия, координаты обоих игроков в моменты переключения управления убегающего и момент поимки. Все эти точки мы нашли в третьем параграфе. В **values** – точки траектории убегающего, в **values1** – точки траектории преследователя.

```
var values = [{"x": -12, "y": -13}, {"x": -16, "y": -13}, {"x": -16, "y": -7},{"x": -15.5, "y": -6.5}];
var values1 = [{"x": 0,"y": 0}, {"x": -148/25, "y": -52/25}, {"x": -364/25, "y": -136/25},
{"x": -15.5, "y": -6.5}];
```

Создадим объекты – область рисования и оси, зададим размеры объектов и некоторые свойства, например, количество зарубок:

```
            var svg = d3 // опереденяем контейнер svg, в котором будем рисовать
                    .select("body")
                       .append("div") // добавляем в body новый div
                       .attr("class", "main") // добавляем css-класс main
                       .append("div") // в этот div - еще один div
                       .attr("class","sidebar") // для нового div - класс sidebar
                    .append("svg") // непосредственно сам svg
                    .attr("width", w) // задаем параметры для контейнера
                    .attr("height", h);

             var xScale = d3 // функцию пересчета из модельных координат в экранные
                    .scale
                    .linear() // преобразование линейное
                    .domain([x_min, x_max]) // из отрезка модельных координат
                    .range([p, w - p]); // в отрезок экранных координат

             var yScale = d3 // аналогично для y
                    .scale
                    .linear()
                    .domain([y_min, y_max])
                    .range([h - p, p]);
 var vis1 = svg.append("svg:g"); // рисуем сеточку на заднем фоне
 var rules1 = vis1.selectAll("g.rule")
.data(xScale.ticks(xTicks)) // здесь формируется список координат x
.enter().append("svg:g") // добавляем тег g, в котором будут верт. линии
.attr("class", "rule"); // добавляем к ним класс rule

rules1.append("svg:line") // добавляем Line (линии)
.attr("x1", xScale) // задаем координаты x начала
.attr("x2", xScale) // задаем координаты y начала
.attr("y1", p)
.attr("y2", h - p);

var vis2 = svg.append("svg:g"); // все то же самое, но для горизонтальных
var rules2 = vis2.selectAll("g.rule")
.data(yScale.ticks(yTicks))
.enter().append("svg:g")
.attr("class", "rule");

rules2.append("svg:line")
.attr("y1", yScale)
.attr("y2", yScale)
```

```
.attr("x1", p)
.attr("x2", w - p);
              var xAxis = d3 // рисуем оси
                      .svg
                      .axis()
                      .ticks(xTicks) // Зарубки
                      .scale(xScale); // масштаб осей

              var yAxis = d3 // тут для оси y
                      .svg
                      .axis()
                      .ticks(yTicks)
                      .orient("left")
                      .scale(yScale);
svg   // рисуем оси - переносим их
 .append("g")
 .attr("transform", "translate("+(0)+","+(p+(h-2*p)*(y_max-0)/(y_max-y_min))+")")
 .attr("class", "x axis")
 .call(xAxis);

svg
 .append("g")
 .attr("transform", "translate("+(p+(w-2*p)*(0-x_min)/(x_max-x_min))+","+(0)+")")
 .attr("class", "y axis")
 .call(yAxis);
```

Теперь у нас есть оси координат. Следующий фрагмент кода задает траекторию убегающего:

```
               var line = d3 // задаем <<функцию>> рисования линии
                       .svg
                       .line()
                       // d.x - координата x одного элемента данных, это модельная координата
                       .x(function(d) { return xScale(d.x); })
// а нам нужны экранные, поэтому мы применяем ранее опред. функцию пересчета
                       .y(function(d) { return yScale(d.y); })
// указывает, что линия рисуется как линейно-интерполированная между двумя точками
                       .interpolate("linear");

               var path = svg // отрисовываем линию
                       .append("path")
// в качестве данных (т.е. узлов интерполяции) передаем заполненные выше массивы
                       .attr("d", line(values))
                       .attr("stroke", "blue") // цвет линии голубой
                       .attr("stroke-width", 2) // толщина линии
                       .attr("fill", "none");

               var totalLength = path.node().getTotalLength();  // полная длина линии
     var dashLen = 10;     // darray для того, чтобы линия была пунктирной
     var ddLen = dashLen*2;
     var darray =dashLen;
     while(ddLen < totalLength){
         darray+=","+dashLen+","+dashLen;
         ddLen+=dashLen*2;}

               path   // зададим процесс плавной отрисовки линии
                   .attr("stroke-dasharray", darray + " " + totalLength)
                   .attr("stroke-dashoffset", totalLength)
                   .transition()
                   .duration(5000) // длительность отрисовки
                   .ease("linear")
                   .attr("stroke-dashoffset", 0);
```

Заметим, что траектория убегающего – пунктирная линия. Траектория преследователя:

```
            var path1 = svg
                      .append("path")
                      .attr("d", line(values1))
                      .attr("stroke", "green")
                      .attr("stroke-width", 2)
                      .attr("fill", "none");

               var totalLength1 = path1.node().getTotalLength();

               path1
                 .attr("stroke-dasharray", totalLength1 + "  " + totalLength1)
                 .attr("stroke-dashoffset", totalLength1)
                 .transition()
                 .duration(5000)
                 .ease("linear")
                 .attr("stroke-dashoffset", 0);
      var circles1 =  svg \\точки values отметим "кружочками"
.append("g")
                      .selectAll("circle")
                      .data(values)
                      .enter()
                      .append("circle");

               circles1
                      .attr("cx", function (d) { return xScale(d.x); })
                      .attr("cy", function (d) { return yScale(d.y); })
                      .attr("r", 3) //радиус точки - 3 пискела
                      .attr("fill", "red");

      var circles2 =  svg \\точки values1 отметим "кружочками"
                      .append("g")
                      .selectAll("circle")
                      .data(values1)
                      .enter()
                      .append("circle");

               circles2
                      .attr("cx", function (d) { return xScale(d.x); })
                      .attr("cy", function (d) { return yScale(d.y); })
                      .attr("r", 3)
                      .attr("fill", "purple");

          };
       </script>
   </head>
</html>
```

Внимательно проверим скрипт на наличие ошибок и сохраним файл (**F2**). Запустим скрипт (**enter**).

В окне браузера видим, что начальное положение убегающего E – точка с координатами $(-12, -13)$. Траектория игрока E задана пунктирной линией. Начальное положение преследователя P – точка с координатами $(0, 0)$. Траектория преследователя обозначена сплошной линией (см. рис. 18).

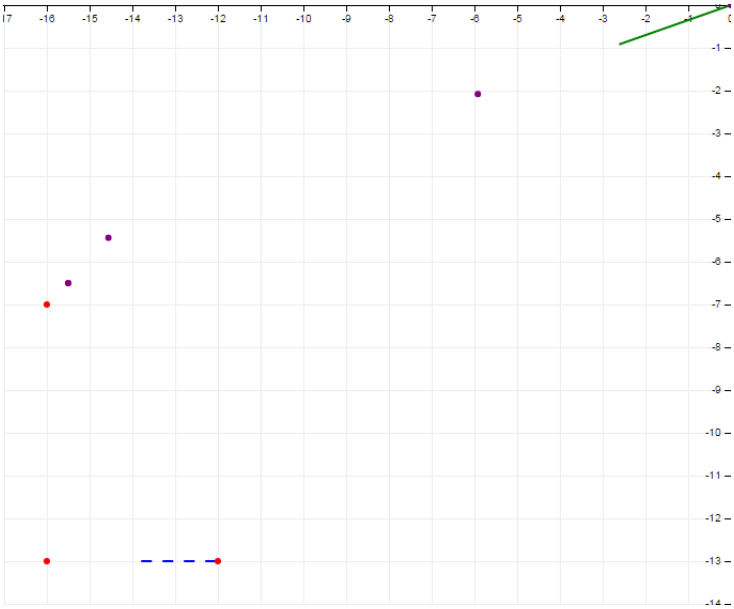


Рис. 18. Начало работы скрипта

В ходе игры происходит переключение управления два раза – траектория игроков меняет направление. Убегающий переключает управление в точках с координатами $(-16, -13)$ и $(-16, -7)$. Точки траектории, в которых происходит переключение, — небольшие кружки красного цвета.

Можно заметить, что преследователь использует стратегию параллельного сближения. Преследователь меняет направление одновременно с убегающим E. Скорость преследователя в два раза больше скорости убегающего. Если соединить прямой начальные положения игроков, то прямые, соединяющие траектории игроков в каждый момент времени, будут параллельны этой. Поимка происходит в точке пересечения траекторий игроков P и E.

По окончании отрисовки анимированного изображения, иллю-

стирующего ход дифференциальной игры, картинка на странице в браузере должна иметь следующий вид (рис. 19)

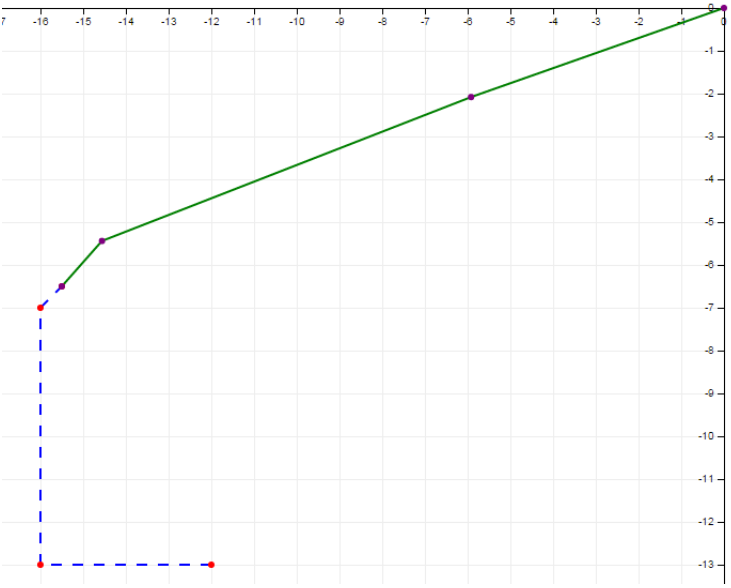


Рис. 19. Траектории игроков

ЗАДАНИЕ

Решите задачу поимки одного убегающего одним преследователем, поставленную в первом параграфе, исходя из новых исходных данных:

№ вар.	v_0	τ_1	v_1	τ_2	v_2	y_0
1	$(-2,0)$	1	$(0,2)$	2	$(1,1)$	$(0,10)$
2	$(-2,0)$	2	$(1,1)$	4	$(0,2)$	$(0,-10)$
3	$(0,2)$	2	$(-1,1)$	4	$(0,-2)$	$(11,12)$
4	$(0,2)$	2	$(-1,-1)$	4	$(0,-2)$	$(11,-12)$
5	$(0,2)$	2	$(1,-1)$	4	$(0,0)$	$(11,12)$
6	$(1,1)$	2	$(-2,0)$	4	$(2,0)$	$(12,13)$
7	$(-1,-1)$	2	$(2,0)$	4	$(-2,0)$	$(12,13)$
8	$(-1,1)$	2	$(-2,0)$	4	$(1,-1)$	$(12,13)$
9	$(1,-1)$	2	$(0,2)$	4	$(1,-1)$	$(12,13)$

Постройте с помощью библиотеки **D3** анимированную иллюстрацию заданной игры.

ПРИЛОЖЕНИЕ

Полный текст скрипта – визуализации игры простого преследования на плоскости – для заданных в третьем параграфе начальных условий и управлений.

```
<!DOCTYPE html>
<html>
    <head>
        <title>D3JS</title>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width">
        <script src="http://d3js.org/d3.v3.min.js" charset="utf-8"></script>
        <style>
          body {
            font: 10px sans-serif;
            margin: 0;
          }

          .axis {
            shape-rendering: crispEdges;
          }
          .x.axis line, .x.axis path  {
            fill: none;
            stroke: #000;
          }
          .y.axis line, .y.axis path  {
            fill: none;
            stroke: #000;
          }

.rule line {
  stroke: #eee;
  shape-rendering: crispEdges;
}
.main {
overflow:hidden;
}
.sidebar {
width:860px;
float:left;
}
.content{
    margin-top:30px;
margin-bottom:30px;
margin-right:860px;
```

```
    padding-left: 30px;
    box-sizing: border-box;
}
    .content img {
      width: 100%;
    }
        </style>
        <script>
            window.onload = function()
            {
              var w = 860 // svg width - ширина контейнера SVG
                  ,h = 860 // svg height - высота контейнера SVG
                  ,p = 30  // svg padding - отступы от границ SVG
                  ,x_max = 0 // x data max - макс коор-та x модельная
                  ,y_max = 0  // y data max - макс коор-та y модельная
                  ,xTicks = 20 // число зарубок на оси x
                  ,yTicks = 20 // число зарубок на оси y
  ,x_min =-20 // мин коор-та x модельная
  ,y_min =-20 // мин коор-та y модельная
           ;
              var values = [{"x": -12, "y": -13}, {"x": -16, "y": -13},
{"x": -16, "y": -7},{"x": -15.5, "y": -6.5}];
             var values1 = [{"x": 0, "y": 0}, {"x": -148/25, "y": -52/25},
            {"x": -364/25, "y": -136/25},{"x": -15.5, "y": -6.5}];

              var svg = d3
                      .select("body")
  .append("div")
  .attr("class", "main")
  .append("div")
  .attr("class","sidebar")
                      .append("svg")
                      .attr("width", w)
                      .attr("height", h);

              var xScale = d3
                      .scale
                      .linear()
                      .domain([x_min, x_max])
                      .range([p, w - p]);

              var yScale = d3
                      .scale
                      .linear()
                      .domain([y_min, y_max])
                      .range([h - p, p]);

 var vis1 = svg.append("svg:g");
```

```
  var rules1 = vis1.selectAll("g.rule")
.data(xScale.ticks(xTicks))
.enter().append("svg:g")
.attr("class", "rule");

rules1.append("svg:line")

.attr("x1", xScale)
.attr("x2", xScale)
.attr("y1", p)
.attr("y2", h - p);

var vis2 = svg.append("svg:g");
var rules2 = vis2.selectAll("g.rule")
.data(yScale.ticks(yTicks))
.enter().append("svg:g")
.attr("class", "rule");

rules2.append("svg:line")

.attr("y1", yScale)
.attr("y2", yScale)
.attr("x1", p)
.attr("x2", w - p);


              var xAxis = d3
                      .svg
                      .axis()
                      .ticks(xTicks)
                      .scale(xScale);

              var yAxis = d3
                      .svg
                      .axis()
                      .ticks(yTicks)
                      .orient("left")
                      .scale(yScale);

              svg
                .append("g")
                .attr("transform", "translate("+(0)+","
                +(p+(h-2*p)*(y_max-0)/(y_max-y_min))+")")
                .attr("class", "x axis")
                .call(xAxis)
              ;

              svg
                .append("g")
```

```
            .attr("transform", "translate("+
          (p+(w-2*p)*(0-x_min)/(x_max-x_min))+","+(0)+")")
            .attr("class", "y axis")
            .call(yAxis)
           ;

            var line = d3
                    .svg
                    .line()
                    .x(function(d) { return xScale(d.x); })
                    .y(function(d) { return yScale(d.y); })
                    .interpolate("linear");
            var path = svg
                    .append("path")
                    .attr("d", line(values))
                    .attr("stroke", "blue")
                    .attr("stroke-width", 2)
                    .attr("fill", "none")
                    ;

            var totalLength = path.node().getTotalLength();
 var dashLen = 10;
 var ddLen = dashLen*2;
 var darray =dashLen;
 while(ddLen < totalLength){
     darray+=","+dashLen+","+dashLen;
     ddLen+=dashLen*2;}

            path
                .attr("stroke-dasharray", darray + " " + totalLength)
     .attr("stroke-dashoffset", totalLength)
.transition()
                .duration(5000)                      .ease("linear")
                .attr("stroke-dashoffset", 0);

           var path1 = svg
                    .append("path")
                    .attr("d", line(values1))
                    .attr("stroke", "green")
                    .attr("stroke-width", 2)
                    .attr("fill", "none")
                    ;

            var totalLength1 = path1.node().getTotalLength();

            path1
              .attr("stroke-dasharray", totalLength1 + "  " + totalLength1)
              .attr("stroke-dashoffset", totalLength1)
              .transition()
```

```
                .duration(5000)
                .ease("linear")
                .attr("stroke-dashoffset", 0);
            var circles1 =  svg
.append("g")
                        .selectAll("circle")
                        .data(values)
                        .enter()
                        .append("circle");

                circles1
                        .attr("cx", function (d) { return xScale(d.x); })
                        .attr("cy", function (d) { return yScale(d.y); })
                        .attr("r", 3)
                        .attr("fill", "red");
 var circles2 =  svg
            .append("g")
                        .selectAll("circle")
                        .data(values1)
                        .enter()
                        .append("circle");

                circles2
                        .attr("cx", function (d) { return xScale(d.x); })
                        .attr("cy", function (d) { return yScale(d.y); })
                        .attr("r", 3)
                        .attr("fill", "purple");

            };

        </script>
    </head>
</html>
```

СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

1. *Айзекс Р.* Дифференциальные игры/ Р. Айзекс. – М.: Мир, 1967. – 479 с.

2. *Банников А. С.* Введение в дифференциальные игры/ А. С. Банников, Н.Н. Петров, Л.С. Чиркова. – Ижевск, 2013 – 48 с.

3. *Берж К.* Общая теория игр нескольких лиц/ К. Берж. – М.: ФМЛ, 1961. – 127 с.

4. *Благодатских А.И.* Конфликтное взаимодействие групп управляемых объектов/ А.И. Благодатских, Н.Н. Петров. – Ижевск.: Изд-во Удмуртс. ун-та, 2009. – 266 с.

5. *Благодатских В.И.* Введение в оптимальное управление / В.И. Благодатских. – М.: Высшая школа, 2001. – 239 с.

6. *Вайсборд Э.М.* Введение в дифферениицальные игры нескольких лиц и их приложения/ Э.М. Вайсборд, В.И. Жуковский. – М.: Сов. радио, 1980. – 304 с.

7. *Григоренко Н.Л.* Математические методы управления несколькими динамическими процессами/ Н.Л. Григоренко. – М.: Изд-во Московского ун-та, 1990. – 197 с.

8. *Жуковский В.И.* Линейно-квадратичные дифференциальные игры/ В.И. Жуковский, А.А. Чикрий. – Киев.: Наук. думка, 1994. – 320 с.

9. *Зак В.Л.* Задача уклонения от многих преследователей/ В. Л. Зак// ДАН СССР. 1982. Т. 265. N 5. С. 1051-1053.

10. *Иванов Р.П.* Простое преследование на компакте/Р.П. Иванов// ДАН СССР. – 1980. - Т. 254. - N 6. – С. 1318-1321.

11. *Красовский Н.Н.* Игровые задачи о встречи движений/ Н. Н. Красовский. – М.: Наука, 1970. – 420 с.

12. *Красовский Н.Н.* Альтернатива для игровой задачи сближения/ Н.Н. Красовский, А.И. Субботин// Прикладная математика и механика. – 1970. - Т. 34. - Вып. 6. – С. 1005-1022.

13. *Красовский Н.Н.* Позиционные дифференциальные игры/ Н.Н. Красовский, А.И. Субботин. – М.: Наука. 1974. – 456 с.

14. *Красовский Н.Н.* Управление динамической системой: задача о минимуме гарантированного результата/ Н.Н. Красовский. М.: Наука, 1985. – 518 с.

15. *Мезенцев А.В.* Дифференциальные игры с интегральными ограничениями/ А.В. Мезенцев. – М.: МГУ, 1988. – 135 с.

16. *Никольский М.С.* Первый прямой метод Л.С.Понтрягина в дифференциальных играх/ М.С. Никольский. – М.: МГУ, 1984. – 64 с.

17. *Оуэн Г.* Теория игр/ Г. Оуэн. – М.: Мир, 1971. – 230 с.

18. *Партхасаратхи Т.* Некоторые вопросы теории игр двух лиц/ Т. Партхасаратхи, Т. Рагхаван. – М.: Мир. 1974. – 259 с.

19. *Петров Н.Н.* Об управляемости автономных систем/ Н. Н. Петров //Дифференциальные уравнения. – 1968. - Т. 4. - N 4. – С. 606–617.

20. *Петров Н.Н.* Теория игр/Н.Н. Петров. – Ижевск: Изд-во Удмуртского ун-та, 1997. – 195 с.

21. *Петров Н.Н.* Введение в выпуклый анализ /Н.Н. Петров. – Ижевск: Изд-во Удмуртского ун-та, 2009. – 163 с.

22. *Петросян Л.А.* Дифференциальные игры преследования/Л.А. Петросян. – Л.: Изд-во ЛГУ, 1977. – 222 с.

23. *Петросян Л.А.* Теория игр/Л.А. Петросян, Н.А. Зенкевич, Е.А. Семина. – М.: Высшая школа, 1998. – 304 с.

24. *Петросян Л.А.* Геометрия простого преследования/Л.А. Петросян, Г.В. Томский. – Новосибирск: Наука, 1983. – 140 с.

25. *Петросян Л.А.* Игры поиска/ Л.А. Петросян, А.Ю. Гарнаев. СПб.: Изд-во Санкт-Петерб. ун-та, 1992. – 217 с.

26. *Петросян Л.А.* Преследование на плоскости/ Л.А. Петросян, Б. Б. Рихсиев. – М.: Наука, 1991. – 96 с.

27. *Понтрягин Л.С.* Избранные научные труды. В 3 т. Т.2/ Л. С. Понтрягин. – М.:Наука, 1988. – 342 с.

28. *Пшеничный Б.Н.* Дифференциальные игры/Б.Н. Пшеничный, В.В. Остапенко. – Киев.: Наукова думка, 1992. – 262 с.

29. *Рихсиев Б.Б.* Дифференциальные игры с простым движением/ Б.Б. Рихсиев. – Ташкент.: Фан, 1989. – 187 с.

30. *Рокафеллар Р.* Выпуклый анализ/Р. Рокафеллар. – М.: Мир, 1973. – 470 с.

31. *Сатимов Н. Ю.* Методы решения задачи уклонения от встречи в математической теории управления/Н.Ю. Сатимов, Б. Б. Рихсиев. – Ташкент.: Фан, 2000. – 176 с.

32. *Субботин А.И.* Оптимизация гарантии в задачах управления/ А.И. Субботин, А.Г. Ченцов. – М.: Наука, 1981. – 288 с.

33. *Ухоботов В.И.* Метод одномерного проектирования в линейных дифференциальных играх с интегральными ограничениями общего вида/ В.И. Ухоботов. – Челябинск. Изд-во Челябин. ун-та, 1998. – 78 с.

34. *Филиппов А. Ф.* О некоторых вопросах теории оптимального регулирования/А. Ф. Филиппов //Вестник МГУ. Серия Математика. механика. – 1959. - N 2. – С. 25-32.

35. *Черноусько Ф.Л.* Игровые задачи управления и поиска/ Ф. Л. Черноусько, А.А. Меликян. – М.: Наука, 1978. – 272 с.

36. *Чикрий А.А.* Конфликтно управляемые процессы/А.А. Чикрий. – Киев.: Наукова думка, 1992. – 380 с.

37. *Дунаев В.* Основы SVG [Электронный ресурс] URL: http://dunaevv1.narod.ru/mybooks/svg.pdf (дата обращения: 27.06.2015)

38. Введение в D3 [Электронный ресурс] URL: http://habrahabr.ru/company/datalaboratory/blog/217905/ (дата обращения: 30.06.2015)

39. *Mike Bostock* Мыслим связками[Электронный ресурс] URL: http://d3-js.ru/mike/join/ (дата обращения: 30.06.2015)

40. Введение в d3.js [Электронный ресурс] URL: http://frontender.info/vvedenie-v-djs/ (дата обращения: 12.09.2015)

41. Инструменты визуализации данных: D3.js [Электронный ресурс] URL: http://datareview.info/article/instrumentyi-vizualizatsii-dannyih-d3-js/ (дата обращения: 28.09.2015)

42. Объектная модель документа [Электронный ресурс] URL: http://mydocx.ru/3-7119.html (дата обращения: 30.09.2015)

Банников Александр Сергеевич
Чиркова Любовь Сергеевна

ВИЗУАЛИЗАЦИЯ ДИФФЕРЕНЦИАЛЬНЫХ ИГР ПРОСТОГО ПРЕСЛЕДОВАНИЯ

Авторская редакция

Подписано в печать 01.11.15. Формат $60 \times 84\ ^1/_{16}$.
Печать офсетная. Уч.- изд. л. 3,03. Усл. п. л. 2,79.
Тираж 10 экз. Заказ №

Издателельский центр «Удмуртский университет»
426034, г. Ижевск, ул. Университетская, 1, корп. 4.
Тел/факс: +7(3412)500-295, e-mail: editorial@udsu.ru

Типография Издателельского центра «Удмуртский университет»
426034, г. Ижевск, ул. Университетская, 1, корп. 2.