



Министерство образования и науки Российской Федерации  
Федеральное государственное бюджетное образовательное учреждение  
профессионального образования  
**Московский государственный технический университет имени Н.Э. Баумана**  
(национальный исследовательский университет)  
(МГТУ им. Н.Э. Баумана)

УТВЕРЖДАЮ  
Первый проректор-проректор по учебной работе

\_\_\_\_\_ Б.В. Падалкин

«\_\_\_» \_\_\_\_\_ 2022 г.

## **Методические указания**

### **ИТ каникул для школьников профильных классов образовательных организаций города Москвы**

#### **«БАУМАНСКАЯ ШКОЛА БУДУЩИХ ИНЖЕНЕРОВ - 8»**

**по инновационному инженерно-техническому направлению «Обучение  
интеллектуальных агентов решению задач на основе анализа больших  
данных» (21 аудиторный час)**

Автор: Терехов В.И., доцент кафедры «Системы обработки информации и управления»,  
к.т.н., зав. кафедрой

Москва, 2022 г.

# Введение

На занятиях мы будем разрабатывать браузерную игру "Пинг-понг" с возможностью сбора данных о действиях пользователя-игрока. Эти данные будут использоваться для обучения самостоятельной игре интеллектуального агента – нейронной сети.

Смысл игры — отбивать мячик, который летает по игровому полю. Для этого участник двигает свою платформу с помощью клавиш. Как только мячик вылетает за пределы поля, разыгрывается новый раунд.

Логика проекта по шагам:

1. Берём пустую HTML-страницу и рисуем на ней игровое поле. Там же делаем отдельный скрипт для игры.
2. В нём прописываем все переменные и константы, которые понадобятся.
3. Описываем отдельно платформы и мячик.
4. Делаем большой бесконечный цикл, где мы двигаем платформы и мячик. Это и будет наш игровой движок.
5. Не забываем следить за тем, какие клавиши нажаты, чтобы управлять платформой.

# Воркшоп 1

## Шаг 0. Страница с игровым полем

### Инструкции по выполнению шага

Начнем с того, что создадим папку, в которую мы будем размещать все файлы для нашего проекта. Например, создадим папку `ping-pong`. Перейдем в эту папку и будем в ней работать.

На начальном шаге создадим файл `index.html`. Это центральный файл, который будет связующим звеном для всех остальных вспомогательных скриптов и графических элементов. На данном этапе ограничимся простым заголовком страницы с помощью элемента `<title>`. Этот элемент указывается в элементе `<head>`, так как он не используется при отображении HTML страницы, а содержит лишь мета-информацию.

Вот так будет выглядеть наш `index.html` после добавления заголовка страницы:

```
/ping-pong/index.html
<!DOCTYPE html>
<html>

<head>
    <title>Ping-Pong JavaScript</title>
</head>

<body>
</body>

</html>
```

В элемент `<body>`, который содержит сам контент для отображения, добавим элемент холста `<canvas>`. Этот элемент является нововведением HTML5 и предназначен для создания растрового двухмерного изображения при помощи скриптов. Его мы и будем использовать как "экранчик", на котором будет отображаться весь игровой процесс. Размеры "экранчика" задаются через параметры `height` и `width`.

Наконец, необходимо подгрузить картинки, которые будут использованы при построении игровой сцены. Это можно сделать при помощи элемента `<img>` и его атрибута `src`.

Ко всем этим элементам хорошо бы добавить какой-то идентификатор, чтобы к ним было удобнее обращаться через скрипты в дальнейшем. Сделать это можно при помощи атрибута `id`.

Вот так будет выглядеть структура нашей папки, когда мы добавим картинки для игровой сцены:

```
ping-pong
├── index.html
├── ball.png
└── paddle.png
└── background.jpg
```

Вот как сейчас выглядит содержимое файла `index.html`:

```
/ping-pong/index.html
<!DOCTYPE html>
<html>

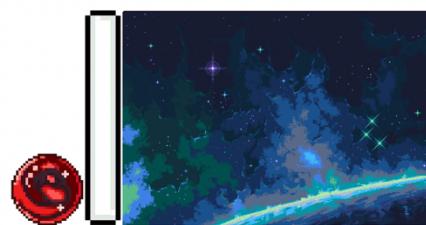
<head>
    <title>Ping-Pong JavaScript</title>
</head>

<body>
    <!-- Рисуем игровое поле -->
    <canvas width="750" height="585" id="game"></canvas>
    </img>
    </img>
    </img>
</body>

</html>
```

## Состояние игры к концу выполнения шага

Если открыть страницу `index.html` в браузере, то можно увидеть примерно следующий результат:



# Шаг 1. Добавление CSS

## Инструкции по выполнению шага

На данный момент страница выглядит не очень похожей на игру. Все картинки расположены бок о бок, при этом мячик выглядит каким-то уж совсем большим. А хоста `canvas` и вовсе нигде не видно. Да и белый фон как-то не очень сочетается с добавленными картинками.

Все эти проблемы можно решить с помощью каскадных таблиц стилей (Cascading Style Sheets, CSS). Главной задачей CSS является описание внешнего вида HTML страниц. При этом его можно использовать как напрямую внутри HTML с помощью элемента `<style>`, так и через вспомогательные внешние файлы, которые затем можно добавить на страницу при помощи элемента `<link>`. Такой способ позволяет структурировать кодовую базу, поэтому разобраться в нём и найти нужные фрагменты в дальнейшем будет проще.

Создадим папку `assets`, в которой будем хранить все вспомогательные файлы, отвечающие за стилистику игры. Перетащим в неё наши картинки и не забудем указать новое расположение в атрибутах `src`. Создадим файл `style.css`.

Вот так будет выглядеть наша папка проекта после создания файла `style.css`:

```
ping-pong
└── index.html
└── assets
    ├── ball.png
    ├── paddle.png
    ├── background.jpg
    └── style.css
```

Внесём в `style.css` него базовые настройки стиля нашей игры: цвет фона, выравнивание элементов по центру и масштаб. В дальнейшем будем добавлять в этот файл новые настройки отображения HTML-элементов по мере необходимости. В результате получим такой перечень стилей:

```
/ping-pong/assets/style.css
html,
body {
    height: 100%;
    margin: 0;
}

body {
    background: black;
    display: flex;
    align-items: center;
    justify-content: center;
}
```

После чего можем успешно привязать их к главному файлу с игрой `index.html` сразу после элемента `title`:

```
/ping-pong/index.html
```

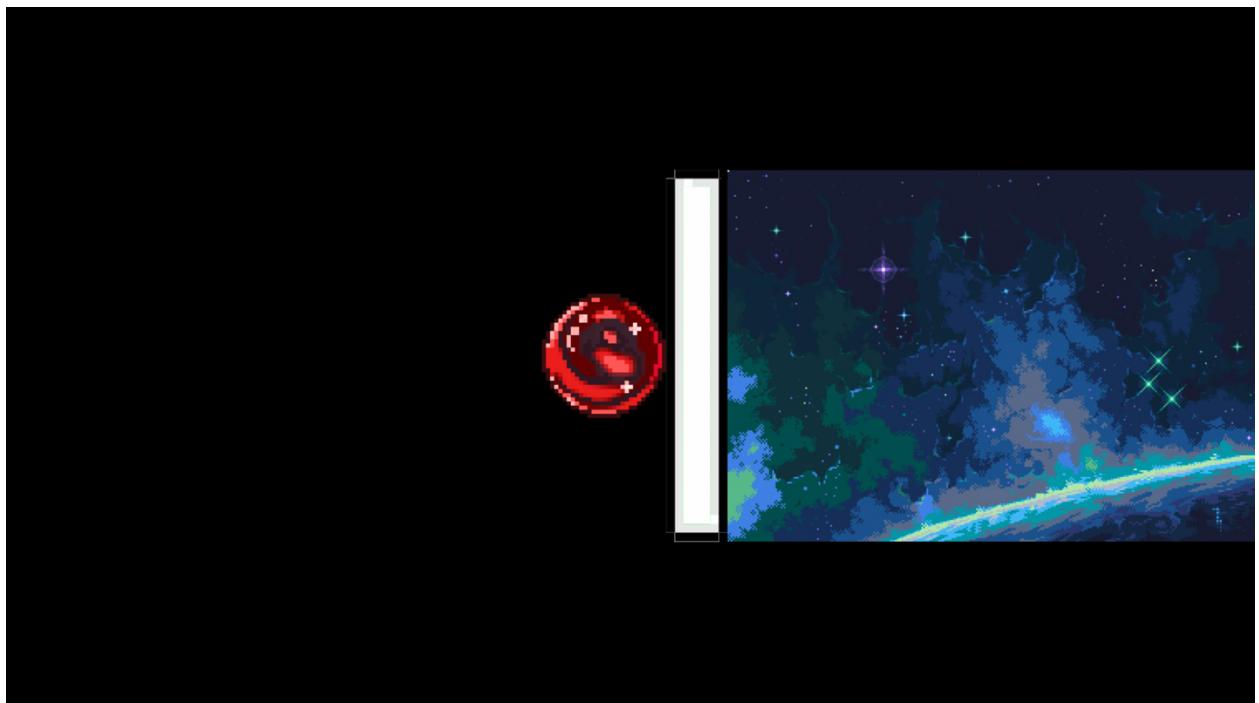
```
<!-- Начало места, которое мы изменяем -->

<head>
  <title>Ping-Pong JavaScript</title>
  <link rel="stylesheet" href="assets/style.css">
</head>

<!-- Конец редактирования -->
```

## Состояние игры к концу выполнения шага

Вот как теперь выглядит страница с игрой:



## Состояние файлов к концу выполнения шага

### index.html

В результате выполнения шага файл `index.html` будет выглядеть так:

```

/ping-pong/index.html
<!DOCTYPE html>
<html>

<head>
    <title>Ping-Pong JavaScript</title>
    <link rel="stylesheet" href="assets/style.css">
</head>

<body>
    <!-- Рисуем игровое поле -->
    <canvas width="750" height="585" id="game"></canvas>
    </img>
    </img>
    </img>
</body>

</html>

```

## Шаг 2. Рендеринг сцены игры

### Инструкции по выполнению шага

Пока что результат выглядит лишь отдалённо напоминающий игру. На этом шаге с помощью скриптом сделаем отрисовку игрового поля на нашем двухмерном холсте `<canvas>`. Для этого потребуется воспользоваться специальным скриптом, который будет получать доступ к элементам HTML страницы, считывать их и отображать в правильном порядке и масштабе.

Сделать это поможет язык JavaScript. Создадим в папке нашего проекта папку `js`, в которую будем складывать все скрипты. После чего добавим новый пустой файл `render.js`, который будет отвечать за отрисовку игрового поля.

Вот так будет выглядеть наша папка проекта после создания файла `render.js`:

```

ping-pong
├── index.html
└── assets
    ├── ball.png
    ├── paddle.png
    ├── background.jpg
    └── style.css
└── js
    └── render.js

```

Файл `render.js` необходимо добавить в качестве внешнего файла в `index.html` после элементов изображений с помощью элемента `<script>`:

```

/ping-pong/index.html
<!-- Начало места, которое мы изменяем -->
</img>
<script src="js/render.js"></script>
</body>

</html>
<!-- Конец редактирования -->

```

Прежде всего, необходимо получить доступ к элементам HTML страницы (холсту и изображениям). Сделать это можно по уникальному идентификатору, который мы заблаговременно указали на подготовительном [Шаге 0](#). Так, получить доступ к элементу холста можно при помощи вызова **команды** `document.getElementById("game")`. Получим все необходимые элементы (файл `render.js`):

```

/ping-pong/js/render.js
// Обращаемся к игровому полю из документа
const canvas = document.getElementById("game");

// Обращаемся к изображению ракетки из документа
const paddleImg = document.getElementById("paddle");

// Обращаемся к изображению мячика из документа
const ballImg = document.getElementById("ball");

// Обращаемся к изображению фона из документа
const backgroundImg = document.getElementById("background");

// Размер игровой клетки
const grid = 15;

// Высота платформы
const paddleHeight = grid * 5; // 80

// Задаём максимальное расстояние, на которое могут двигаться платформы
const LeftmaxPaddleY = canvas.height - grid - paddleHeight * 2;
const RightmaxPaddleY = canvas.height - grid - paddleHeight;

// Описываем левую платформу
const leftPaddle = {
    // Ставим её по центру
    x: grid * 2,
    y: 0,
    // Ширина — одна клетка
    width: grid,
    // Высоту берём из константы
    height: canvas.height, //paddleHeight * 2,
    // Платформа на старте никуда не движется
    dy: 0,
    paddleSpeed: 10
};
leftPaddle.dy = 0; //paddleSpeed;
// Описываем правую платформу
const rightPaddle = {
    // Ставим по центру с правой стороны
    x: canvas.width - grid * 3,
    y: canvas.height / 2 - paddleHeight / 2,
    // Задаём такую же ширину и высоту
}

```

```

width: grid,
height: paddleHeight,
// Правая платформа тоже пока никуда не двигается
dy: 0,
paddleSpeed: 10
};

var ballSpeed = 5;
// Описываем мячик
const ball = {
    // Он появляется в самом центре поля
    x: canvas.width / 2,
    y: canvas.height / 2,
    // квадратный, размером с клетку
    width: grid * 2,
    height: grid * 2,
    // На старте мяч пока не забит, поэтому убираем признак того, что мяч
    // нужно ввести в игру заново
    resetting: false,
    // Подаём мяч в правый верхний угол
    dx: ballSpeed,
    dy: -ballSpeed
};

```

Фигурными скобками мы демонстрируем, что создаём словарь – набор именованных полей объекта. Это достаточно удобный способ хранения сведений о свойствах того или иного объекта. Не нужно создавать множество переменных, а достаточно одной глобальной с вложенными полями-свойствами. В дальнейшем получать значения вложенных свойств можно либо через точку (`ball.x`), либо через квадратные скобки (`ball["x"]`).

Данная конфигурация мячика и ракеток позволит в ближайшее время максимально удобным способом отображать объекты в нужных точках игрового поля. Но перед этим предлагается отключить видимость элементов картинок, поскольку их содержимое и так будет отображаться на холсте. Это можно сделать с помощью последовательного обращения к свойствам `style`, а затем `display` и присваивания ему значения "none":

## INFO

Следующую часть кода добавляем в конец файла `render.js`.

```

/ping-pong/js/render.js
// Отключаем видимость элементов
paddleImg.style.display = "none";
ballImg.style.display = "none";
backgroundImg.style.display = "none";

```

Теперь при открытии страницы с игрой можно убедиться, что картинки пропали.

Наконец, необходимо расположить элементы на игровом поле. Чтобы это сделать требуется получить доступ к двухмерному представлению холста. Делается это при помощи команды `getContext("2d")`:

## INFO

Следующую часть кода добавляем в конец файла render.js.

```
/ping-pong/js/render.js
// Делаем поле двухмерным
const context = canvas.getContext("2d");
```

После чего производим следующие действия с двухмерным представлением холста:

## INFO

Следующую часть кода добавляем в конец файла render.js.

```
/ping-pong/js/render.js
// Рисуем содержимое заднего фона на холст
context.drawImage(backgroundImg, 0, 0, backgroundImg.width,
backgroundImg.height, 0, 0, canvas.width, canvas.height);

// Рисуем левую ракетку на холсте
context.drawImage(paddleImg, 0, 0, paddleImg.width, paddleImg.height,
leftPaddle.x, leftPaddle.y, leftPaddle.width, leftPaddle.height);

// Рисуем мячик
context.drawImage(ballImg, 0, 0, ballImg.width, ballImg.height, ball.x,
ball.y, ball.width, ball.height);

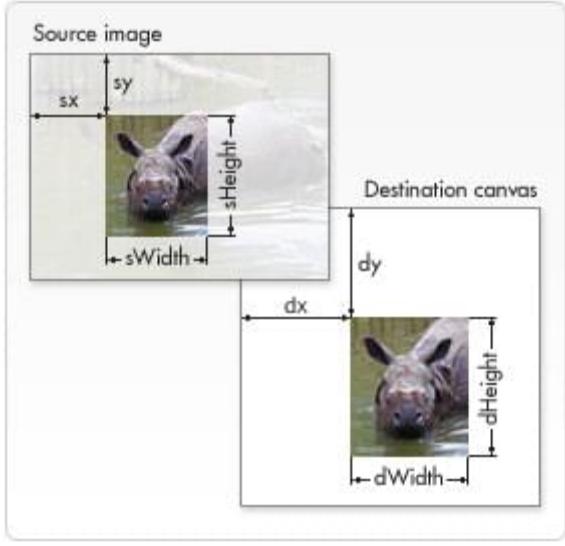
// Рисуем правую ракетку на холсте
context.drawImage(paddleImg, 0, 0, paddleImg.width, paddleImg.height,
rightPaddle.x, rightPaddle.y, rightPaddle.width, rightPaddle.height);

// Рисуем стены
context.fillStyle = "lightgrey";
context.fillRect(0, 0, canvas.width, grid);
context.fillRect(0, canvas.height - grid, canvas.width, canvas.height);

// Рисуем сетку посередине
for (let i = grid; i < canvas.height - grid; i += grid * 2) {
    context.fillRect(canvas.width / 2 - grid / 2, i, grid / 2, grid / 2);
}
```

Метод `clearRect(x, y, width, height)` очищает пространство холста в заданной прямоугольной области, делая его прозрачным. Стоит отметить, что координаты холста вдоль горизонтальной оси x растут слева направо, а вот вдоль вертикальной оси сверху вниз. Другими словами точке с координатами  $(0,0)$  соответствует верхний левый угол холста.

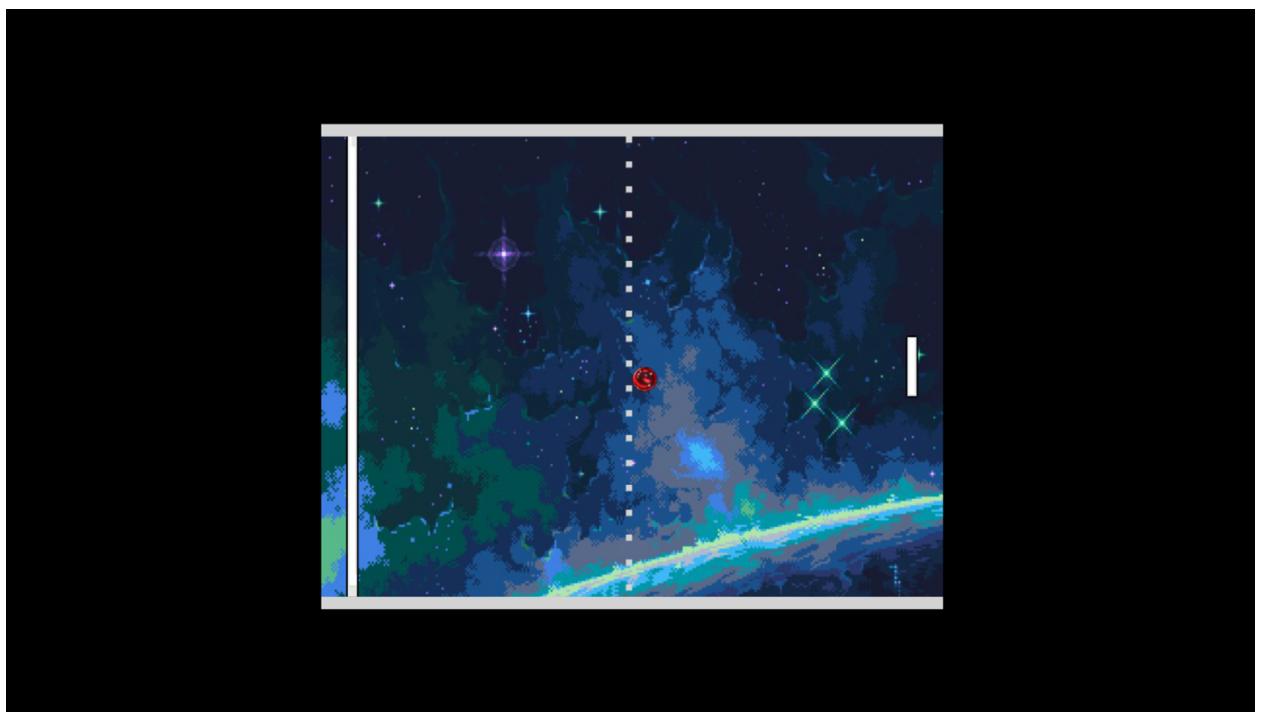
Метод `drawImage(image, sx, sy, sWidth, sHeight, dx, dy, dWidth, dHeight)` позволяет перемещать фрагмент элемента `<img>` в заданное место холста:



Метод `fillStyle` позволяет выбрать цвет для заливки геометрических фигур, а метод `fillRect(x, y, width, height)` рисует прямоугольники с заданными координатами на холсте.

## Состояние игры к концу выполнения шага

В результате проделанной работы внешний вид страницы содержит игровое поле с правильно расположенными игровыми объектами:



# Состояние файлов к концу выполнения шага

## render.js

Вот так будет выглядеть наш файл `render.js` в результате проделанной работы:

```
/ping-pong/js/render.js
// Обращаемся к игровому полю из документа
const canvas = document.getElementById("game");

// Обращаемся к изображению ракетки из документа
const paddleImg = document.getElementById("paddle");

// Обращаемся к изображению мячика из документа
const ballImg = document.getElementById("ball");

// Обращаемся к изображению фона из документа
const backgroundImg = document.getElementById("background");

// Размер игровой клетки
const grid = 15;

// Высота платформы
const paddleHeight = grid * 5; // 80

// Задаём максимальное расстояние, на которое могут двигаться платформы
const LeftmaxPaddleY = canvas.height - grid - paddleHeight * 2;
const RightmaxPaddleY = canvas.height - grid - paddleHeight;

// Описываем левую платформу
const leftPaddle = {
    // Ставим её по центру
    x: grid * 2,
    y: 0,
    // Ширина – одна клетка
    width: grid,
    // Высоту берём из константы
    height: canvas.height, //paddleHeight * 2,
    // Платформа на старте никуда не движется
    dy: 0,
    paddleSpeed: 10
};
leftPaddle.dy = 0; //paddleSpeed;
// Описываем правую платформу
const rightPaddle = {
    // Ставим по центру с правой стороны
    x: canvas.width - grid * 3,
    y: canvas.height / 2 - paddleHeight / 2,
    // Задаём такую же ширину и высоту
    width: grid,
    height: paddleHeight,
    // Правая платформа тоже пока никуда не двигается
    dy: 0,
    paddleSpeed: 10
}
```

```

};

var ballSpeed = 5;
// Описываем мячик
const ball = {
    // Он появляется в самом центре поля
    x: canvas.width / 2,
    y: canvas.height / 2,
    // квадратный, размером с клетку
    width: grid * 2,
    height: grid * 2,
    // На старте мяч пока не забит, поэтому убираем признак того, что мяч
    // нужно ввести в игру заново
    resetting: false,
    // Подаем мяч в правый верхний угол
    dx: ballSpeed,
    dy: -ballSpeed
};

// Отключаем видимость элементов
paddleImg.style.display = "none";
ballImg.style.display = "none";
backgroundImg.style.display = "none";

// Делаем поле двухмерным
const context = canvas.getContext("2d");

// Рисуем содержимое заднего фона на холст
context.drawImage(backgroundImg, 0, 0, backgroundImg.width,
backgroundImg.height, 0, 0, canvas.width, canvas.height);

// Рисуем левую ракетку на холсте
context.drawImage(paddleImg, 0, 0, paddleImg.width, paddleImg.height,
leftPaddle.x, leftPaddle.y, leftPaddle.width, leftPaddle.height);

// Рисуем мячик
context.drawImage(ballImg, 0, 0, ballImg.width, ballImg.height, ball.x,
ball.y, ball.width, ball.height);

// Рисуем правую ракетку на холсте
context.drawImage(paddleImg, 0, 0, paddleImg.width, paddleImg.height,
rightPaddle.x, rightPaddle.y, rightPaddle.width, rightPaddle.height);

// Рисуем стены
context.fillStyle = "lightgrey";
context.fillRect(0, 0, canvas.width, grid);
context.fillRect(0, canvas.height - grid, canvas.width, canvas.height);

// Рисуем сетку посередине
for (let i = grid; i < canvas.height - grid; i += grid * 2) {
    context.fillRect(canvas.width / 2 - grid / 2, i, grid / 2, grid / 2);
}

```

## Шаг 3. Оборачивание рендеринга в функцию

# Инструкции по выполнению шага

Поскольку в дальнейшем игровое поле нужно будет перерисовывать достаточно часто, то хорошо бы обернуть весь код, который отвечает за компоновку игровых элементов на сцене, в отдельную функцию, которую затем можно будет удобно вызывать. На языке JavaScript существует несколько способов задания функций. Следующие примеры создают одну и ту же функцию сложения двух чисел:

```
// Изначальный синтаксис объявления функций
function add(a,b) {
    return a+b;
};

// Функция-выражение
var add = function (a,b) {
    return a+b;
};

// Объявление функции через конструктор
const add = new Function('a', 'b', 'return a + b');

// Стрелочная функция, стандартная запись
let add = (a, b) => {
    return a+b;
};

// Стрелочная функция, краткая запись
let add = (a, b) => (a+b);
```

Как можно увидеть, JavaScript предоставляет большую гибкость при объявлении и описании функций. Хорошей практикой является использование одного типа объявления функций в своём проекте. Очень редко используют два типа объявлений (некоторые объявления отличаются по логике работы). При разработке будем использовать по возможности объявления в виде стрелочных функций в стандартной форме. Объявим стрелочную функцию `redraw` без параметров, которая будет перерисовывать сцену игры каждый раз при вызове:

## INFO

Следующая часть кода переписывает часть кода в `render.js` после строчки `const context = canvas.getContext("2d");` до конца файла.

```
/ping-pong/js/render.js
const redraw = () => {
    // Очищаем холст от предыдущего кадра
    context.clearRect(0, 0, canvas.width, canvas.height);

    // Рисуем содержимое заднего фона на холст
    context.drawImage(backgroundImg, 0, 0, backgroundImg.width,
backgroundImg.height, 0, 0, canvas.width, canvas.height);

    // Рисуем левую ракетку на холсте
```

```

    context.drawImage(paddleImg, 0, 0, paddleImg.width, paddleImg.height,
leftPaddle.x, leftPaddle.y, leftPaddle.width, leftPaddle.height);

    // Рисуем мячик
    context.drawImage(ballImg, 0, 0, ballImg.width, ballImg.height, ball.x,
ball.y, ball.width, ball.height);

    // Рисуем правую ракетку на холсте
    context.drawImage(paddleImg, 0, 0, paddleImg.width, paddleImg.height,
rightPaddle.x, rightPaddle.y, rightPaddle.width, rightPaddle.height);

    // Рисуем стены
    context.fillStyle = "lightgrey";
    context.fillRect(0, 0, canvas.width, grid);
    context.fillRect(0, canvas.height - grid, canvas.width, canvas.height);

    // Рисуем сетку посередине
    for (let i = grid; i < canvas.height - grid; i += grid * 2) {
        context.fillRect(canvas.width / 2 - grid / 2, i, grid / 2, grid / 2);
    };
}

```

Теперь для того чтобы перерисовать игровую сцену, достаточно будет написать `redraw()` в любом месте кода. Что и предлагается сделать после описания функции:

#### INFO

Следующую часть кода добавляем в конец файла `render.js`.

```
/ping-pong/js/render.js
redraw();
```

Внешний вид страницы должен остаться прежним (в случае отсутствия некоторых элементов игры попробуйте обновить страницу).

## Состояние файлов к концу выполнения шага

### **render.js**

Так будет выглядеть наш файл `render.js` в результате проделанной работы:

```
/ping-pong/js/render.js
// Обращаемся к игровому полю из документа
const canvas = document.getElementById("game");

// Обращаемся к изображению ракетки из документа
const paddleImg = document.getElementById("paddle");
```

```
// Обращаемся к изображению мячика из документа
const ballImg = document.getElementById("ball");

// Обращаемся к изображению фона из документа
const backgroundImg = document.getElementById("background");

// Размер игровой клетки
const grid = 15;

// Высота платформы
const paddleHeight = grid * 5; // 80

// Задаём максимальное расстояние, на которое могут двигаться платформы
const LeftmaxPaddleY = canvas.height - grid - paddleHeight * 2;
const RightmaxPaddleY = canvas.height - grid - paddleHeight;

// Описываем левую платформу
const leftPaddle = {
    // Ставим её по центру
    x: grid * 2,
    y: 0,
    // Ширина – одна клетка
    width: grid,
    // Высоту берём из константы
    height: canvas.height, //paddleHeight * 2,
    // Платформа на старте никуда не движется
    dy: 0,
    paddleSpeed: 10
};
leftPaddle.dy = 0; //paddleSpeed;
// Описываем правую платформу
const rightPaddle = {
    // Ставим по центру с правой стороны
    x: canvas.width - grid * 3,
    y: canvas.height / 2 - paddleHeight / 2,
    // Задаём такую же ширину и высоту
    width: grid,
    height: paddleHeight,
    // Правая платформа тоже пока никуда не двигается
    dy: 0,
    paddleSpeed: 10
};
var ballSpeed = 5;
// Описываем мячик
const ball = {
    // Он появляется в самом центре поля
    x: canvas.width / 2,
    y: canvas.height / 2,
    // квадратный, размером с клетку
    width: grid * 2,
    height: grid * 2,
    // На старте мяч пока не забит, поэтому убираем признак того, что мяч
    // нужно ввести в игру заново
    resetting: false,
    // Подаем мяч в правый верхний угол
    dx: ballSpeed,
    dy: -ballSpeed
};

// Отключаем видимость элементов
paddleImg.style.display = "none";
ballImg.style.display = "none";
backgroundImg.style.display = "none";
```

```
// Делаем поле двухмерным
const context = canvas.getContext("2d");

const redraw = () => {
    // Очищаем холст от предыдущего кадра
    context.clearRect(0, 0, canvas.width, canvas.height);

    // Рисуем содержимое заднего фона на холст
    context.drawImage(backgroundImg, 0, 0, backgroundImg.width,
backgroundImg.height, 0, 0, canvas.width, canvas.height);

    // Рисуем левую ракетку на холсте
    context.drawImage(paddleImg, 0, 0, paddleImg.width, paddleImg.height,
leftPaddle.x, leftPaddle.y, leftPaddle.width, leftPaddle.height);

    // Рисуем мячик
    context.drawImage(ballImg, 0, 0, ballImg.width, ballImg.height, ball.x,
ball.y, ball.width, ball.height);

    // Рисуем правую ракетку на холсте
    context.drawImage(paddleImg, 0, 0, paddleImg.width, paddleImg.height,
rightPaddle.x, rightPaddle.y, rightPaddle.width, rightPaddle.height);

    // Рисуем стены
    context.fillStyle = "lightgrey";
    context.fillRect(0, 0, canvas.width, grid);
    context.fillRect(0, canvas.height - grid, canvas.width, canvas.height);

    // Рисуем сетку посередине
    for (let i = grid; i < canvas.height - grid; i += grid * 2) {
        context.fillRect(canvas.width / 2 - grid / 2, i, grid / 2, grid / 2);
    };
};

redraw();
```

# Воркшоп 2

## Шаг 4. Написание движка игры

### Инструкции по выполнению шага

Теперь, когда функция отрисовки игрового поля вместе с описанием объектов мячика и ракетки у нас имеется, можно заняться описанием логики игры. Каждый момент времени в игре пинг-понга необходимо проделывать следующие вещи:

- очистить игровое поле от всего;
- проследить за тем, чтобы ничего при движении не ушло за границы игрового поля;
- перезапустить мяч из центра, если он всё-таки вылетел за платформу;
- если игрок успел подставить платформу — сделать отскок мяча;
- отрисовать игровое поле.

Всю механику игру опишем в отдельном скрипте JavaScript `engine.js`. Не забудем указать новоиспечённый файл в главном файле игры — `index.html`:

```
/ping-pong/index.html
<!-- Начало места, которое мы изменяем -->
</img>
<script src="js/render.js"></script>
<script src="js/engine.js"></script>
</body>

</html>
<!-- Конец редактирования -->
```

Вот так будет выглядеть наша папка проекта после создания файла `engine.js`:

```
ping-pong
├── index.html
└── assets
    ├── ball.png
    ├── paddle.png
    ├── background.jpg
    └── style.css
└── js
    ├── render.js
    └── engine.js
```

## Обнаружение столкновения

Центральной механикой игры является движение мячика и его взаимодействие с другими объектами. Поэтому прежде всего нам понадобится функция, которая будет проверять столкновения мячика с платформами. Для этого возьмём готовую функцию из интернета. Так как она написана под лицензией Creative Commons Attribution-ShareAlike license, то мы добавим в проект ссылку на оригинальную статью:

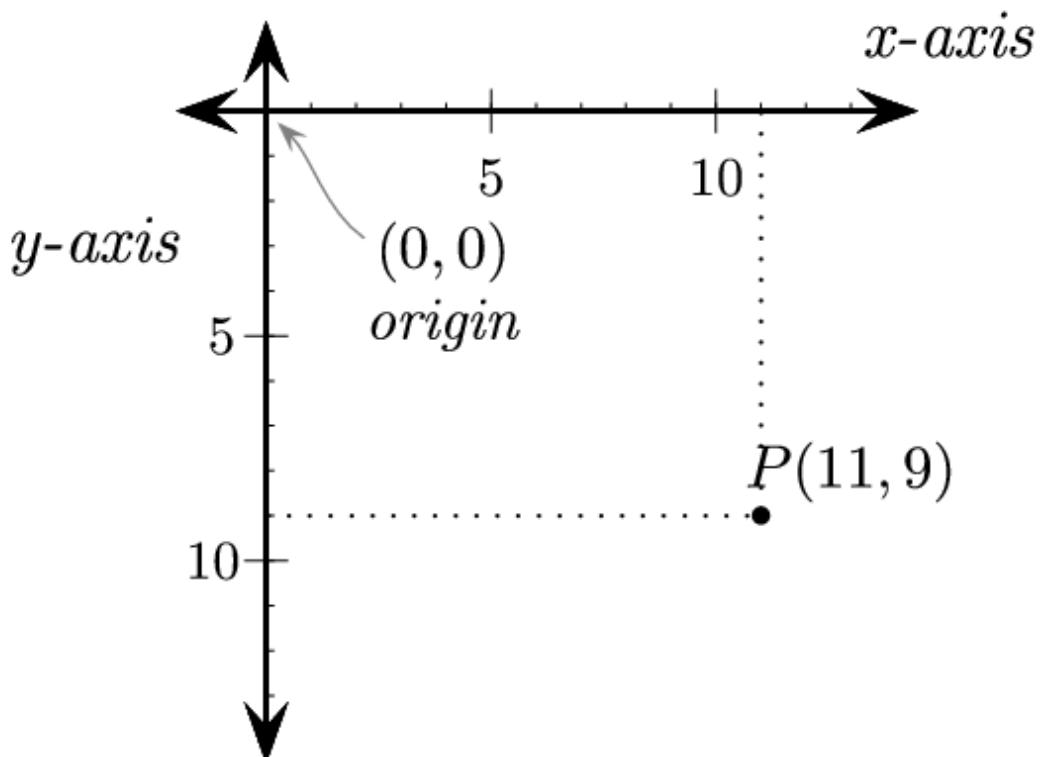
### INFO

Следующую часть кода добавляем в начало файла `engine.js`.

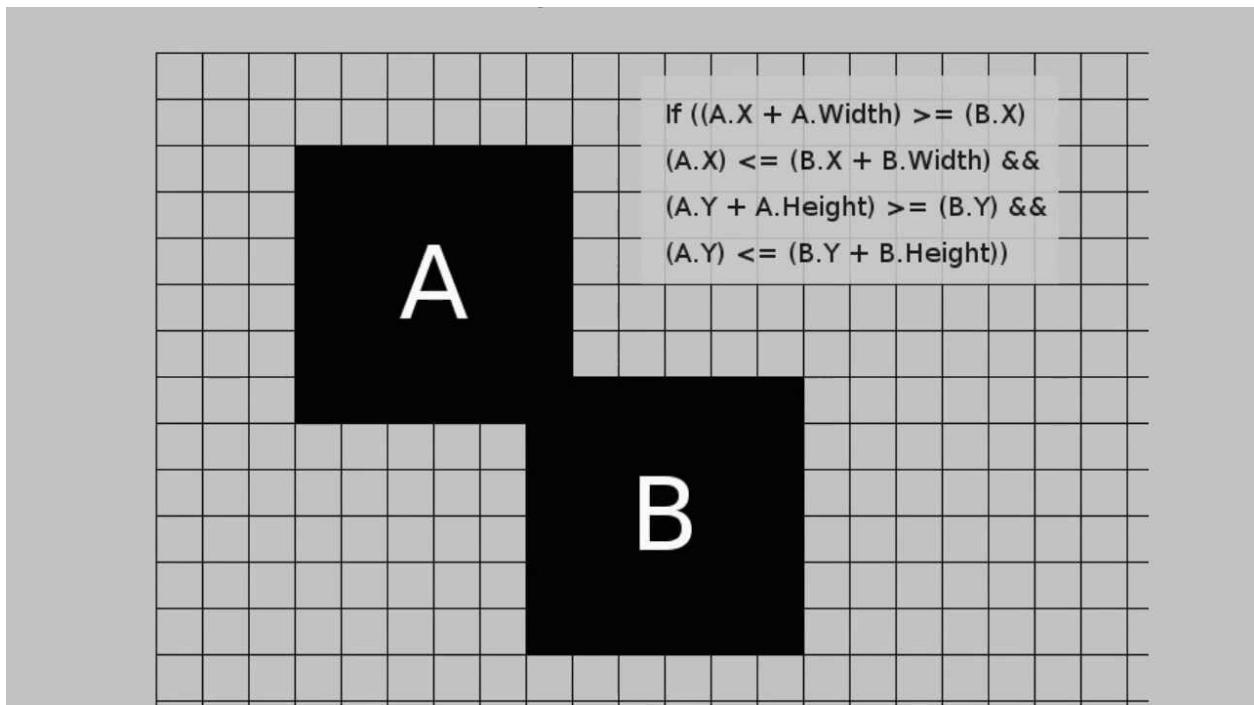
`/ping-pong/js/engine.js`

```
// Проверка на то, пересекаются два объекта с известными координатами или нет
// Подробнее тут: https://developer.mozilla.org/en-US/docs/Games/Techniques/2D_collision_detection
const collides = (obj1, obj2) => {
    return (
        obj1.x < obj2.x + obj2.width &&
        obj1.x + obj1.width > obj2.x &&
        obj1.y < obj2.y + obj2.height &&
        obj1.y + obj1.height > obj2.y
    );
};
```

Давайте взглянем на основы `canvas`, используемые в программировании игр. Холст представляет собой сетку из пикселей. Когда мы используем холст для рисования 2D-изображений, каждый пиксель имеет местоположение `x` и `y`, которое описывает, где он находится на холсте:



Верхнеуровнево функция `collides` проверяет наличие пересечения между двумя фигурами как на рисунке ниже:



## Создание основного игрового цикла

Теперь можно создать игровой цикл, в котором будем обрабатывать все события игры:

### INFO

Следующую часть кода добавляем в конец файла `engine.js`.

```
/ping-pong/js/engine.js
// Главный цикл игры
const loop = () => {
    // Логика
    // ...

    // Отрисовываем новый кадр
    redraw();

    // Логирование
    console.log('!');

    // Рекурсивный вызов игрового движка
    requestAnimationFrame(loop);
};
```

Цикл представим в виде функции без аргументов, которая вызывает в конце саму себя с ограничением на частоту вызова при помощи метода `requestAnimationFrame`. Таким образом, функция будет бесконечно вызывать саму себя снова и снова. Альтернативно можно было написать бесконечный цикл `while`. Логирование можно выполнить с помощью

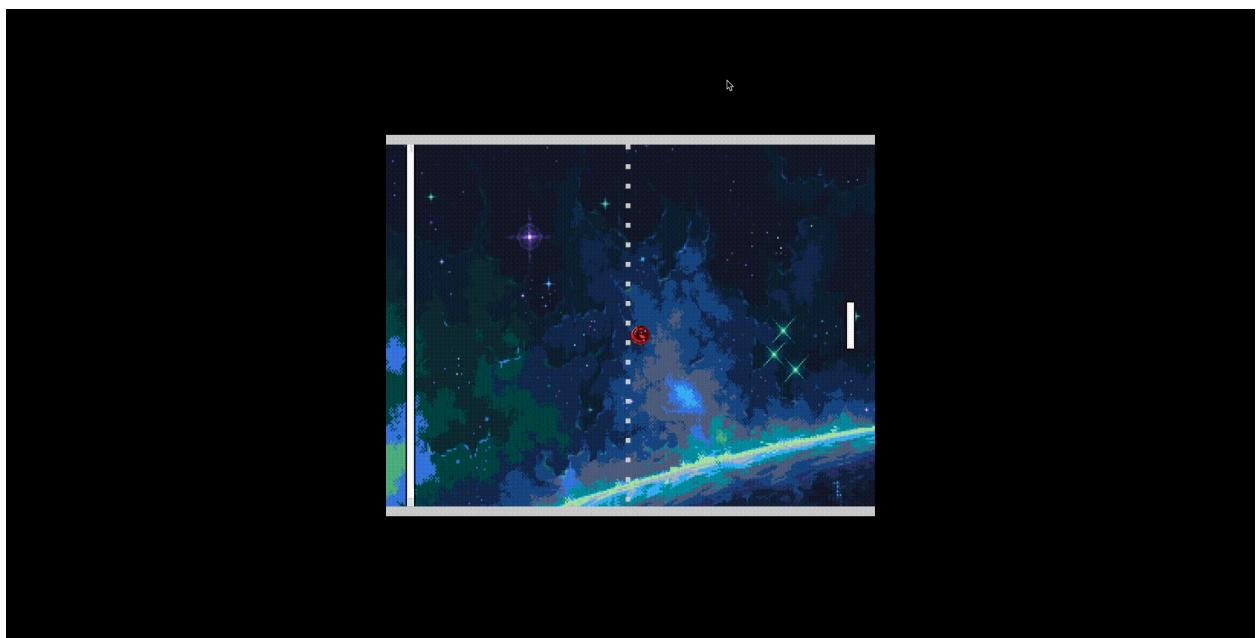
команды `console.log(TEXT)`. Это позволит производить отладку программы более эффективно. Проверим работоспособность игрового цикла. Добавим в конце файла `engine.js` следующие строчки:

#### INFO

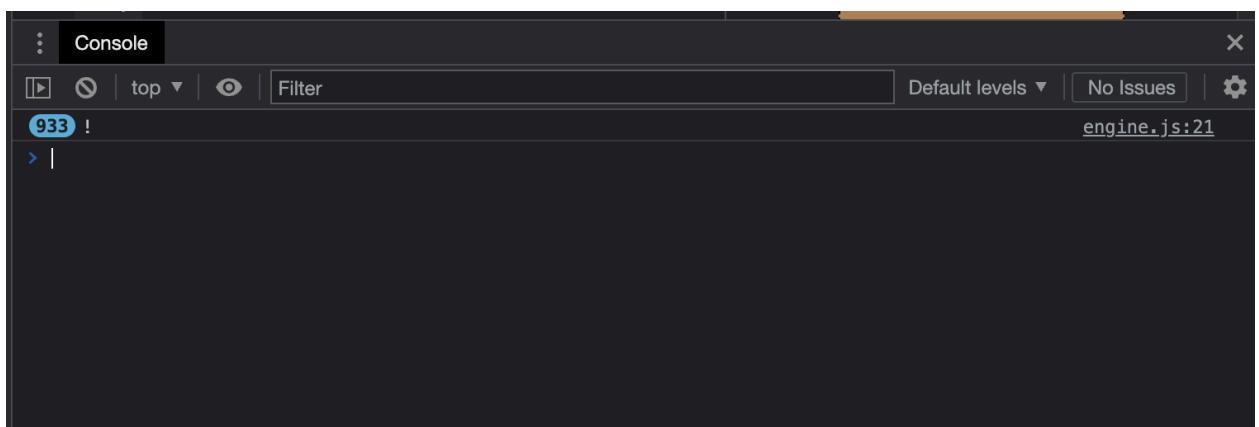
Следующую часть кода добавляем в конец файла `engine.js`.

```
/ping-pong/js/engine.js
// Запускаем игру
requestAnimationFrame(loop);
```

После чего откроем главную страницу игры и откройте режим разработчика (в Google Chrome и Firefox это можно сделать при помощи горячей клавиши F12). Затем выберите вкладку `Console` ИЛИ Консоль.



Если вы всё сделали правильно, то на экране должен появиться восклицательный знак и постоянно растущее рядом с ним число:



## Движение мячика

Когда работоспособность рекурсивного цикла была успешно проверена, можем перейти к написанию логики игры. Что нам нужно сделать в следующий момент игры? Сместить движущиеся предметы на следующий шаг и проверить, не вылезли ли они за границы поля. Только после этого можно вызывать написанную на [Шаге 3](#) функцию отрисовки `redraw()`. Код обновления положения объектов выглядит следующим образом:

### INFO

Следующая часть кода изменяет функцию `loop` в `engine.js`.

```
/ping-pong/js/engine.js
const loop = () => {
    // Если платформы на предыдущем шаге куда-то двигались — пусть продолжают
    // двигаться
    rightPaddle.y += rightPaddle.dy;

    // Если правая платформа пытается вылезти за игровое поле вниз,
    if (rightPaddle.y < grid) {
        // то оставляем её на месте
        rightPaddle.y = grid;
    }
    // Проверяем то же самое сверху
    else if (rightPaddle.y > RightmaxPaddleY) {
        rightPaddle.y = RightmaxPaddleY;
    }

    // Если мяч на предыдущем шаге куда-то двигался — пусть продолжает
    // двигаться
    ball.x += ball.dx;
    ball.y += ball.dy;
    // Если мяч касается стены снизу — меняем направление по оси Y на
    // противоположное
    if (ball.y < grid) {
        ball.y = grid;
        ball.dy *= -1;
    }
    // Делаем то же самое, если мяч касается стены сверху
    else if (ball.y + grid > canvas.height - grid) {
        ball.y = canvas.height - grid * 2;
        ball.dy *= -1;
    }

    // Отрисовываем новый кадр
    redraw();

    // Рекурсивный вызов игрового движка
    requestAnimationFrame(loop);
};
```

Возвращаем мяч в игру, если он улетел

Всё, что нам для этого понадобится, — проверить, выходят ли координаты мяча за координаты границ поля. Если да — ставим мяч по центру и даём игроку секунду на подготовку.

## INFO

Следующая часть кода изменяет функцию `loop` в `engine.js`.

```
/ping-pong/js/engine.js
// Главный цикл игры
const loop = () => {
    // Если платформы на предыдущем шаге куда-то двигались — пусть продолжают
    // двигаться
    rightPaddle.y += rightPaddle.dy;

    // Если правая платформа пытается вылезти за игровое поле вниз,
    if (rightPaddle.y < grid) {
        // то оставляем её на месте
        rightPaddle.y = grid;
    }
    // Проверяем то же самое сверху
    else if (rightPaddle.y > RightmaxPaddleY) {
        rightPaddle.y = RightmaxPaddleY;
    }

    // Если мяч на предыдущем шаге куда-то двигался — пусть продолжает
    // двигаться
    ball.x += ball.dx;
    ball.y += ball.dy;
    // Если мяч касается стены снизу — меняем направление по оси Y на
    // противоположное
    if (ball.y < grid) {
        ball.y = grid;
        ball.dy *= -1;
    }
    // Делаем то же самое, если мяч касается стены сверху
    else if (ball.y + grid > canvas.height - grid) {
        ball.y = canvas.height - grid * 2;
        ball.dy *= -1;
    }
    // Если мяч улетел за игровое поле влево или вправо — перезапускаем его
    if ((ball.x < 0 || ball.x > canvas.width) && !ball.resetting) {
        // Помечаем, что мяч перезапущен, чтобы не зациклился
        ball.resetting = true;
        // Даём секунду на подготовку игрокам
        setTimeout(() => {
            // Всё, мяч в игре
            ball.resetting = false;
            // Снова запускаем его из центра
            ball.x = canvas.width / 2;
            ball.y = canvas.height / 2;
            rightPaddle.x = canvas.width - grid * 3;
            rightPaddle.y = canvas.height / 2 - paddleHeight / 2;
        }, 1000);
    }

    // Отрисовываем новый кадр
    redraw();

    // Рекурсивный вызов игрового движка
    requestAnimationFrame(loop);
};


```

Как можно догадаться, метод, который даёт игроку секунду на подготовку называется `setTimeout(callback, timeOut)`. Как только пройдёт `timeOut`

миллисекунд (1000 мс = 1с), то вызывается так называемая функция обратного вызова или callback. В данном случае, мы задаёс её как анонимную стрелочную функцию без имени.

### Если игрок успел отбить мяч

Используем функцию `collides`, о которой мы говорили раньше, — она проверяет, есть на пути мяча препятствие или нет. Если есть — меняем направление движения мячика:

```
/ping-pong/js/engine.js
// Главный цикл игры
const loop = () => {
    // Если платформы на предыдущем шаге куда-то двигались — пусть продолжают
    // двигаться
    rightPaddle.y += rightPaddle.dy;

    // Если правая платформа пытается вылезти за игровое поле вниз,
    if (rightPaddle.y < grid) {
        // то оставляем её на месте
        rightPaddle.y = grid;
    }
    // Проверяем то же самое сверху
    else if (rightPaddle.y > RightmaxPaddleY) {
        rightPaddle.y = RightmaxPaddleY;
    }

    // Если мяч на предыдущем шаге куда-то двигался — пусть продолжает
    // двигаться
    ball.x += ball.dx;
    ball.y += ball.dy;
    // Если мяч касается стены снизу — меняем направление по оси Y на
    // противоположное
    if (ball.y < grid) {
        ball.y = grid;
        ball.dy *= -1;
    }
    // Делаем то же самое, если мяч касается стены сверху
    else if (ball.y + grid > canvas.height - grid) {
        ball.y = canvas.height - grid * 2;
        ball.dy *= -1;
    }
    // Если мяч улетел за игровое поле влево или вправо — перезапускаем его
    if ((ball.x < 0 || ball.x > canvas.width) && !ball.resetting) {
        // Помечаем, что мяч перезапущен, чтобы не зациклился
        ball.resetting = true;
        // Даём секунду на подготовку игрокам
        setTimeout(() => {
            // Всё, мяч в игре
            ball.resetting = false;
            // Снова запускаем его из центра
            ball.x = canvas.width / 2;
            ball.y = canvas.height / 2;
            rightPaddle.x = canvas.width - grid * 3;
            rightPaddle.y = canvas.height / 2 - paddleHeight / 2;
        }, 1000);
    }

    // Если мяч коснулся левой платформы,
    if (collides(ball, leftPaddle)) {
```

```

    // то отправляем его в обратном направлении
    ball.dx *= -1;
    // Увеличиваем координаты мяча на ширину платформы, чтобы не
    засчитался новый отскок
    ball.x = leftPaddle.x + leftPaddle.width;
}
// Проверяем и делаем то же самое для правой платформы
else if (collides(ball, rightPaddle)) {
    ball.dx *= -1;
    ball.x = rightPaddle.x - ball.width;
}

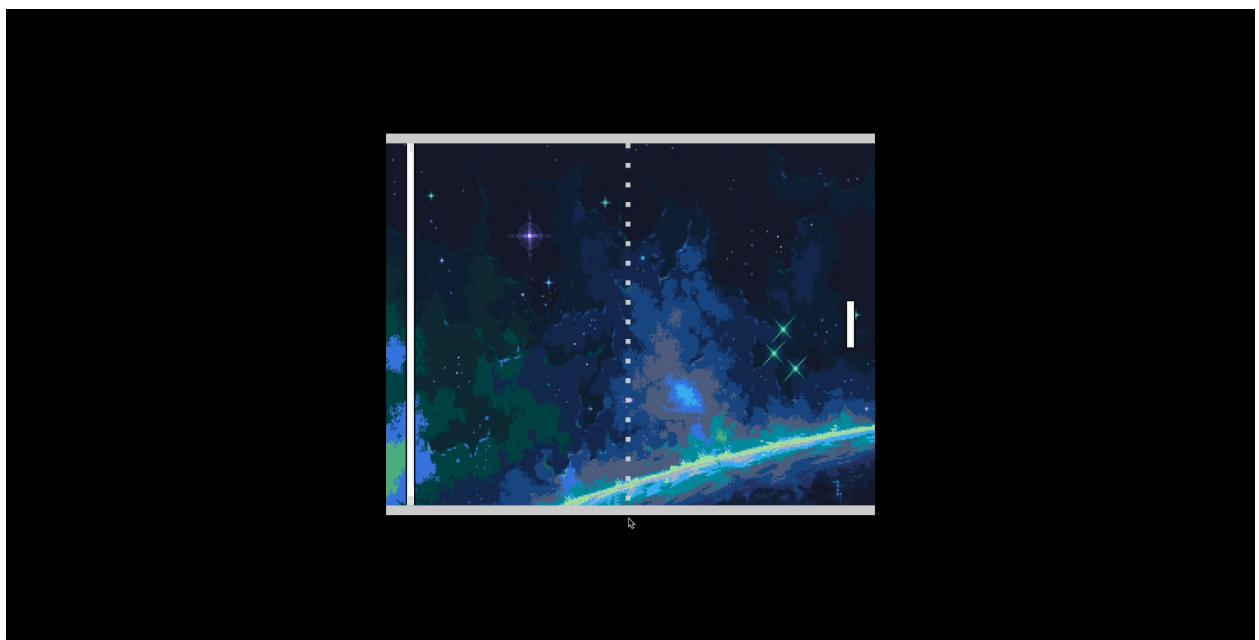
// Отрисовываем новый кадр
redraw();

// Рекурсивный вызов игрового движка
requestAnimationFrame(loop);
};

```

## Состояние игры к концу выполнения шага

На этом основная логика игры успешно нами описана. Теперь при открытии главной страницы `index.html` в браузере, можно увидеть, как мячик летит в направлении правой ракетки:



## Состояние файлов к концу выполнения шага

# engine.js

Так будет выглядеть наш файл engine.js в результате проделанной работы:

```
/ping-pong/js/engine.js
// Проверка на то, пересекаются два объекта с известными координатами или нет
// Подробнее тут: https://developer.mozilla.org/en-US/docs/Games/Techniques/2D_collision_detection
const collides = (obj1, obj2) => {
    return (
        obj1.x < obj2.x + obj2.width &&
        obj1.x + obj1.width > obj2.x &&
        obj1.y < obj2.y + obj2.height &&
        obj1.y + obj1.height > obj2.y
    );
};

// Главный цикл игры
const loop = () => {
    // Если платформы на предыдущем шаге куда-то двигались — пусть продолжают
    // двигаться
    rightPaddle.y += rightPaddle.dy;

    // Если правая платформа пытается вылезти за игровое поле вниз,
    if (rightPaddle.y < grid) {
        // то оставляем её на месте
        rightPaddle.y = grid;
    }
    // Проверяем то же самое сверху
    else if (rightPaddle.y > RightmaxPaddleY) {
        rightPaddle.y = RightmaxPaddleY;
    }

    // Если мяч на предыдущем шаге куда-то двигался — пусть продолжает
    // двигаться
    ball.x += ball.dx;
    ball.y += ball.dy;
    // Если мяч касается стены снизу — меняем направление по оси Y на
    // противоположное
    if (ball.y < grid) {
        ball.y = grid;
        ball.dy *= -1;
    }
    // Делаем то же самое, если мяч касается стены сверху
    else if (ball.y + grid > canvas.height - grid) {
        ball.y = canvas.height - grid * 2;
        ball.dy *= -1;
    }
    // Если мяч улетел за игровое поле влево или вправо — перезапускаем его
    if ((ball.x < 0 || ball.x > canvas.width) && !ball.resetting) {
        // Помечаем, что мяч перезапущен, чтобы не зациклился
        ball.resetting = true;
        // Даём секунду на подготовку игрокам
        setTimeout(() => {
            // Всё, мяч в игре
            ball.resetting = false;
            // Снова запускаем его из центра
            ball.x = canvas.width / 2;
            ball.y = canvas.height / 2;
        });
    }
}
```

```

        rightPaddle.x = canvas.width - grid * 3;
        rightPaddle.y = canvas.height / 2 - paddleHeight / 2;
    }, 1000);
}
// Если мяч коснулся левой платформы,
if (collides(ball, leftPaddle)) {
    // то отправляем его в обратном направлении
    ball.dx *= -1;
    // Увеличиваем координаты мяча на ширину платформы, чтобы не
    засчитался новый отскок
    ball.x = leftPaddle.x + leftPaddle.width;
}
// Проверяем и делаем то же самое для правой платформы
else if (collides(ball, rightPaddle)) {
    ball.dx *= -1;
    ball.x = rightPaddle.x - ball.width;
}

// Отрисовываем новый кадр
redraw();

// Рекурсивный вызов игрового движка
requestAnimationFrame(loop);
};

// Запускаем игру
requestAnimationFrame(loop);

```

## Шаг 5. Добавление управления

### Инструкции по выполнению шага

На прошлом шаге мы реализовали физику игры. Мячик летает, отпрыгивает от стен, и игра перезапускается при вылете мячика за пределы поля. Но при этом игрок ничего не может сделать, кроме как смириться с постоянными поражениями! Ракетка не двигается! Поэтому на этом шаге займёмся тем, чтобы добавить управление в игру.

Создадим дополнительный скрипт `controls.js` в папке `js`. И не забудем добавить его в главный файл `index.html` перед добавлением скрипта `engine.js`:

```

/ping-pong/index.html
<!-- Начало места, которое мы изменяем -->
</img>
<script src="js/render.js"></script>
<script src="js/controls.js"></script>
<script src="js/engine.js"></script>
</body>

</html>
<!-- Конец редактирования -->

```

Вот так будет выглядеть наша папка проекта после создания файла `controls.js`:

```

ping-pong
├── index.html
└── assets
    ├── ball.png
    ├── paddle.png
    ├── background.jpg
    └── style.css
└── js
    ├── render.js
    ├── engine.js
    └── controls.js

```

Используем для реализации управления движением ракетки стандартный обработчик событий `document.addEventListener()`. Но хитрость в том, что нам нужно отслеживать как нажатие на клавиши, так и тот момент, когда игрок их отпускает.

Смысл в том, что платформы движутся только когда игрок зажимает клавишу. Как только он её отпускает — платформа останавливается. Именно поэтому мы сделаем два обработчика: один будет следить за нажатыми клавишами, а второй — за отпущенными.

Следующий код мы добавим в файл `controls.js`:

```

/ping-pong/js/controls.js
// Вводим словарик с состояниями нажатых клавиш
const keyPresses = {
    up: 0,
    down: 0,
    nothing: 1
};

// Отслеживаем нажатия клавиш
document.addEventListener("keydown", function (e) {
    // Если нажата клавиша вверх,
    if (e.which === 38) {
        // то двигаем правую платформу вверх
        rightPaddle.dy = -rightPaddle.paddleSpeed;
        keyPresses["up"] = 1;
        keyPresses["down"] = 0;
        keyPresses["nothing"] = 0;
    }
    // Если нажата клавиша вниз,
    else if (e.which === 40) {
        // то двигаем правую платформу вниз
        rightPaddle.dy = rightPaddle.paddleSpeed;
        keyPresses["up"] = 0;
        keyPresses["down"] = 1;
        keyPresses["nothing"] = 0;
    }
    // Если нажата клавиша W,
    if (e.which === 87) {
        // то двигаем левую платформу вверх
        leftPaddle.dy = -leftPaddle.paddleSpeed;
    }
    // Если нажата клавиша S,
    else if (e.which === 83) {
        // то двигаем левую платформу вниз
        leftPaddle.dy = leftPaddle.paddleSpeed;
    }
});

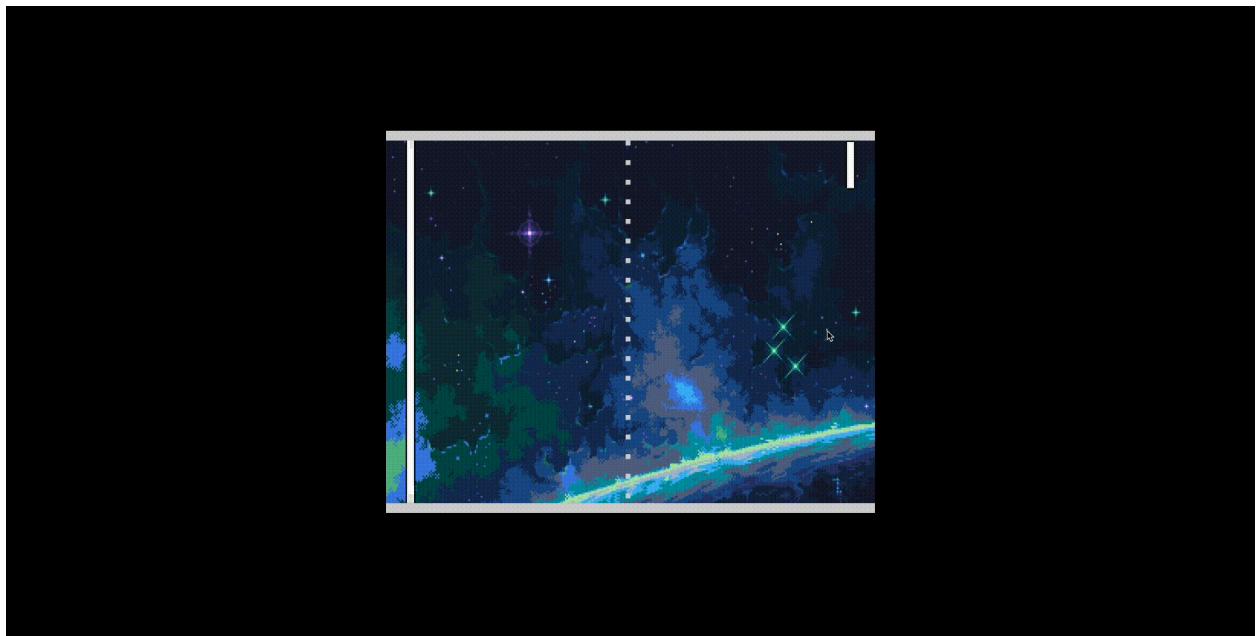
```

```
// А теперь следим за тем, когда кто-то отпустит клавишу, чтобы остановить
// движение платформы
document.addEventListener("keyup", function (e) {
    // Если это стрелка вверх или вниз,
    if (e.which === 38 || e.which === 40) {
        // останавливаем правую платформу
        rightPaddle.dy = 0;
        keyPresses["up"] = 0;
        keyPresses["down"] = 0;
        keyPresses["nothing"] = 1;
    }
    // А если это W или S,
    if (e.which === 83 || e.which === 87) {
        // останавливаем левую платформу
        leftPaddle.dy = 0;
    }
});
```

Словарик с действиями игрока keyPresses нам пригодится в дальнейшем.

## Состояние игры к концу выполнения шага

Теперь в пинг-понг действительно можно сыграть. При открытии главной страницы игры убедитесь, что стрелки вверх-вниз двигают правую ракетку.



## Шаг 6. Добавление паузы

# Инструкции по выполнению шага

Теперь, когда основные механики игры уже реализованы, можно заняться внедрением дополнительных функций. Одна из самых распространённых функций в играх это возможность поставить игровой процесс на паузу.

Для начала создадим новый скрипт и назовём его `pause.js`. Привяжем этот скрипт к `index.html` перед подключением скрипта с управлением `controls.js`:

```
/ping-pong/index.html
<!-- Начало места, которое мы изменяем -->
</img>
<script src="js/render.js"></script>
<script src="js/pause.js"></script>
<script src="js/controls.js"></script>
<script src="js/engine.js"></script>
</body>

</html>
<!-- Конец редактирования -->
```

Вот так будет выглядеть наша папка проекта после создания файла `pause.js`:

```
ping-pong
├── index.html
└── assets
    ├── ball.png
    ├── paddle.png
    ├── background.jpg
    └── style.css
└── js
    ├── render.js
    ├── engine.js
    └── controls.js
        └── pause.js
```

Добавим в только что созданный файл следующий код:

```
/ping-pong/js/pause.js
var isPaused = false;
// https://stackoverflow.com/questions/16554094/canvas-requestanimationframe-pause
window.requestAnimFrame = (function () {
    return (
        window.requestAnimationFrame ||
        window.webkitRequestAnimationFrame ||
        window.mozRequestAnimationFrame ||
        function (callback) {
            window.setTimeout(callback, 1000 / 60);
        }
    );
})();
```

```

// Отслеживаем нажатия клавиши
document.addEventListener("keydown", function (e) {
    // Если нажата клавиша ESC,
    if (e.which !== 27) return;
    console.log('Paused');
    isPaused = !isPaused;
    // Рисуем иконку паузы
    pauseDisplay();
});

```

Мы создали переменную-переключатель `isPaused`, которая хранит состояние игры. Это булева переменная и она может принимать два значения: `false` ИЛИ `true`. Аналогично как мы это делали в [шаге 5](#), добавим обработчик событий нажатий на клавишу `Escape`. При этом переводим наш переключатель в другое состояние. Если в переменной `isPaused` было записано значение `false`, то после нажатия на клавишу паузы значение будет уже `true`.

В обработчике событий вызывается функция `pauseDisplay()`. Это нестандартная функция JavaScript. Из названия можно догадаться, что она отвечает за отрисовку состояния игры во время паузы. Т.к. вся отрисовка лежит в скрипте `render.js`, нужно добавить туда следующее:

#### INFO

Следующую часть кода добавляем в конец файла `render.js`.

`/ping-pong/js/render.js`

```

// Функция отрисовки иконки паузы
const pauseDisplay = () => {
    context.fillStyle = "rgba(255, 255, 255, 0.5)";
    context.fillRect(
        canvas.width / 2 - canvas.width / 7,
        canvas.height / 3,
        canvas.width / 10,
        canvas.height / 3
    );
    context.fillRect(
        canvas.width / 2 + canvas.width / 24,
        canvas.height / 3,
        canvas.width / 10,
        canvas.height / 3
    );
}

```

Со всеми этими функциями вы уже знакомы. В `fillStyle` можно передать конкретное значение в палитре RGBA (Red, Green, Blue, Alpha), где альфа это число от 0 до 1. Этот параметр задаёт прозрачность цвета. Если альфа 0, то цвет полностью прозрачный, 1 – полностью непрозрачный. Говоря по существу, данная функция рисует два белых полупрозрачных прямоугольника поверх игрового экрана на нашем двухмерном холсте.

Вспомним, что `redraw()` у нас вызывается функцией `loop()` из файла `engine.js`. Поэтому уберите лишний вызов этой функции из файла `render.js`.

Поскольку теперь игровой процесс должен "замирать" при нажатии на кнопку паузы, то нужно контролировать вызов функции игрового движка `loop()`. Сперва удалим из скрипта `engine.js` строчку (в двух местах):

```
/ping-pong/js/engine.js
requestAnimationFrame(loop);
```

Вместо них напишем следующее:

#### INFO

Следующую часть кода добавляем в конец файла `engine.js`.

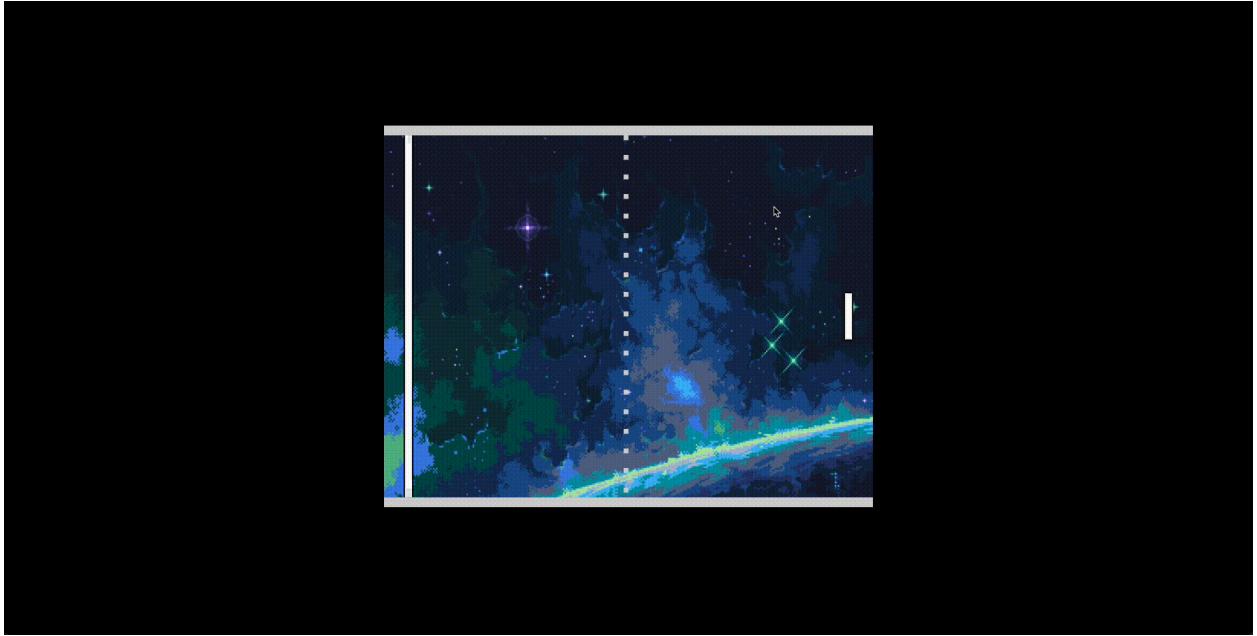
```
/ping-pong/js/engine.js
function Start() {
    if (!isPaused) {
        loop();
    }
    // Рекурсивный вызов игрового движка
    requestAnimFrame(Start);
}

Start();
loop();
```

Теперь переменная `isPaused` контролирует вызов расчёта игрового движка.

## Состояние игры к концу выполнения шага

При открытии главной страницы игры можно увидеть следующую картинку при нажатии на клавишу `ESC`:



## Состояние файлов к концу выполнения шага

### engine.js

Так будет выглядеть наш файл engine.js в результате проделанной работы:

```
/ping-pong/js/engine.js
// Проверка на то, пересекаются два объекта с известными координатами или нет
// Подробнее тут: https://developer.mozilla.org/en-US/docs/Games/Techniques/2D_collision_detection
const collides = (obj1, obj2) => {
    return (
        obj1.x < obj2.x + obj2.width &&
        obj1.x + obj1.width > obj2.x &&
        obj1.y < obj2.y + obj2.height &&
        obj1.y + obj1.height > obj2.y
    );
};

// Главный цикл игры
const loop = () => {
    // Если платформы на предыдущем шаге куда-то двигались — пусть продолжают
    // двигаться
    rightPaddle.y += rightPaddle.dy;

    // Если правая платформа пытается вылезти за игровое поле вниз,
    if (rightPaddle.y < grid) {
        // то оставляем её на месте
        rightPaddle.y = grid;
    }
    // Проверяем то же самое сверху
    else if (rightPaddle.y > RightmaxPaddleY) {
```

```

        rightPaddle.y = RightmaxPaddleY;
    }

    // Если мяч на предыдущем шаге куда-то двигался — пусть продолжает
    // двигаться
    ball.x += ball.dx;
    ball.y += ball.dy;
    // Если мяч касается стены снизу — меняем направление по оси Y на
    // противоположное
    if (ball.y < grid) {
        ball.y = grid;
        ball.dy *= -1;
    }
    // Делаем то же самое, если мяч касается стены сверху
    else if (ball.y + grid > canvas.height - grid) {
        ball.y = canvas.height - grid * 2;
        ball.dy *= -1;
    }
    // Если мяч улетел за игровое поле влево или вправо — перезапускаем его
    if ((ball.x < 0 || ball.x > canvas.width) && !ball.resetting) {
        // Помечаем, что мяч перезапущен, чтобы не зациклился
        ball.resetting = true;
        // Даём секунду на подготовку игрокам
        setTimeout(() => {
            // Всё, мяч в игре
            ball.resetting = false;
            // Снова запускаем его из центра
            ball.x = canvas.width / 2;
            ball.y = canvas.height / 2;
            rightPaddle.x = canvas.width - grid * 3;
            rightPaddle.y = canvas.height / 2 - paddleHeight / 2;
        }, 1000);
    }
    // Если мяч коснулся левой платформы,
    if (collides(ball, leftPaddle)) {
        // то отправляем его в обратном направлении
        ball.dx *= -1;
        // Увеличиваем координаты мяча на ширину платформы, чтобы не
        // засчитался новый отскок
        ball.x = leftPaddle.x + leftPaddle.width;
    }
    // Проверяем и делаем то же самое для правой платформы
    else if (collides(ball, rightPaddle)) {
        ball.dx *= -1;
        ball.x = rightPaddle.x - ball.width;
    }

    // Отрисовываем новый кадр
    redraw();
};

// Запускаем игру
function Start() {
    if (!isPaused) {
        loop();
    }
    // Рекурсивный вызов игрового движка
    requestAnimFrame(Start);
}

Start();
loop();

```

# render.js

Так будет выглядеть наш файл render.js в результате проделанной работы:

```
/ping-pong/js/render.js
// Обращаемся к игровому полю из документа
const canvas = document.getElementById("game");

// Обращаемся к изображению ракетки из документа
const paddleImg = document.getElementById("paddle");

// Обращаемся к изображению мячика из документа
const ballImg = document.getElementById("ball");

// Обращаемся к изображению фона из документа
const backgroundImg = document.getElementById("background");

// Размер игровой клетки
const grid = 15;

// Высота платформы
const paddleHeight = grid * 5; // 80

// Задаём максимальное расстояние, на которое могут двигаться платформы
const LeftmaxPaddleY = canvas.height - grid - paddleHeight * 2;
const RightmaxPaddleY = canvas.height - grid - paddleHeight;

// Описываем левую платформу
const leftPaddle = {
    // Ставим её по центру
    x: grid * 2,
    y: 0,
    // Ширина – одна клетка
    width: grid,
    // Высоту берём из константы
    height: canvas.height, //paddleHeight * 2,
    // Платформа на старте никуда не движется
    dy: 0,
    paddleSpeed: 10
};
leftPaddle.dy = 0; //paddleSpeed;
// Описываем правую платформу
const rightPaddle = {
    // Ставим по центру с правой стороны
    x: canvas.width - grid * 3,
    y: canvas.height / 2 - paddleHeight / 2,
    // Задаём такую же ширину и высоту
    width: grid,
    height: paddleHeight,
    // Правая платформа тоже пока никуда не движется
    dy: 0,
    paddleSpeed: 10
};
var ballSpeed = 5;
// Описываем мячик
const ball = {
    // Он появляется в самом центре поля
    x: canvas.width / 2,
    y: canvas.height / 2,
    // квадратный, размером с клетку
    width: grid * 2,
```

```
height: grid * 2,
// На старте мяч пока не забит, поэтому убираем признак того, что мяч
// нужно ввести в игру заново
resetting: false,
// Подаём мяч в правый верхний угол
dx: ballSpeed,
dy: -ballSpeed
};

// Отключаем видимость элементов
paddleImg.style.display = "none";
ballImg.style.display = "none";
backgroundImg.style.display = "none";

// Делаем поле двухмерным
const context = canvas.getContext("2d");

// Перепишем отрисовку как функцию для переиспользования
const redraw = () => {
    // Очищаем холст от предыдущего кадра
    context.clearRect(0, 0, canvas.width, canvas.height);

    // Рисуем содержимое заднего фона на холсте
    context.drawImage(backgroundImg, 0, 0, backgroundImg.width,
backgroundImg.height, 0, 0, canvas.width, canvas.height);

    // Рисуем левую ракетку на холсте
    context.drawImage(paddleImg, 0, 0, paddleImg.width, paddleImg.height,
leftPaddle.x, leftPaddle.y, leftPaddle.width, leftPaddle.height);

    // Рисуем мячик
    context.drawImage(ballImg, 0, 0, ballImg.width, ballImg.height, ball.x,
ball.y, ball.width, ball.height);

    // Рисуем правую ракетку на холсте
    context.drawImage(paddleImg, 0, 0, paddleImg.width, paddleImg.height,
rightPaddle.x, rightPaddle.y, rightPaddle.width, rightPaddle.height);

    // Рисуем стены
    context.fillStyle = "lightgrey";
    context.fillRect(0, 0, canvas.width, grid);
    context.fillRect(0, canvas.height - grid, canvas.width, canvas.height);

    // Рисуем сетку посередине
    for (let i = grid; i < canvas.height - grid; i += grid * 2) {
        context.fillRect(canvas.width / 2 - grid / 2, i, grid / 2, grid / 2);
    };
};

// ФУНКЦИЯ ОТРИСОВКИ ИКОНКИ ПАУЗЫ
const pauseDisplay = () => {
    context.fillStyle = "rgba(255, 255, 255, 0.5)";
    context.fillRect(
        canvas.width / 2 - canvas.width / 7,
        canvas.height / 3,
        canvas.width / 10,
        canvas.height / 3
    );
    context.fillRect(
        canvas.width / 2 + canvas.width / 24,
        canvas.height / 3,
        canvas.width / 10,
        canvas.height / 3
    );
};
```

```
};
```

## Шаг 7. Добавление доски рекордов

### Инструкции по выполнению шага

Игра продолжает обрастать новыми фишками. На этот раз давайте добавим в неё доску со статистиками игры. Согласитесь, когда вы видите результат ваших игр, то игровой процесс получается более азартным: есть стремление побить свой прошлый рекорд. На этом шаге мы с вами добавим рядом с холстом игры несколько чисел, которые будут отражать способности игрока:

- *Current* – текущее число успешно отбитых столкновений с мячиком;
- *Best score* – максимальное число успешно отбитых столкновений с мячиком без проигрышей за всё время;
- *Average* – среднее число успешно отбитых столкновений с мячиком за все игры.

Данные характеристики игры можно добавить на главную страницу с помощью HTML элементов, которые впоследствии нужно будет расположить в нужном месте при помощи CSS. Саму логику же реализуем при помощи скрипта на JavaScript.

Для начала давайте добавим в главный файл `index.html` в элементе `<body>` перед добавлением скриптов следующие строчки:

```
/ping-pong/index.html
<!-- Начало места, которое мы изменяем -->
</img>
<h1 class="score">
    Current: <a>0</a><br />
    Best score: <a>0</a><br />
    Average: <a>0</a>
</h1>
<script src="js/render.js"></script>
<!-- Конец редактирования -->
```

Элемент `<h1>` это элемент заголовка – текст будет отображаться большим. Существуют также и заголовки `<h2>`, `<h3>`, `<h4>`, `<h5>`. Каждый из них как в матрёшке имеет все меньший размер текста. Для разнообразия добавим вместо уникального идентификатора, как мы уже делали ранее на [Шаге 0](#), имя класса при помощи атрибута `class`. Имя класса, в отличие от значения уникального идентификатора может повторяться внутри одного HTML файла. Зачастую рекомендуется использовать именно атрибуты классов, чтобы обращаться к конкретным

элементам в CSS или JavaScript. Внутри так же используются элементы `<a>` и `<br>`. Последний позволяет сделать перенос строки при отображении текста в браузере. А первый чаще всего используется для добавления гиперссылок, однако в данном конкретном случае мы будем его использовать в качестве маркера. При написании JavaScript кода это позволит изменять конкретное значение содержимого ссылки.

После добавления элемента с текстом рекордов, нужно его правильно расположить на главной странице. Для этого добавим следующие строчки в конец нашего CSS файла `style.css`:

#### INFO

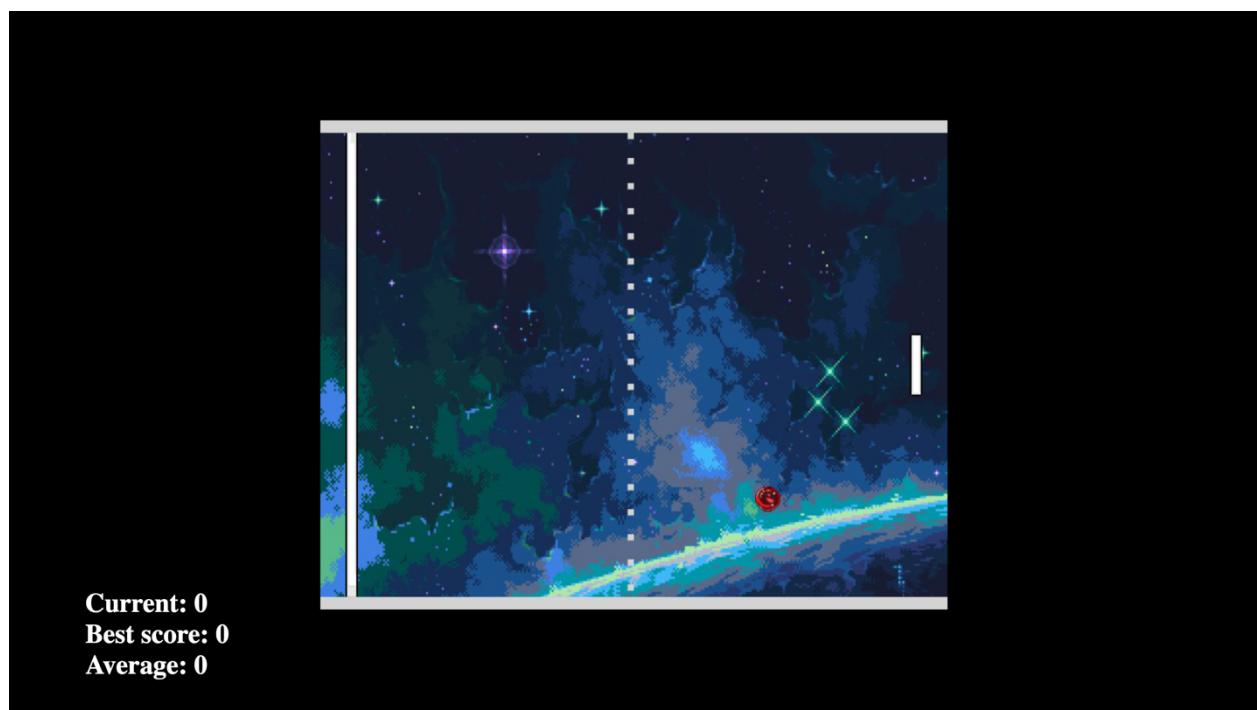
Следующая часть добавляется в конец файла `style.css`.

`/ping-pong/assets/style.css`

```
.score {  
    color: white;  
    position: fixed;  
    left: 100px;  
    bottom: 35px;  
}
```

Тут происходит обращение через селектор класса к HTML элементам с таким именем класса (`score`). К ним применяется настройки цвета текста, фиксированного положения. После чего элементам задаются координаты от левого и нижнего краёв страницы.

Если вы теперь откроете главную страницу игры, то увидите следующую картину:



Доска рекордов теперь отображается на экране, но в процессе игры ничего не происходит. Всё потому, что у нас нет логики изменения чисел в элементе с оценками. Поэтому создадим в папке js новый скрипт scores.js и не забудем его подгрузить в index.html перед скриптом с движком:

```
/ping-pong/index.html
<!-- Начало места, которое мы изменяем -->
</img>
<script src="js/render.js"></script>
<script src="js/pause.js"></script>
<script src="js/controls.js"></script>
<script src="js/scores.js"></script>
<script src="js/engine.js"></script>
</body>

</html>
<!-- Конец редактирования -->
```

Вот так будет выглядеть наша папка проекта после создания файла scores.js:

```
ping-pong
├── index.html
└── assets
    ├── ball.png
    ├── paddle.png
    ├── background.jpg
    └── style.css
└── js
    ├── render.js
    ├── engine.js
    ├── controls.js
    ├── pause.js
    └── scores.js
```

В самом только что созданном файле создадим переменную-словарик scoreStats аналогичную переменным ball и rightPaddle из render.js:

#### INFO

Следующая часть кода добавляется в начало файла scores.js.

```
/ping-pong/js/scores.js
var scoreStats = {
  currentCombo: 0,
  bestCombo: 0,
  totalTries: 0,
  mean: 0
};
```

Тут мы задали поля с начальными значениями в виде нулей. Текущее число отбитых столкновений хранится в поле currentCombo, лучшее число отбитых столкновений – в поле bestCombo. Для подсчёта среднего значения по всем играм нам нужно минимум два поля, это число сыгранных партий и собственно сама

средняя оценка. В дальнейшем мы реализуем алгоритм, который позволит обновлять значение среднего по следующей рекуррентной формуле:

$$\text{Average}(n) = \text{Average}(n-1) + \frac{\text{Value} - \text{Average}(n-1)}{n}$$
$$\text{Average}(n) = \text{Average}(n-1) + n \cdot \text{Value} - \text{Average}(n-1), \quad (1)$$

где  $n$  – число сыгранных игр,  $\text{Average}(n-1)$  – среднее за прошлые игры,  $\text{Value}$  – оценка за эту последнюю. Заметим, что логически у нас обновление доски рекордов происходит при трёх сценариях:

- В результате столкновения ракетки с мячом;
- В случае проигрыша;
- В случае сброса игры.

Тогда напишем функцию `updateStatus`, которая будет соответствовать этой логике:

## INFO

Следующая часть добавляется в конец файла `scores.js`.

```
//ping-pong/js/scores.js
// вызывается при проигрыше
const updateStatus = (gameover = false, reset = false) => {
    const scoreElements = Array.from(
        document.getElementsByClassName("score") [0].getElementsByTagName("a")
    );
    if (!gameover) {
        scoreStats.currentCombo++;
        scoreStats.bestCombo = Math.max(
            scoreStats.currentCombo,
            scoreStats.bestCombo
        );
    } else {
        scoreStats.totalTries++;
        scoreStats.mean =
            Math.floor(
                (scoreStats.mean +
                    (scoreStats.currentCombo - scoreStats.mean) /
                scoreStats.totalTries) *
                100
            ) / 100;
        scoreStats.currentCombo = 0;
    }
    if (reset) {
        scoreStats.totalTries = 0;
        scoreStats.currentCombo = 0;
        scoreStats.bestCombo = 0;
        scoreStats.mean = 0;
    }
    // Current
    scoreElements[0].textContent = scoreStats.currentCombo;
    // Best
    scoreElements[1].textContent = scoreStats.bestCombo;
    // Average
    scoreElements[2].textContent = scoreStats.mean;
};
```

Для начала получаем доступ к элементу с рейтингом при помощи метода `document.getElementsByClassName`. По умолчанию этот метод возвращает набор подходящих элементов. Поскольку у нас в HTML файле всего один такой элемент, то мы обращаемся к нему через индекс `[0]`. Т.к. внутри элемента с рейтингом `<h1>` лежат ещё и дочерние элементы ссылок `<a>`, обратимся к ним при помощи метода `getElementsByName("a")`. Этот метод так же возвращает набор элементов. В случае, когда игра не завершена, т.е. когда мячик столкнулся с ракеткой, мы увеличиваем значение текущей оценки на единичку при помощи оператора инкремента `++`. В тот же момент нам нужно проверить установки нового рекорда. Чтобы не писать дополнительных условных операций, воспользуемся стандартной функцией `Math.max`. Она возвращает максимальный элемент из набора переданных аргументов. В случае проигрыша, необходимо увеличить счётчик сыгранных партий, а затем обновить значение среднего числа по ранее упомянутой формуле (1)(1). Среднее может иметь дробные разряды. Так, если было сыграно три игры с числом столкновений 1, 1 и 2, то среднее будет  $1 \cdot (3)$ . В JavaScript такие дробные числа представляются в виде чисел с плавающей точкой и будут выглядеть, как  $1.3333333333333333$ . Что выглядит не слишком аккуратно. Поэтому в коде выполняется округление значений через умножение на 100, округление до целых в меньшую сторону функцией `Math.floor` и последующее деление на 100. В результате таких махинаций предыдущий пример будет отображаться уже в виде 1.33. Наконец, после проигрыша надо не забыть обнулить текущее число оценок.

На последнем этапе мы обращаемся к элементам `<a>` и записываем в них текущие значения полей из словарика. Для этого используются индексация массива и метод `textContent`.

После реализации функции обновления оценок игры, их нужно добавить в игровой движок. То есть в основной цикл игры – функцию `loop`.

Найдите в `engine.js` следующие строчки и допишите вызов функции:

## INFO

Следующая часть кода модифицирует в конец файла `engine.js`, а именно добавляется вызов функции `updateStatus()` в нескольких местах .  
`/ping-pong/js/engine.js`

```
rightPaddle.y = canvas.height / 2 - paddleHeight / 2;
// Обновляем доску с рекордами
updateStatus(true);
}, 1000);
}
// Если мяч коснулся левой платформы,
if (collides(ball, leftPaddle)) {
    // то отправляем его в обратном направлении
    ball.dx *= -1;
    // Увеличиваем координаты мяча на ширину платформы, чтобы не
    засчитался новый отскок
    ball.x = leftPaddle.x + leftPaddle.width;
```

```

}

// Проверяем и делаем то же самое для правой платформы
else if (collides(ball, rightPaddle)) {
    ball.dx *= -1;
    ball.x = rightPaddle.x - ball.width;

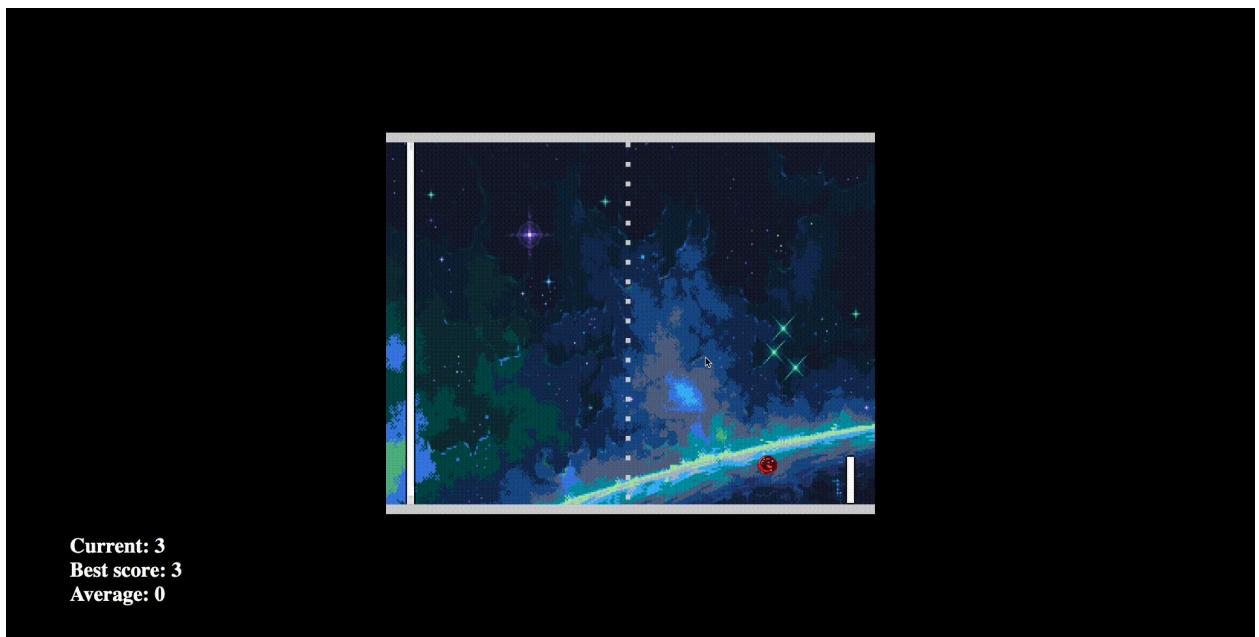
    // Обновляем доску с рекордами
    updateStatus(false);
}

// Отрисовываем новый кадр
redraw();
};

```

## Состояние игры к концу выполнения шага

Попробуйте теперь запустить игры в браузере и сыграть несколько партий. Числа на доске должны обновляться:



## Состояние файлов к концу выполнения шага

### engine.js

Так будет выглядеть наш файл `engine.js` в результате проделанной работы:

## /ping-pong/js/engine.js

```
// Проверка на то, пересекаются два объекта с известными координатами или нет
// Подробнее тут: https://developer.mozilla.org/en-US/docs/Games/Techniques/2D_collision_detection
const collides = (obj1, obj2) => {
    return (
        obj1.x < obj2.x + obj2.width &&
        obj1.x + obj1.width > obj2.x &&
        obj1.y < obj2.y + obj2.height &&
        obj1.y + obj1.height > obj2.y
    );
};

// Главный цикл игры
const loop = () => {
    // Если платформы на предыдущем шаге куда-то двигались — пусть продолжают
    // двигаться
    rightPaddle.y += rightPaddle.dy;

    // Если правая платформа пытается вылезти за игровое поле вниз,
    if (rightPaddle.y < grid) {
        // то оставляем её на месте
        rightPaddle.y = grid;
    }
    // Проверяем то же самое сверху
    else if (rightPaddle.y > RightmaxPaddleY) {
        rightPaddle.y = RightmaxPaddleY;
    }

    // Если мяч на предыдущем шаге куда-то двигался — пусть продолжает
    // двигаться
    ball.x += ball.dx;
    ball.y += ball.dy;
    // Если мяч касается стены снизу — меняем направление по оси Y на
    // противоположное
    if (ball.y < grid) {
        ball.y = grid;
        ball.dy *= -1;
    }
    // Делаем то же самое, если мяч касается стены сверху
    else if (ball.y + grid > canvas.height - grid) {
        ball.y = canvas.height - grid * 2;
        ball.dy *= -1;
    }
    // Если мяч улетел за игровое поле влево или вправо — перезапускаем его
    if ((ball.x < 0 || ball.x > canvas.width) && !ball.resetting) {
        // Помечаем, что мяч перезапущен, чтобы не зациклиться
        ball.resetting = true;
        // Даём секунду на подготовку игрокам
        setTimeout(() => {
            // Всё, мяч в игре
            ball.resetting = false;
            // Снова запускаем его из центра
            ball.x = canvas.width / 2;
            ball.y = canvas.height / 2;
            rightPaddle.x = canvas.width - grid * 3;
            rightPaddle.y = canvas.height / 2 - paddleHeight / 2;

            // Обновляем доску с рекордами
            updateStatus(true);
        }, 1000);
    }
    // Если мяч коснулся левой платформы,
```

```

if (collides(ball, leftPaddle)) {
    // то отправляем его в обратном направлении
    ball.dx *= -1;
    // Увеличиваем координаты мяча на ширину платформы, чтобы не
    засчитался новый отскок
    ball.x = leftPaddle.x + leftPaddle.width;
}
// Проверяем и делаем то же самое для правой платформы
else if (collides(ball, rightPaddle)) {
    ball.dx *= -1;
    ball.x = rightPaddle.x - ball.width;

    // Обновляем доску с рекордами
    updateStatus(false);
}

// Отрисовываем новый кадр
redraw();
};

// Запускаем игру
function Start() {
    if (!isPaused) {
        loop();
    }
    // Рекурсивный вызов игрового движка
    requestAnimFrame(Start);
}

Start();
loop();

```

## Шаг 8. Сохранение информации о матчах. Часть 1

### Инструкции по выполнению шага

Следующей полезной функции игры будет "запись матчей". Нам бы хотелось сохранять историю траекторий мячика, ракетки и нажатий на клавиши управления, чтобы в дальнейшем можно было бы использовать это в качестве данных для обучения искусственного интеллекта. В этом шаге мы займёмся предварительными приготовлениями: созданием HTML-элементов, настройкой стилей, написанием логики сбора данных в виде JavaScript скриптов. А на следующем шаге мы создадим кнопку, с помощью которой можно эти данные будет выгрузить на жесткий диск в виде файла. Итак, приступим.

Для начала создадим два элемента в `index.html` перед элементом с доской рейтинга из прошлого шага:

```
/ping-pong/index.html
<!-- Начало места, которое мы изменяем -->
</img>
<h5 class="logging"></h5>
<h1 class="datasetSize">0</h1>
<h1 class="score">
    Current: <a>0</a><br />
    Best score: <a>0</a><br />
    Average: <a>0</a>
</h1>
<!-- Конец редактирования -->
```

В первом элементе будем отображать текущее значение системы: координаты мячика и ракеток, а так же флаги нажатых клавиш. Во втором элементе будем вести подсчёт числа сохранённых состояний игры.

Аналогично прошлому шагу, эти элементы нужно выровнять на главной странице. Поэтому добавим в файл `style.css` следующие настройки:

## INFO

Следующая часть добавляется в конец файла `style.css`.

`/ping-pong/assets/style.css`

```
.logging {
    color: white;
    position: fixed;
    left: 100px;
    top: 50px;
}

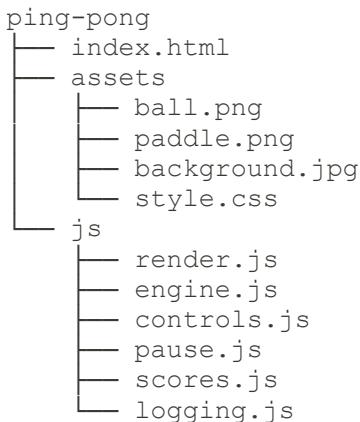
.datasetSize {
    color: white;
    position: fixed;
    left: 400px;
    top: 35px;
}
```

Теперь опишем логику сохранения состояний игры. Для этого создадим в папке `js` новый скрипт `logging.js`. Не забудем подгрузить его в `index.html` перед скриптом `scores.js`:

```
/ping-pong/index.html
<!-- Начало места, которое мы изменяем -->
<script src="js/render.js"></script>
<script src="js/pause.js"></script>
<script src="js/controls.js"></script>
<script src="js/logging.js"></script>
<script src="js/scores.js"></script>
<script src="js/engine.js"></script>
</body>

</html>
<!-- Конец редактирования -->
```

Вот так будет выглядеть наша папка проекта после создания файла `logging.js`:



В файле `logging.js` напишем следующим код:

```
/ping-pong/js/logging.js
const logText = document.getElementsByClassName("logging")[0];
const datasetSize = document.getElementsByClassName("datasetSize")[0];
let lastGoodMove = 1;
const header = "ball_x,ball_y,paddle_y,up,down,nothing";
let recordData = [header];

const updateDataset = () => {
    logText.innerHTML = [
        ball.x,
        ball.y,
        rightPaddle.y,
        keyPresses.up,
        keyPresses.down,
        keyPresses.nothing
    ];
    recordData.push(logText.innerHTML);
    datasetSize.textContent = recordData.length - 1;
}
```

Тут мы получаем доступ к ранее созданным HTML элементам. Создаём переменную номер состояния игры последнего отбитого мяча. Она позволит нам автоматически отбрасывать плохие куски партий, когда игрок пропустил мячик. Переменная `header` содержит названия состояний системы, которые мы будем сохранять:

- `ball_x` – горизонтальная координата мячика;
- `ball_y` – вертикальная координата мячика;
- `paddle_y` – вертикальная координата ракетки;
- `up` – флаг зажатой стрелки вверх;
- `down` – флаг зажатой стрелки вниз;
- `nothing` – флаг отсутствия нажатий на клавиши.

Помимо этого, заведём массив `recordData`, в который и будем сохранять вышеупомянутые состояния системы.

Наконец, создадим функцию `updateDataset`. При каждом её вызове она последовательно запишем в HTML элемент `logging` текущее состояние системы, затем добавит его в конец массива `recordData` при помощи метода `push` и наконец обновит HTML элемент `datasetSize`, записав в него текущее число хранящихся состояний игры за вычетом первого элемента с описанием признаков.

Финальным этапом внедряем написанную логику в движок игры в скрипте `engine.js`.

В начале цикла игры:

INFO

Следующая часть отображает изменения файла `engine.js`.

```
/ping-pong/js/engine.js
// Главный цикл игры
const loop = () => {
    // Сохраняем текущие значения объектов
    updateDataset();
    // Если платформы на предыдущем шаге куда-то двигались — пусть продолжают
    // двигаться
    rightPaddle.y += rightPaddle.dy;
```

Помимо этого добавим в функцию обратного вызова метода `setTimeout` после проигрыша следующий код:

INFO

Следующие части отображают изменения файла `engine.js`.

```
/ping-pong/js/engine.js
rightPaddle.y = canvas.height / 2 - paddleHeight / 2;

// Вырезаем плохой фрагмент из датасета
recordData = recordData.slice(0, lastGoodMove);

// Обновляем доску с рекордами
updateStatus(true);
}, 1000);
```

В обработчике столкновений мячика с ракеткой обновляем значение переменной последнего удачного момента:

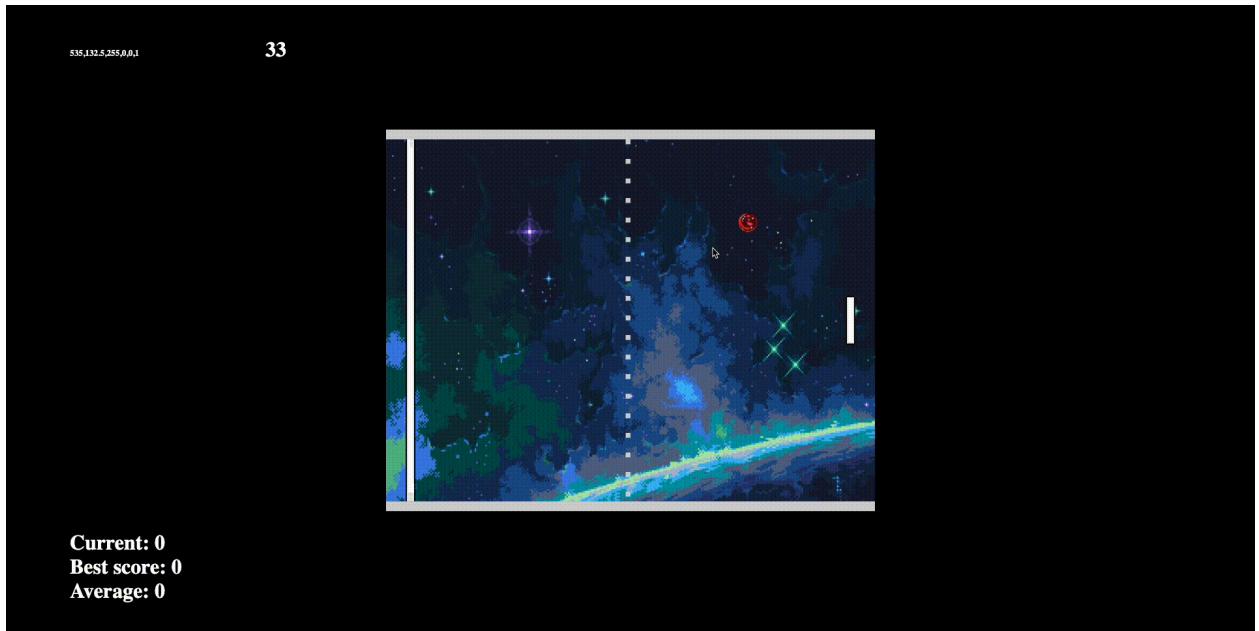
```
/ping-pong/js/engine.js
ball.x = rightPaddle.x - ball.width;

// Обновляем доску с рекордами
updateStatus(false);
// Сохраняем временную позицию последнего удачно отбитого мячика
lastGoodMove = recordData.length;
```

}

# Состояние игры к концу выполнения шага

На этом всё. Теперь при запуске игры можно увидеть в верхнем левом углу новые элементы: текущее состояние системы и число сохранённых состояний.



# Состояние файлов к концу выполнения шага

## engine.js

```
/ping-pong/js/engine.js
// Проверка на то, пересекаются два объекта с известными координатами или нет
// Подробнее тут: https://developer.mozilla.org/en-US/docs/Games/Techniques/2D_collision_detection
const collides = (obj1, obj2) => {
    return (
        obj1.x < obj2.x + obj2.width &&
        obj1.x + obj1.width > obj2.x &&
        obj1.y < obj2.y + obj2.height &&
        obj1.y + obj1.height > obj2.y
    );
};

// Главный цикл игры
const loop = () => {
```

```

// Сохраняем текущие значения объектов
updateDataset();

// Если платформы на предыдущем шаге куда-то двигались — пусть продолжают
// двигаться
rightPaddle.y += rightPaddle.dy;

// Если правая платформа пытается вылезти за игровое поле вниз,
if (rightPaddle.y < grid) {
    // то оставляем её на месте
    rightPaddle.y = grid;
}
// Проверяем то же самое сверху
else if (rightPaddle.y > RightmaxPaddleY) {
    rightPaddle.y = RightmaxPaddleY;
}

// Если мяч на предыдущем шаге куда-то двигался — пусть продолжает
// двигаться
ball.x += ball.dx;
ball.y += ball.dy;
// Если мяч касается стены снизу — меняем направление по оси Y на
// противоположное
if (ball.y < grid) {
    ball.y = grid;
    ball.dy *= -1;
}
// Делаем то же самое, если мяч касается стены сверху
else if (ball.y + grid > canvas.height - grid) {
    ball.y = canvas.height - grid * 2;
    ball.dy *= -1;
}
// Если мяч улетел за игровое поле влево или вправо — перезапускаем его
if ((ball.x < 0 || ball.x > canvas.width) && !ball.resetting) {
    // Помечаем, что мяч перезапущен, чтобы не зациклился
    ball.resetting = true;
    // Даём секунду на подготовку игрокам
    setTimeout(() => {
        // Всё, мяч в игре
        ball.resetting = false;
        // Снова запускаем его из центра
        ball.x = canvas.width / 2;
        ball.y = canvas.height / 2;
        rightPaddle.x = canvas.width - grid * 3;
        rightPaddle.y = canvas.height / 2 - paddleHeight / 2;

        // Вырезаем плохой фрагмент из датасета
        recordData = recordData.slice(0, lastGoodMove);

        // Обновляем доску с рекордами
        updateStatus(true);
    }, 1000);
}
// Если мяч коснулся левой платформы,
if (collides(ball, leftPaddle)) {
    // то отправляем его в обратном направлении
    ball.dx *= -1;
    // Увеличиваем координаты мяча на ширину платформы, чтобы не
    // засчитался новый отскок
    ball.x = leftPaddle.x + leftPaddle.width;
}
// Проверяем и делаем то же самое для правой платформы
else if (collides(ball, rightPaddle)) {
    ball.dx *= -1;
}

```

```

ball.x = rightPaddle.x - ball.width;

// Обновляем доску с рекордами
updateStatus(false);
// Сохраняем временную позицию последнего удачно отбитого мячика
lastGoodMove = recordData.length;
}

// Отрисовываем новый кадр
redraw();
};

// Запускаем игру
function Start() {
    if (!isPaused) {
        loop();
    }
    // Рекурсивный вызов игрового движка
    requestAnimFrame(Start);
}

Start();
loop();

```

## Шаг 9. Сохранение информации о матчах. Часть 2

### Инструкции по выполнению шага

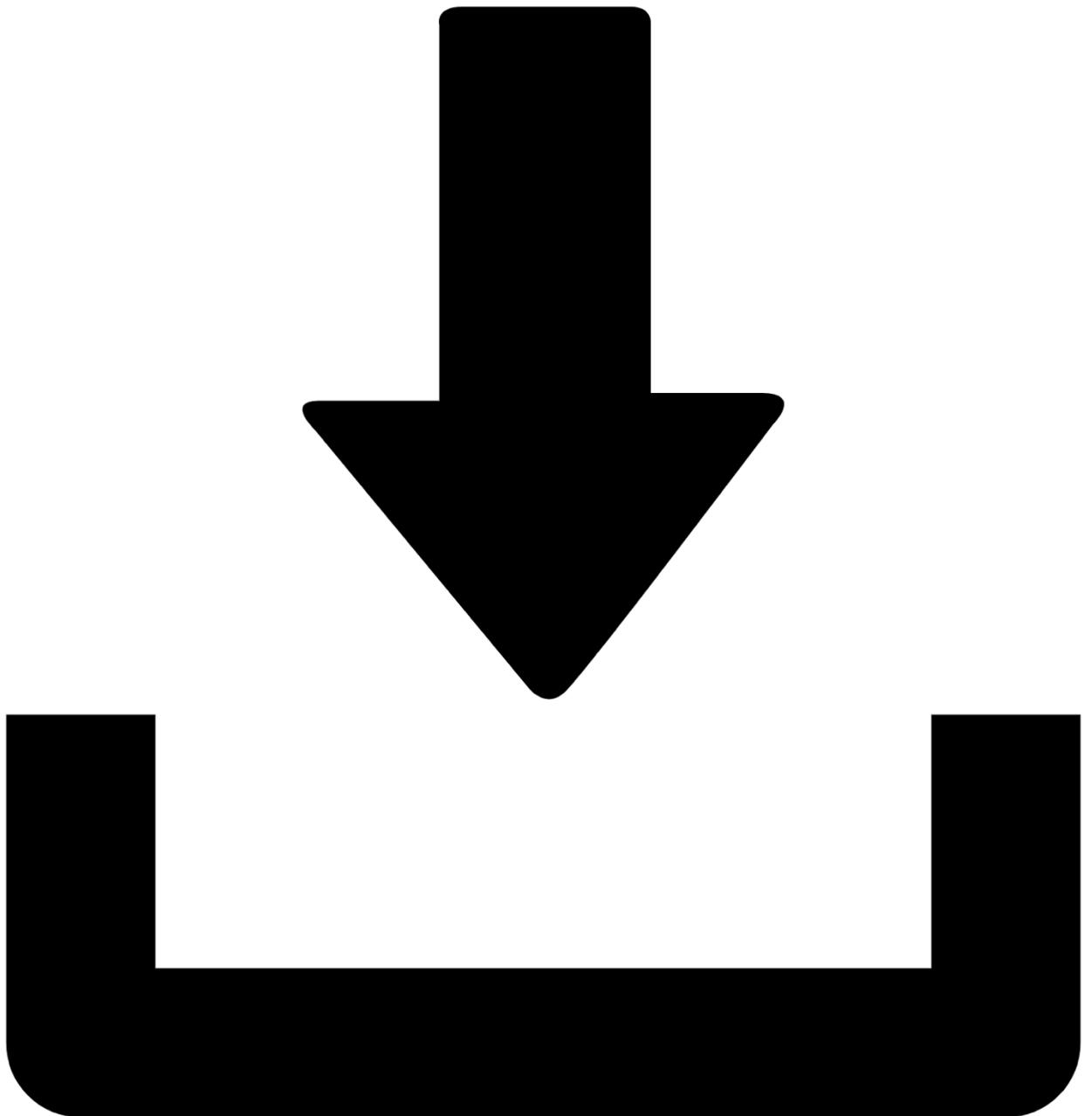
На данный момент игра сохраняет сведения о положениях мячика, ракетки и нажатий. Однако не имеется никакой возможности выгрузить эти данные или где-то сохранить. Для того, чтобы получилось сделать такую функцию, внесём следующие изменения в наш главный файл `index.html` сразу после элемента `datasetSize`:

```

/ping-pong/index.html
<!-- Начало места, которое мы изменяем -->
<h1 class="datasetSize">0</h1>
<div class="download">
    <a
download="ping_pong_dataset.csv" href='none'></a></img>
</div>
<h1 class="score">
    Current: <a>0</a><br />
    Best score: <a>0</a><br />
    Average: <a>0</a>
</h1>
<!-- Конец редактирования -->

```

Тут мы создали элемент группы `<div>` и вложили в него элемент изображения `<img>`. А внутри изображения `<img>` мы вложили элемент ссылки `<a>`, который будем использовать для выгрузки файла в формате CSV. Файл CSV это текстовый файл, в котором содержится информация. Каждая строка – это отдельная строка таблицы, а столбцы отделены один от другого знаком разделителя. Чаще всего разделитель это запятая , или точка с запятой ;. Для картинки можно использовать любое изображение, отдаленно ассоциирующееся с кнопкой загрузки. В нашем примере будет использоваться следующая картинка:



Положение только что созданного элемента необходимо зафиксировать, поэтому внесём в файл `style.css` следующие настройки:

## INFO

Следующая часть добавляется в конец файла style.css.

/ping-pong/assets/style.css

```
.download {
    position: fixed;
    cursor: pointer;
    left: 250px;
    top: 50px;
}
```

Новая настройка `cursor: pointer` меняет внешний вид мышки при наведении на элемент с классом `download`. Тем самым пользователю даётся подсказка, что это не просто картинка, а кнопка, на которую можно нажать.

Аналогично прошлым шагам, опишем логику выгрузки файла в виде скрипта на JavaScript. Создаём новый файл в папке `js` под именем `download.js` и не забываем добавить его в `index.html` перед подгрузкой скрипта с движком игры:

/ping-pong/index.html

```
<!-- Начало места, которое мы изменяем -->
<script src="js/scores.js"></script>
<script src="js/download.js"></script>
<script src="js/engine.js"></script>
</body>

</html>
<!-- Конец редактирования -->
```

Вот так будет выглядеть наша папка проекта после создания файла `download.js`:

```
ping-pong
├── index.html
└── assets
    ├── ball.png
    ├── paddle.png
    ├── background.jpg
    ├── style.css
    └── download.svg
└── js
    ├── render.js
    ├── engine.js
    ├── controls.js
    ├── pause.js
    ├── scores.js
    ├── logging.js
    └── download.js
```

В самом файле напишем следующий код:

/ping-pong/js/download.js

```
// https://stackoverflow.com/questions/21012580/is-it-possible-to-write-data-
// to-file-using-only-javascript
let textFile = null;
```

```

const makeTextFile = function (text) {
    var data = new Blob([text], {
        type: "text/plain"
    });

    // При замене ранее сгенерированного файла необходимо вручную удалить
    // связь с привязанной к нему ссылкой во избежание утечек памяти
    if (textFile !== null) {
        window.URL.revokeObjectURL(textFile);
    }

    textFile = window.URL.createObjectURL(data);

    return textFile;
};

const downloadBtn = document.getElementsByClassName("download")[0];

downloadBtn.addEventListener(
    "click",
    function () {
        var link = downloadBtn.getElementsByTagName("a")[0];
        link.href = makeTextFile(recordData.join("\r\n"));
        link.style.display = "block";
        link.click();
    },
    false
);

```

В начале файла мы объявляем переменную с пустым значением `null`, в которой будем хранить ссылку на текстовое представление массива `recordData`. Ниже объявляется функция `makeTextFile`. Ей в качестве входного аргумента передаётся текст, который подаётся на вход объекту `Blob`. Объект `Blob` - это объект, подобный файлу, который содержит необработанные данные только для чтения. Поскольку это файлоподобный объект, то к нему можно получить доступ в виде гиперссылки при помощи метода `window.URL.createObjectURL`. После получения ссылки на объект, функция `makeTextFile` её возвращает как результат своей работы.

После этого нам осталось только получить доступ к элементу с иконкой загрузки и привязать к нему обработчик событий клика клавиши. Внутри функции обратного вызова мы получаем доступ к элементу ссылке при помощи метода `getElementsByTagName`. Устанавливаем в качестве значения атрибута `href` (от англ. *hypertext reference* — гипертекстовая ссылка) полученную из текстового представления `recordData` гиперссылку. Включаем её отображение и вызываем эмуляцию нажатия при помощи метода `click`.

Проделанные шаги теперь позволяют выкачивать набор данных с записями состояний матчей в виде текстового файла. Пример такого файла показан на рисунке ниже:



A

B

C

D

E

F

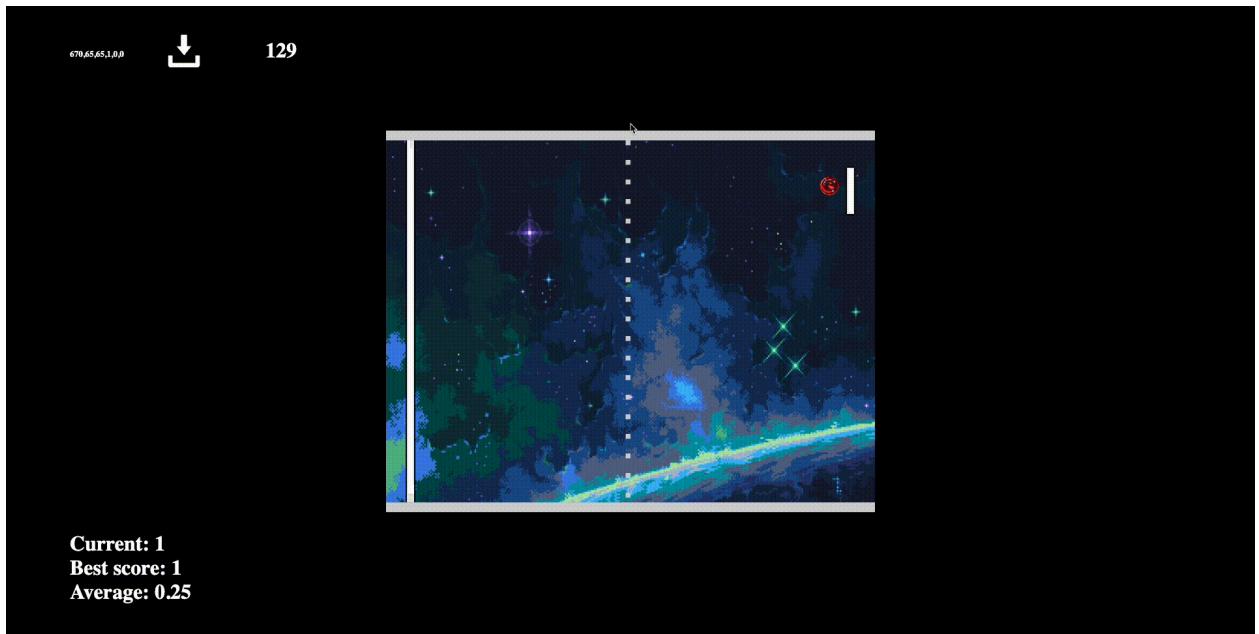


### ping\_pong\_dataset (4)

	ball_x	ball_y	paddle_y	up	down	nothing
1						
2	375	292.5	255	0	0	1
3	380	287.5	255	0	0	1
4	385	282.5	255	0	0	1
5	390	277.5	255	0	0	1
6	395	272.5	255	0	0	1
7	400	267.5	255	0	0	1
8	405	262.5	255	0	0	1
9	410	257.5	255	0	0	1
10	415	252.5	255	0	0	1
11	420	247.5	255	0	0	1
12	425	242.5	255	0	0	1
13	430	237.5	255	0	0	1
14	435	232.5	255	0	0	1
15	440	227.5	255	0	0	1
16	445	222.5	255	0	0	1
17	450	217.5	255	0	0	1
18	455	212.5	255	0	0	1
19	460	207.5	255	0	0	1

## Состояние игры к концу выполнения шага

Теперь при нажатии на значок download выгружается файл:



## Состояние файлов к концу выполнения шага

### index.html

```
/ping-pong/index.html
<!DOCTYPE html>
<html>

<head>
    <title>Ping-Pong JavaScript</title>
    <link rel="stylesheet" href="assets/style.css">
</head>

<body>
    <!-- Рисуем игровое поле -->
    <canvas width="750" height="585" id="game"></canvas>
    </img>
    </img>
    </img>
    <h5 class="logging"></h5>
    <h1 class="datasetSize">0</h1>
    <div class="download">
        <a
download="ping_pong_dataset.csv" href='none'></a></img>
    </div>
    <h1 class="score">
        Current: <a>0</a><br />
        Best score: <a>0</a><br />
        Average: <a>0</a>
    </h1>
    <script src="js/render.js"></script>
    <script src="js/pause.js"></script>
```

```
<script src="js/controls.js"></script>
<script src="js/logging.js"></script>
<script src="js/scores.js"></script>
<script src="js/download.js"></script>
<script src="js/engine.js"></script>
</body>
</html>
```

## style.css

```
/ping-pong/assets/style.css
html,
body {
    height: 100%;
    margin: 0;
}

body {
    background: black;
    display: flex;
    align-items: center;
    justify-content: center;
}

.score {
    color: white;
    position: fixed;
    left: 100px;
    bottom: 35px;
}

.logging {
    color: white;
    position: fixed;
    left: 100px;
    top: 50px;
}

.datasetSize {
    color: white;
    position: fixed;
    left: 400px;
    top: 35px;
}

.download {
    position: fixed;
    cursor: pointer;
    left: 250px;
    top: 50px;
}
```

# Воркшоп 3

## Шаг 10. Интерфейс для искусственного интеллекта

### Инструкции по выполнению шага

За предыдущие шаги мы сделали хорошую игру, в которую можно поиграть самому. Чего-то не хватает? Наверное, "интеллекта"? Давайте этим и займемся!

"Искусственный интеллект" мы создадим на основе нейронной сети. Приведем одно (из многочисленных) определений:

**Нейронная сеть — математическая модель**, а также её программное или аппаратное воплощение, построенная по принципу организации и функционирования биологических нейронных сетей — сетей нервных клеток живого организма.

Важным словом, которое выделено, является **модель** нейронной сети. Без модели наш искусственный интеллект на основе нейронной сети не заведется.

Модель нейронной сети описывает её архитектуру и конфигурацию, а также используемые алгоритмы обучения.

- *Архитектура нейронной сети* определяет общие принципы её построения (плоскослоистая, полносвязная, слабосвязная, прямого распространения, рекуррентная и т.д.).
- *Конфигурация* конкретизирует структуру сети в рамках заданной архитектуры: число нейронов, число входов и выходов сети, используемые активационные функции.

Более того, нам нужно каким-то образом запускать модель нейронной сети запускать... И желательно в браузере. Для этого мы воспользуемся ONNX.js.

[ONNX.js](#) - это библиотека JavaScript для запуска моделей ONNX в браузерах и на Node.js.

Почему модели ONNX? [ONNX](#) (формат обмена открытыми нейронными сетями) является открытым стандартом для представления моделей машинного обучения. Самым большим преимуществом ONNX является то, что он обеспечивает совместимость между различными фреймворками искусственного интеллекта с открытым исходным кодом, что само по себе обеспечивает большую гибкость при внедрении фреймворков искусственного интеллекта.

При этом ONNX.js позволит запускать нам интерактивно модели нейронных сетей без установки и независимо от устройства, задержки связи между сервером и клиентом.

Давайте внедрять ONNX.js в нашу замечательную игру!

Сначала, мы добавим `onnx.min.js`, взятый из публичного репозитория, в папку `js`.

Далее мы добавим кнопку, которая позволит нам загружать модели нейронных сетей в формате ONNX в браузерное окружение.

Кнопку сделаем из двух ассетов: `upload1.svg` и `upload2.svg`.

Две картинки мы будем использовать для анимации. Когда модель будет успешно загружена в браузерное окружение, кнопка будет менять картинку через равные промежутки времени.

Добавим `upload1.svg` и `upload2.svg` в папку наших `assets`.

Далее нам необходимо добавить нашу кнопку в игру. Для этого внесём следующие изменения в наш главный файл `index.html` сразу после элемента `h1` (информация про способности игрока):

```
/ping-pong/index.html
<!-- Начало места, которое мы изменяем -->
</h1>
<p class='upload'>
  <label for="file_input">
    </img>
    </img>
    ONNX file
  </label>
  <input type="file" id='file_input' class="modelFile" accept=".onnx" />
</p>
<script src="js/render.js"></script>
<!-- Конец редактирования -->
```

Теперь опишем механизмы, которые будут реализовывать искусственный интеллект. Для этого создадим в папке `js` новый скрипт `ai.js`. Не забудем подгрузить его в `index.html` перед скриптом `engine.js`. За ним добавим `onnx.min.js`:

```
/ping-pong/index.html
<!-- Начало места, которое мы изменяем -->
<script src="js/download.js"></script>
<script src="js/onnx.min.js"></script>
<script src="js/ai.js"></script>
<script src="js/engine.js"></script>
</body>

</html>
<!-- Конец редактирования -->
```

Вот так будет выглядеть наша папка проекта после создания файла `ai.js`:

```
ping-pong
├── index.html
└── assets
    └── ball.png
```

```
    ├── paddle.png
    ├── background.jpg
    ├── style.css
    ├── download.svg
    └── upload1.svg
       └── upload2.svg
js
    ├── render.js
    ├── engine.js
    ├── controls.js
    ├── pause.js
    ├── scores.js
    ├── logging.js
    ├── download.js
    ├── onnx.min.js
    └── ai.js
```

Положение только что созданного элемента необходимо зафиксировать, поэтому внесём в файл `style.css` следующие настройки:

#### INFO

Следующая часть добавляется в конец файла `style.css`.

```
/ping-pong/assets/style.css
.upload {
  color: white;
  position: fixed;
  right: 100px;
  top: 35px;
}

.upload input {
  display: none;
}

.upload label:hover {
  cursor: pointer;
}
```

Таким образом, мы привязали нашу кнопку к конкретному месту на экране с помощью CSS селектора.

Остается добавить логику для загрузки модели и реализовать анимацию в случае успешной загрузки модели. Приступим!

В файле `ai.js` напишем следующий код:

```
/ping-pong/js/ai.js
const botImages = Array.from(
  document.getElementsByClassName("upload") [0].getElementsByTagName("img")
);

var onnxSess;
var use_bot = {
  state: false,
  busy: false,
  intervalId: 0
};
```

```

// https://stackoverflow.com/questions/52184291/async-await-with-setinterval
async function waitUntil(condition) {
    let frame = 0;
    return await new Promise((resolve) => {
        condition.intervalId = setInterval(() => {
            if (!condition.state) {
                resolve("a");
            } else {
                if (!condition.busy && !isPaused) {
                    if (frame === 0) {
                        botImages[0].style.display = "block";
                        botImages[1].style.display = "none";
                        const tmp = botImages[0];
                        botImages[0] = botImages[1];
                        botImages[1] = tmp;
                    }
                    frame = (frame + 1) % 10;
                }
            }
        }, 50);
    });
}

document.getElementsByClassName("modelFile")[0].onchange = async function (
    event
) {
    var fileList = this.files;
    use_bot.state = false;
    if (use_bot.intervalId) {
        clearInterval(use_bot.intervalId);
    }
    if (!fileList.length) {
        return;
    }

    // breakLoop = true;
    let file = fileList[0];
    let reader = new FileReader();
    reader.onloadend = async function () {
        onnxSess = new onnx.InferenceSession();
        await onnxSess.loadModel(reader.result);
        use_bot.state = true;
        waitUntil(use_bot);
    };
    reader.readAsDataURL(file);
    console.log(fileList[0]);
};

```

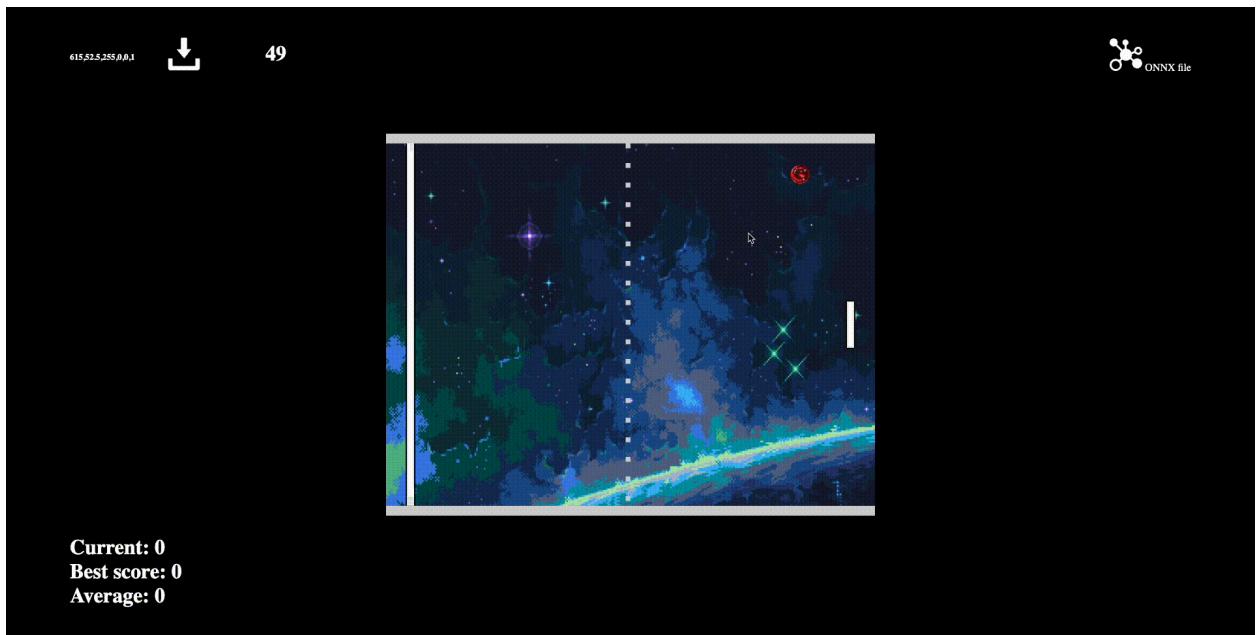
Давайте обсудим то, что делает написанные код:

1. В начале мы на странице обнаружили нашу кнопку и получаем ее картинку в переменную `botImages`.
2. Далее мы создали именованный массив `use_bot`, куда будем сохранять признак загрузки модели, занятости и информацию для реализации интервальной анимации.
3. Метод `waitUntil` позволяет организовать интервальную анимацию кнопки и кое-какую другую тактовую деятельность, о которой пойдет речь на следующем шаге.
4. Когда мы нажимаем на кнопку, появляется окно загрузки файла с локального диска. Мы в `index.html` указали, что ждем файлы с расширением `.onnx`. Когда мы

выбрали модель или закрыли окно, мы приостанавливаем анимацию кнопки, если она была запущена. Далее мы проверяем, был ли передан файл в ожидаемом формате. Если да, то осуществим чтением модели из файла и запустим анимацию кнопки.

## Состояние игры к концу выполнения шага

Теперь у нас имеется кнопка загрузки модели, на которую можно нажать, и что-то произойдет:



## Шаг 11. Обработчик искусственного интеллекта

### Инструкции по выполнению шага

На прошлом шаге была добавлена кнопка с загрузкой модели на главную страницу игры, привязана к конкретному месту на экране с помощью CSS селектора, добавлена анимация кнопки при успешной загрузке модели. Наконец, созданы скрипты с чтением модели из файла при нажатии на кнопку. То есть на данный момент имеется вся механика интерфейса подключения нейросетевой модели, но пока модель не контролирует ракетку вместо игрока. На данном шаге предлагается добавить возможность следующие функции в игру:

- А. Базовая: контроль ракетки за игрока
- В. Основная: игра против игрока левой ракеткой
- С. Продвинутая: зеркальная игра ботов друг с другом

## Функционалы

### Функционал А

## Инструкции по реализации функционала

Для начала давайте добавим функцию прогноза нейронной сети в ранее написанный скрипт `ai.js`:

#### INFO

Следующая часть кода добавляем в конец файла `ai.js`.

```
/ping-pong/js/ai.js
const normalizationConstant = 705;
// Предсказание нейронной сеткой
const runONNX = async () => {
  use_bot.busy = true;
  console.time("onnx");
  var inp = Float32Array.from([
    ball.x / normalizationConstant,
    ball.y / normalizationConstant,
    rightPaddle.y / normalizationConstant
  ]);
  let input = new onnx.Tensor(inp, "float32", [1, 3]);
  let output = (await onnxSess.run([input])).get("output").data;
  const actionId = indexOfMax(output);
  if (actionId === 2) {
    // up
    keyPresses["up"] = 1;
    keyPresses["down"] = 0;
    keyPresses["nothing"] = 0;
    rightPaddle.dy = -rightPaddle.paddleSpeed;
  } else if (actionId === 1) {
    // down
    keyPresses["up"] = 0;
    keyPresses["down"] = 1;
    keyPresses["nothing"] = 0;
    rightPaddle.dy = rightPaddle.paddleSpeed;
  } else {
    //nothing
    keyPresses["up"] = 0;
    keyPresses["down"] = 0;
    keyPresses["nothing"] = 1;
    rightPaddle.dy = 0;
  }
}
```

```
    console.timeEnd("onnx");
    use_bot.busy = false;
    return output;
};
```

Разберём по шагам, что тут происходит. Вначале мы вводим константу `normalizationConstant`, которую использовали при нормализации данных в ноутбуке, когда обучали нейронную сеть на собранном наборе данных. Её значение на самом деле не превышает координату правой ракетки, поскольку все вылеты за пределы поля удаляются из набора данных при его сборе. Поэтому вместо такого "магического числа" (вы, может, удивитесь, но это реальный термин у программистов) хорошо было бы написать `rightPaddle.x`. И на практике так и стоит поступать. Потому что если мы захотим изменить размеры холста, то это скажется и на горизонтальной координате правой ракетки, и на координатах в обучающем наборе. Затем при обучении модели мы отшкалируем диапазон координат мячика от 0 до 1, поделив на максимальную его координату из набора. А эта координата уже будет отличаться от зашитой нами сейчас в коде. В результате данные будут неправильно подготавливаться перед подачей их на вход в нейронную сеть, в результате чего предсказания модели могут быть катастрофически неправильными. А догадаться вы об этом сходу не сможете, если давно писали этот участок кода и на выявление ошибки может уйти продолжительное время. Вот почему сообщество программистов зачастую так не любит "магические числа" в коде.

Но вернёмся к нашим баранам. После объявления нормирующей константы объявляется функция прямого прохода по нейронной сети `runONNX`. Она асинхронная, что можно понять по приставке `async`, потому что это одно из необходимых требований для работы с нейросетевым фреймворком ONNX.js. Затем мы выставляем флаг занятости бота в истинное значение. Зачем мы это делаем? Всё потому, что ранее созданная сессия, в которую был загружен граф нейронной сети, не может обрабатывать больше одного асинхронного запроса на прогноз в один момент времени. Кроме того, это лишняя нагрузка на процессор, поскольку нет необходимости делать прогнозы с такой большой частотой.

После выставления флага занятости мы устанавливаем отладочный таймер с меткой `onnx` при помощи метод `console.time`. В дальнейшем при запуске можно будет увидеть время прогноза действия нейронной сеткой в миллисекундах.

Затем создаётся типизированный входной массив из нормализованных значений координат мячика и вертикальной координаты правой ракетки. Этот типизированный массив подаётся на вход в специальную структуру данных фреймворка ONNX.js – `Tensor`. На вход эта структура принимает типизированный массив, тип массива и его размерность. Размерность массива это число примеров на число признаков. Поэтому указывается `[1, 3]`. На следующей строчке и происходит всё волшебство предсказания при помощи функции `run`. На вход ей подаётся входная структура `Tensor`, она её прогоняет через нейронную сеть и

"выплёвывает" значения нейронов выходного слоя. Поскольку нам не интересны сами числа, а интересен лишь номер нейрона-победителя, то мы передаём результат в функцию `indexOfMax`. Она возвращает номер нейрона с максимальным значением. Это нестандартная функция из JavaScript и её надо реализовать:

## INFO

Следующая часть кода добавляется перед `const normalizationConstant = 705;` файла `ai.js`.

```
// Получение позиции максимального элемента в массиве
const indexOfMax = (arr) => {
    if (arr.length === 0) {
        return -1;
    }

    var max = arr[0];
    var maxIndex = 0;

    for (var i = 1; i < arr.length; i++) {
        if (arr[i] > max) {
            maxIndex = i;
            max = arr[i];
        }
    }
    return maxIndex;
};
```

Когда мы получили номер нейрона-победителя, мы должны использовать эту информацию, чтобы бот совершил то или иное действие. Поскольку возможных действий не так много, то делаем это при помощи операторов ветвления. Тут стоит не забыть, что номер нейрона начинает свой отсчёт не с единицы, а с нуля. В зависимости от номера предпринимаются следующие манипуляции с ракеткой:

- Номер 2 – устанавливаем отрицательное (с учётом, что рост вертикальной координаты идёт сверху вниз) значение вертикальной скорости;
- Номер 1 – устанавливаем положительное (с учётом, что рост вертикальной координаты идёт сверху вниз) значение вертикальной скорости;
- Номер 0 – устанавливаем нулевое (с учётом, что рост вертикальной координаты идёт сверху вниз) значение вертикальной скорости;

Вместе с этим не забываем обновлять информацию в словарике для логирования `keyPresses`.

Наконец, останавливаем таймер при помощи метода `console.timeEnd`. Только после этого в консоль разработчика выводится информация о затраченных миллисекундах. И последним штрихом возвращаем свободный статус боту, установив соответствующее значение в словаре.

Написанная функция `runONNX` нигде пока не вызывается. Давайте исправим это. Обновим функцию `waitFor`:

## INFO

Следующая часть кода изменяет функцию `waitFor` файла `ai.js`.

```
/ping-pong/js/ai.js
async function waitFor(condition, func) {
    let frame = 0;
    return await new Promise((resolve) => {
        condition.intervalId = setInterval(() => {
            if (!condition.state) {
                resolve("a");
            } else {
                if (!condition.busy && !isPaused) {
                    func();
                    if (frame === 0) {
                        botImages[0].style.display = "block";
                        botImages[1].style.display = "none";
                        const tmp = botImages[0];
                        botImages[0] = botImages[1];
                        botImages[1] = tmp;
                    }
                    frame = (frame + 1) % 10;
                }
            }
        }, 50);
    });
}
```

А также функцию обратного вызова при подключении файла с моделью:

## INFO

Следующая часть кода изменяет часть кода обратного вызова на событие `document.getElementsByClassName("modelFile")[0].onchange` в файле `ai.js`.

```
/ping-pong/js/ai.js
reader.onloadend = async function () {
    onnxSess = new onnx.InferenceSession();
    await onnxSess.loadModel(reader.result);
    use_bot.state = true;
    waitFor(use_bot, runONNX);
};
```

# Состояние игры к концу реализации функционала

В результате при подключении файла с нейронной сетью мы увидим, как правая ракетка управляет при помощи искусственного интеллекта:



## Состояние файлов к концу реализации функционала

### ai.js

```
/ping-pong/js/ai.js
const botImages = Array.from(
    document.getElementsByClassName("upload")[0].getElementsByTagName("img")
);

var onnxSess;
var use_bot = {
    state: false,
    busy: false,
    intervalId: 0
};

// https://stackoverflow.com/questions/52184291/async-await-with-setinterval
async function waitUntil(condition, func) {
    let frame = 0;
    return await new Promise((resolve) => {
        condition.intervalId = setInterval(() => {
            if (!condition.state) {
                resolve("a");
            } else {
                if (!condition.busy && !isPaused) {
                    func();
                    if (frame === 0) {
                        botImages[0].style.display = "block";
                        botImages[1].style.display = "none";
                        const tmp = botImages[0];
                        botImages[0] = botImages[1];
                        botImages[1] = tmp;
                    }
                }
            }
        }, 10);
    });
}
```

```

        }
        frame = (frame + 1) % 10;
    }
},
}, 50);
});

}

document.getElementsByClassName("modelFile")[0].onchange = async function (
    event
) {
    var fileList = this.files;
    use_bot.state = false;
    if (use_bot.intervalId) {
        clearInterval(use_bot.intervalId);
    }
    if (!fileList.length) {
        return;
    }

    // breakLoop = true;
    let file = fileList[0];
    let reader = new FileReader();
    reader.onloadend = async function () {
        onnxSess = new onnx.InferenceSession();
        await onnxSess.loadModel(reader.result);
        use_bot.state = true;
        waitUntil(use_bot, runONNX);
    };
    reader.readAsDataURL(file);
    console.log(fileList[0]);
};

// Получение позиции максимального элемента в массиве
const indexOfMax = (arr) => {
    if (arr.length === 0) {
        return -1;
    }

    var max = arr[0];
    var maxIndex = 0;

    for (var i = 1; i < arr.length; i++) {
        if (arr[i] > max) {
            maxIndex = i;
            max = arr[i];
        }
    }
    return maxIndex;
};

const normalizationConstant = 705;
// Предсказание нейронной сеткой
const runONNX = async () => {
    use_bot.busy = true;
    console.time("onnx");
    var inp = Float32Array.from([
        ball.x / normalizationConstant,
        ball.y / normalizationConstant,
        rightPaddle.y / normalizationConstant
    ]);
    let input = new onnx.Tensor(inp, "float32", [1, 3]);
    let output = (await onnxSess.run([input])).get("output").data;
    const actionId = indexOfMax(output);
}

```

```

if (actionId === 2) {
    // up
    keyPresses["up"] = 1;
    keyPresses["down"] = 0;
    keyPresses["nothing"] = 0;
    rightPaddle.dy = -rightPaddle.paddleSpeed;
} else if (actionId === 1) {
    // down
    keyPresses["up"] = 0;
    keyPresses["down"] = 1;
    keyPresses["nothing"] = 0;
    rightPaddle.dy = rightPaddle.paddleSpeed;
} else {
    //nothing
    keyPresses["up"] = 0;
    keyPresses["down"] = 0;
    keyPresses["nothing"] = 1;
    rightPaddle.dy = 0;
}
console.timeEnd("onnx");
use_bot.busy = false;
return output;
};


```

## Функционал В

### Инструкции по реализации функционала

Чтобы теперь задействовать левую ракетку, которая до этого момента была заколдована в виде стены, допишем следующие три строчки в функцию обратного вызова при загрузке модели в скрипте ai.js:

#### INFO

Следующая часть кода изменяет часть кода обратного вызова на событие `document.getElementsByClassName("modelFile")[0].onchange` в файле ai.js.

[/ping-pong/js/ai.js](#)

```

reader.onloadend = async function () {
    onnxSess = new onnx.InferenceSession();
    await onnxSess.loadModel(reader.result);
    use_bot.state = true;
    waitUntil(use_bot, runONNX);
    leftPaddle.width = rightPaddle.width;
    leftPaddle.height = rightPaddle.height;
    leftPaddle.y = rightPaddle.y;
};

```

Таким образом, ширина, толщина и начальная позиция левой ракетки станет аналогичной правой ракетке.

Теперь нужно заменить некоторые входные данные в функции runONNX. Отзеркалим горизонтальную координату мячика, а вместо координаты правой ракетки будем подавать на вход координату левой:

## INFO

Следующая часть кода изменяет часть кода в функции runONNX в файле ai.js.

/ping-pong/js/ai.js

```
var inp = Float32Array.from([
    1 - ball.x / normalizationConstant,
    ball.y / normalizationConstant,
    leftPaddle.y / normalizationConstant
]);
```

В теле операторов ветвления также заменяем правую ракетку на левую:

## INFO

Следующая часть кода изменяет часть кода в функции runONNX в файле ai.js.

/ping-pong/js/ai.js

```
if (actionId === 2) {
    // up
    leftPaddle.dy = -leftPaddle.paddleSpeed;
} else if (actionId === 1) {
    // down
    leftPaddle.dy = leftPaddle.paddleSpeed;
} else {
    //nothing
    leftPaddle.dy = 0;
}
```

Завершающим штрихом является добавление логики движений левой ракетки со стенками в движок игры скрипта engine.js:

## INFO

Следующую часть кода добавляем между

строчками updateDataset(); и rightPaddle.y += rightPaddle.dy; в

функции loop файла engine.js.

/ping-pong/js/engine.js

```
updateDataset();

if (use_bot.state) {
    leftPaddle.y += leftPaddle.dy;
    // Если правая платформа пытается вылезти за игровое поле вниз,
    if (leftPaddle.y < grid) {
        // то оставляем её на месте
        leftPaddle.y = grid;
    }
}
```

```

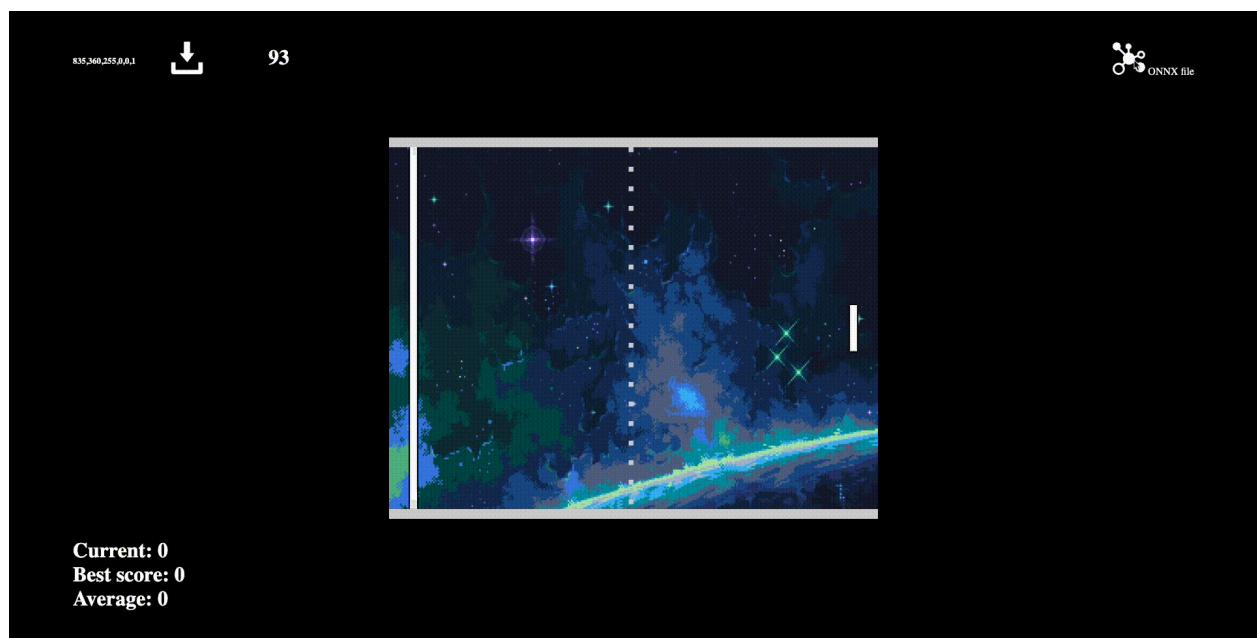
    // Проверяем то же самое сверху
    else if (leftPaddle.y > LeftmaxPaddleY) {
        leftPaddle.y = LeftmaxPaddleY;
    }
}

// Если платформы на предыдущем шаге куда-то двигались — пусть продолжают
// двигаться
rightPaddle.y += rightPaddle.dy;

```

## Состояние игры к концу реализации функционала

Отлично! Теперь при подключении модели, левая ракетка начинает двигаться вслед за мячом. По сути вы теперь играете с цифровым двойником самого **себя**:



## Состояние файлов к концу реализации функционала

### ai.js

```

/ping-pong/js/ai.js
const botImages = Array.from(
  document.getElementsByClassName("upload")[0].getElementsByTagName("img"))
);

```

```

var onnxSess;
var use_bot = {
  state: false,
  busy: false,
  intervalId: 0
};

// https://stackoverflow.com/questions/52184291/async-await-with-setinterval
async function waitUntil(condition, func) {
  let frame = 0;
  return await new Promise((resolve) => {
    condition.intervalId = setInterval(() => {
      if (!condition.state) {
        resolve("a");
      } else {
        if (!condition.busy && !isPaused) {
          func();
          if (frame === 0) {
            botImages[0].style.display = "block";
            botImages[1].style.display = "none";
            const tmp = botImages[0];
            botImages[0] = botImages[1];
            botImages[1] = tmp;
          }
          frame = (frame + 1) % 10;
        }
      }
    }, 50);
  });
}

document.getElementsByClassName("modelFile")[0].onchange = async function (
  event
) {
  var fileList = this.files;
  use_bot.state = false;
  if (use_bot.intervalId) {
    clearInterval(use_bot.intervalId);
  }
  if (!fileList.length) {
    return;
  }

  // breakLoop = true;
  let file = fileList[0];
  let reader = new FileReader();
  reader.onloadend = async function () {
    onnxSess = new onnx.InferenceSession();
    await onnxSess.loadModel(reader.result);
    use_bot.state = true;
    waitUntil(use_bot, runONNX);
    leftPaddle.width = rightPaddle.width;
    leftPaddle.height = rightPaddle.height;
    leftPaddle.y = rightPaddle.y;
  };
  reader.readAsDataURL(file);
  console.log(fileList[0]);
};

const indexOfMax = (arr) => {
  if (arr.length === 0) {
    return -1;
  }
}

```

```

var max = arr[0];
var maxIndex = 0;

for (var i = 1; i < arr.length; i++) {
    if (arr[i] > max) {
        maxIndex = i;
        max = arr[i];
    }
}
return maxIndex;
};

const normalizationConstant = 705;
// Предсказание нейронной сеткой
const runONNX = async () => {
    use_bot.busy = true;
    console.time("onnx");
    var inp = Float32Array.from([
        1 - ball.x / normalizationConstant,
        ball.y / normalizationConstant,
        leftPaddle.y / normalizationConstant
    ]);
    let input = new onnx.Tensor(inp, "float32", [1, 3]);
    let output = (await onnxSess.run([input])).get("output").data;
    const actionId = indexOfMax(output);
    if (actionId === 2) {
        // up
        leftPaddle.dy = -leftPaddle.paddleSpeed;
    } else if (actionId === 1) {
        // down
        leftPaddle.dy = leftPaddle.paddleSpeed;
    } else {
        //nothing
        leftPaddle.dy = 0;
    }
    console.timeEnd("onnx");
    use_bot.busy = false;
    return output;
};

```

## engine.js

/ping-pong/js/engine.js

```

// Проверка на то, пересекаются два объекта с известными координатами или нет
// Подробнее тут: https://developer.mozilla.org/en-US/docs/Games/Techniques/2D_collision_detection
const collides = (obj1, obj2) => {
    return (
        obj1.x < obj2.x + obj2.width &&
        obj1.x + obj1.width > obj2.x &&
        obj1.y < obj2.y + obj2.height &&
        obj1.y + obj1.height > obj2.y
    );
};

// Главный цикл игры
const loop = () => {
    // Сохраняем текущие значения объектов
    updateDataset();

```

```

if (use_bot.state) {
    leftPaddle.y += leftPaddle.dy;
    // Если правая платформа пытается вылезти за игровое поле вниз,
    if (leftPaddle.y < grid) {
        // то оставляем её на месте
        leftPaddle.y = grid;
    }
    // Проверяем то же самое сверху
    else if (leftPaddle.y > LeftmaxPaddleY) {
        leftPaddle.y = LeftmaxPaddleY;
    }
}

// Если платформы на предыдущем шаге куда-то двигались — пусть продолжают
// двигаться
rightPaddle.y += rightPaddle.dy;

// Если правая платформа пытается вылезти за игровое поле вниз,
if (rightPaddle.y < grid) {
    // то оставляем её на месте
    rightPaddle.y = grid;
}
// Проверяем то же самое сверху
else if (rightPaddle.y > RightmaxPaddleY) {
    rightPaddle.y = RightmaxPaddleY;
}

// Если мяч на предыдущем шаге куда-то двигался — пусть продолжает
// двигаться
ball.x += ball.dx;
ball.y += ball.dy;
// Если мяч касается стены снизу — меняем направление по оси Y на
// противоположное
if (ball.y < grid) {
    ball.y = grid;
    ball.dy *= -1;
}
// Делаем то же самое, если мяч касается стены сверху
else if (ball.y + grid > canvas.height - grid) {
    ball.y = canvas.height - grid * 2;
    ball.dy *= -1;
}
// Если мяч улетел за игровое поле влево или вправо — перезапускаем его
if ((ball.x < 0 || ball.x > canvas.width) && !ball.resetting) {
    // Помечаем, что мяч перезапущен, чтобы не зациклился
    ball.resetting = true;
    // Даём секунду на подготовку игрокам
    setTimeout(() => {
        // Всё, мяч в игре
        ball.resetting = false;
        // Снова запускаем его из центра
        ball.x = canvas.width / 2;
        ball.y = canvas.height / 2;
        rightPaddle.x = canvas.width - grid * 3;
        rightPaddle.y = canvas.height / 2 - paddleHeight / 2;

        // Вырезаем плохой фрагмент из датасета
        recordData = recordData.slice(0, lastGoodMove);

        // Обновляем доску с рекордами
        updateStatus(true);
    }, 1000);
}

```

```

// Если мяч коснулся левой платформы,
if (collides(ball, leftPaddle)) {
    // то отправляем его в обратном направлении
    ball.dx *= -1;
    // Увеличиваем координаты мяча на ширину платформы, чтобы не
    засчитался новый отскок
    ball.x = leftPaddle.x + leftPaddle.width;
}
// Проверяем и делаем то же самое для правой платформы
else if (collides(ball, rightPaddle)) {
    ball.dx *= -1;
    ball.x = rightPaddle.x - ball.width;

    // Обновляем доску с рекордами
    updateStatus(false);
    // Сохраняем временную позицию последнего удачно отбитого мячика
    lastGoodMove = recordData.length;
}

// Отрисовываем новый кадр
redraw();
};

// Запускаем игру
function Start() {
    if (!isPaused) {
        loop();
    }
    // Рекурсивный вызов игрового движка
    requestAnimFrame(Start);
}

Start();
loop();

```

## Функционал С

### Инструкции по реализации функционала

Поскольку код для игры за левую ракетку и игру за правую уже был написан ранее, то интересным развитием игры будет натравливание ботов друг против друга. Для достижения этого эффекта необходимо внести следующие изменения в функцию `runONNX` скрипта `ai.js`:

#### INFO

Следующая часть кода отображает изменения функции `runONNX` скрипта `ai.js`.

`/ping-pong/js/ai.js`

`// Предсказание нейронной сеткой`

`const runONNX = async (left = false) => {`

```

use_bot.busy = true;
console.time("onnx");
var inp = Float32Array.from([
    left ? (1 - ball.x / normalizationConstant + 30 /
normalizationConstant) : (ball.x / normalizationConstant),
    ball.y / normalizationConstant,
    left ? (leftPaddle.y / normalizationConstant) : (rightPaddle.y / normalizationConstant)
]);
let input = new onnx.Tensor(inp, "float32", [1, 3]);
let output = (await onnxSess.run([input])).get("output").data;
const actionId = indexOfMax(output);
if (actionId === 2) {
    // up
    if (left) {
        leftPaddle.dy = -leftPaddle.paddleSpeed;
    } else {
        keyPresses["up"] = 1;
        keyPresses["down"] = 0;
        keyPresses["nothing"] = 0;
        rightPaddle.dy = -rightPaddle.paddleSpeed;
    }
} else if (actionId === 1) {
    // down
    if (left) {
        leftPaddle.dy = leftPaddle.paddleSpeed;
    } else {
        keyPresses["up"] = 0;
        keyPresses["down"] = 1;
        keyPresses["nothing"] = 0;
        rightPaddle.dy = rightPaddle.paddleSpeed;
    }
} else {
    //nothing
    if (left) {
        leftPaddle.dy = 0;
    } else {
        keyPresses["up"] = 0;
        keyPresses["down"] = 0;
        keyPresses["nothing"] = 1;
        rightPaddle.dy = 0;
    }
}
console.timeEnd("onnx");
use_bot.busy = false;
return output;
};

```

Мы добавили аргумент функции `left` со значением по умолчанию `false`. Предполагается, что данная функция сможет делать прогнозы для дальнейших действий как для левой ракетки, так и для правой.

Остальные изменения затрагивают правильное формирование входных данных и запись требуемых значений для выбранной ракетки. Для уменьшения числа строк кода используется тернарный оператор – краткая запись оператора ветвления. Она имеет синтаксис:

(логическое выражение) ? значение\_если\_истина : значение\_если\_ложь

Последним действием необходимо прогонять модель через для состояний левой и правой ракеток. Поэтому немного перепишем функцию `waitUntil`:

## INFO

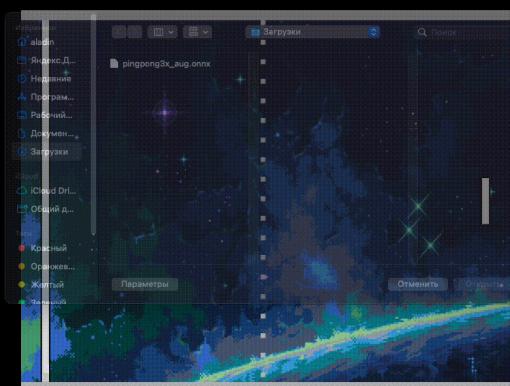
Следующая часть кода отображает изменения функции `waitUntil` скрипта `ai.js`.

`/ping-pong/js/ai.js`

```
async function waitUntil(condition, func) {
    let frame = 0;
    return await new Promise((resolve) => {
        condition.intervalId = setInterval(async () => {
            if (!condition.state) {
                resolve("a");
            } else {
                if (!condition.busy && !isPaused) {
                    await func(false);
                    await func(true);
                    if (frame === 0) {
                        botImages[0].style.display = "block";
                        botImages[1].style.display = "none";
                        const tmp = botImages[0];
                        botImages[0] = botImages[1];
                        botImages[1] = tmp;
                    }
                    frame = (frame + 1) % 10;
                }
            }
        }, 50);
    });
}
```

# Состояние игры к концу реализации функционала

Попробуйте запустить игру и подключить файл с обученной нейронной сетью. Если всё сделано правильно, вы увидите, как две ракетки автоматически управляются искусственным интеллектом и играют друг с другом.



Current: 0  
Best score: 0  
Average: 0

## Состояние файлов к концу реализации функционала

### ai.js

```
/ping-pong/js/ai.js
const botImages = Array.from(
    document.getElementsByClassName("upload")[0].getElementsByTagName("img")
);

var onnxSess;
var use_bot = {
    state: false,
    busy: false,
    intervalId: 0
};

// https://stackoverflow.com/questions/52184291/async-await-with-setinterval
async function waitUntil(condition, func) {
    let frame = 0;
    return await new Promise((resolve) => {
        condition.intervalId = setInterval(async () => {
            if (!condition.state) {
                resolve("a");
            } else {
                if (!condition.busy && !isPaused) {
                    await func(false);
                    await func(true);
                    if (frame === 0) {
                        botImages[0].style.display = "block";
                        botImages[1].style.display = "none";
                        const tmp = botImages[0];
                        botImages[0] = botImages[1];
                        botImages[1] = tmp;
                    }
                }
            }
        }, 100);
    });
}
```

```

                botImages[1] = tmp;
            }
            frame = (frame + 1) % 10;
        }
    },
    50);
});
}

document.getElementsByClassName("modelFile")[0].onchange = async function (
event
) {
    var fileList = this.files;
    use_bot.state = false;
    if (!use_bot.intervalId) {
        clearInterval(use_bot.intervalId);
    }
    if (!fileList.length) {
        return;
    }

    // breakLoop = true;
    let file = fileList[0];
    let reader = new FileReader();
    reader.onloadend = async function () {
        onnxSess = new onnx.InferenceSession();
        await onnxSess.loadModel(reader.result);
        use_bot.state = true;
        waitUntil(use_bot, runONNX);
        leftPaddle.width = rightPaddle.width;
        leftPaddle.height = rightPaddle.height;
        leftPaddle.y = rightPaddle.y;
    };
    reader.readAsDataURL(file);
    console.log(fileList[0]);
};

// Получение позиции максимального элемента в массиве
const indexOfMax = (arr) => {
    if (arr.length === 0) {
        return -1;
    }

    var max = arr[0];
    var maxIndex = 0;

    for (var i = 1; i < arr.length; i++) {
        if (arr[i] > max) {
            maxIndex = i;
            max = arr[i];
        }
    }
    return maxIndex;
};

const normalizationConstant = 705;
// Предсказание нейронной сеткой
const runONNX = async (left = false) => {
    use_bot.busy = true;
    console.time("onnx");
    var inp = Float32Array.from([
        left ? (1 - ball.x / normalizationConstant + 30 /
normalizationConstant) : (ball.x / normalizationConstant),
        ball.y / normalizationConstant,
    ]);
}

```

```

        left ? (leftPaddle.y / normalizationConstant) : (rightPaddle.y /
normalizationConstant)
    ]);
let input = new onnx.Tensor(inp, "float32", [1, 3]);
let output = (await onnxSess.run([input])).get("output").data;
const actionId = indexOfMax(output);
if (actionId === 2) {
    // up
    if (left) {
        leftPaddle.dy = -leftPaddle.paddleSpeed;
    } else {
        keyPresses["up"] = 1;
        keyPresses["down"] = 0;
        keyPresses["nothing"] = 0;
        rightPaddle.dy = -rightPaddle.paddleSpeed;
    }
} else if (actionId === 1) {
    // down
    if (left) {
        leftPaddle.dy = leftPaddle.paddleSpeed;
    } else {
        keyPresses["up"] = 0;
        keyPresses["down"] = 1;
        keyPresses["nothing"] = 0;
        rightPaddle.dy = rightPaddle.paddleSpeed;
    }
} else {
    //nothing
    if (left) {
        leftPaddle.dy = 0;
    } else {
        keyPresses["up"] = 0;
        keyPresses["down"] = 0;
        keyPresses["nothing"] = 1;
        rightPaddle.dy = 0;
    }
}
console.timeEnd("onnx");
use_bot.busy = false;
return output;
};

```