Санкт-Петербургский политехнический университет Петра Великого

 Институт прикладной математики и механики

 **Кафедра «Теоретическая механика»**

 **КУРСОВОЙ ПРОЕКТ**

 **Графический редактор «Paint»**

 по дисциплине «Основы алгоритмизации и программирования»

Выполнили

студент гр. 3630103/90003 Михайлов М. А.

студент гр. 3630103/90003 Шульга М. Е.

«\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2020 г.

 Санкт-Петербург

2020

**СОДЕРЖАНИЕ**

|  |  |
| --- | --- |
| Введение . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .  | 2 |
| Постановка задачи . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .  | 2 |
| 1. Описание кода. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .  | 4 |
|  1.1 HTML . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .  | 4 |
|  1.2 CSS . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .  | 4 |
|  1.3 JavaScript . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .  | 4 |
| Результаты . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .  | 19 |
| Заключение . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .  | 20 |
| Литература . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .  | 21 |

**Введение**

Данная работа посвящена реализации прототипа графического редактора «Paint». Paint- многофункциональный, но в то же время довольно простой в использовании [растровый графический редактор](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D1%81%D1%82%D1%80%D0%BE%D0%B2%D1%8B%D0%B9_%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D1%80%D0%B5%D0%B4%D0%B0%D0%BA%D1%82%D0%BE%D1%80) компании [Microsoft](https://ru.wikipedia.org/wiki/Microsoft), входящий в состав всех операционных систем [Windows](https://ru.wikipedia.org/wiki/Windows), начиная с первых версий. Данный проект нужен нам для возможности применить на практике знания, полученные на лекциях. Предмет исследования – программные инструменты языков JavaScript, HTML и CSS.

**Постановка задачи**

Реализовать прототип графического редактора «Paint» на языке JavaScript со следующими инструментами:

1. «Кисть» - инструмент, отвечающий за рисование кривых линий с возможностью выбора цвета и толщины.

2. «Ластик» - инструмент, отвечающий за очищение выбранной пользователем области холста.

3. «Сбросить всё» - инструмент, отвечающий за мгновенное очищение всего холста.

4. «Ломанная» - инструмент, отвечающий за рисование прямых линий с возможностью выбора цвета и толщины.

5. «Заливка» - инструмент, отвечающий за заливку определённым цветом выбранной пользователем области.

6. «Сохранить» - инструмент, отвечающий за сохранение рисунка отдельным файлом.

7 «Пипетка» - инструмент, отвечающий за изменение текущего цвета на цвет области, на которую нажал пользователь.

8. «Фигуры» - набор инструментов, отвечающих за рисование следующих типов фигур в двух режимах (залитая выбранным цветом/полая фигура с выбранным цветом контура):

* Квадрат
* Прямоугольник
* Эллипс
* Окружность

**1. Описание кода:**

**1.1 HTML:**

В файле HTML cоздадим холст (<canvas>) размерами 900х600 px и таблицу с элементами интерфейса программы.

**1.2 CSS:**

С помощью файла CSS составляем дизайн графического редактора, используя такие свойства как background-color, background-image.

**1.3 JavaScript:**

Зададим необходимые для работы программы переменные, определяющие начальные условия, и функции их считывания:

/\*\*

 \* Комментарии к глобальным переменным:

 \* myColor     - актуальный цвет.

 \* R           - радиус точки / толщина линии.

 \* flag        - флажок для повторного определение точек функции linedraw.

 \* mode\_flag   - флажок для смены режимов.

 \* defImg      - дефолтный background холста (для очистки холста).

 \*/

var canvas      = document.getElementById('c1');

var ctx         =       canvas.getContext('2d');

var myColor     =                     '#000000';

var R           =                             5;

var defImg      =        new Image('holst.jpg');

var flag                                       ;

var mode\_flag                                  ;

/\*\*

 \* Считывание значения ползунка.

 \*/

document.getElementById('lineWidth').oninput = function(){

   R = this.value;

}

/\*\*

 \* Выбор цвета (считывание с виджета).

 \*/

document.getElementById('color').oninput = function(){

   myColor = this.value;

}

После чего заливаем наш холст белым цветом для нормальной работы функций, считывающих код цвета с холста:

ctx.fillStyle = "#ffffff";

ctx.fillRect(0,0,900,600);

Код программы разбит на функции, каждая из которых отвечает за определённый инструмент. Рассмотрим их в отдельности:

**Сбросить всё.** В данной функции мы очищаем наш холст с помощью метода canvas *clearRect(x, y, width, height)*, принимающей параметры x, y в качестве опорной точки и width, height в качестве ширины и высоты очищаемой области. Затем возвращаем исходные настройки.

function def(){

   ctx.clearRect(0,0,900,600);

   flag = true;

   mode\_flag = false;

   ctx.fillStyle = "#ffffff";

   ctx.fillRect(0,0,900,600);

   defImg.onload = function () {

      ctx.drawImage(defImg,0,0,900,600);

   }

}

**Сохранить.** В данной функции для начала с помощью метода createElement() создаём новый элемент «а», затем вставляем этот элемент в document.body с помощью метода appendChild(). Ссылкой на этот элемент будет текущее состояние холста, которое сохраняется с помощью метода toDataURL(). Для скачивания файла в браузере используется специальный механизм download. Далее используем метод removeChild(), чтоб извлечь из document.body, созданный ранее элемент.

function saveImage(){

   const a = document.createElement("a");

   document.body.appendChild(a);

   a.href = canvas.toDataURL();

   a.download = "canvas-image/png";

   a.click();

   document.body.removeChild(a);

   console.log('you save the image');

}

Функции смены режимов определяют логические значения флага для функций рисования фигур:

/\*\*

 \* ФУНКЦИЯ СМЕНЫ РЕЖИМА НА ЗАЛИВКУ.

 \*/

function changeMode\_fill(){

   mode\_flag = true;

   console.log('change mod to fill');

}

/\*\*

 \* ФУНКЦИЯ СМЕНЫ РЕЖИМА НА ОБВОДКУ.

 \*/

function changeMode\_stroke(){

   mode\_flag = false;

   console.log('change mod to stroke');

}

**Кисть.** Для фиксирования событий нажатия мыши, её удержания и разжатия используются методы *onmousedown*, *onmousemove* и *onmouseup,* записанные как функции от события. Для рисования линии будем использовать методы canvas *arc(x, y, r, start\_angle, end\_angle, route)*, рисующий окружность радиуса r в точке (x,y) холста, *moveTo(x,y)*, перемещающий курсор в точку (x,y), и *lineTo(x1,y1)*, рисующий из положения (x,y) прямую к точке (x1,y1).

При каждом нажатии и удерживании мыши на холсте рисуется закрашенный круг, который затем, при перетаскивании зажатой мыши в другую точку, соединяется прямоугольником толщиной 2\*r с новым кругом, нарисованным уже в следующей точке. Таким образом, при перемещении зажатой мыши над холстом, рисуется линия выбранного пользователем с виджета цвета. C помощью условного оператора if написаны проверки на граничные условия.

function draw(){

   canvas.onmousedown = function(event){

      var x = event.offsetX;

      var y = event.offsetY;

      ctx.beginPath();

      ctx.arc(x, y, R, 0, 2 \* Math.PI, true);

      ctx.fillStyle = myColor ;

      ctx.fill();

      canvas.onmousemove = function(event){

         var x1 = event.offsetX;

         var y1 = event.offsetY;

         ctx.beginPath();

         ctx.arc(x1, y1, R, 0, 2 \* Math.PI, true);

         ctx.fillStyle = myColor ;

         ctx.fill();

         ctx.closePath();

         ctx.beginPath();

         ctx.moveTo(x,y);

         ctx.lineTo(x1,y1);

         ctx.lineWidth = 2\*R;

         ctx.strokeStyle = myColor;

         ctx.stroke();

         x = x1;

         y = y1;

if(y1>599){

        canvas.onmousemove = null;

}

if(x1<1){

        canvas.onmousemove = null;

}

if(y1<1){

        canvas.onmousemove = null;

}

if(x1>895){

        canvas.onmousemove = null;

}

    }

      canvas.onmouseup = function(){

         canvas.onmousemove = null;

      }

   }

}

\*Принцип работы следующих нескольких функций схож с только что рассмотренным выше, поэтому далее будут лишь заметки и описание ключевого момента.

**Ластик.** Аналогично функции кисть, но вместо переменной myColor стоит постоянный цвет white.

function eraser(){

   canvas.onmousedown = function(event){

      var x = event.offsetX;

      var y = event.offsetY;

      ctx.beginPath();

      ctx.arc(x, y, R, 0, 2 \* Math.PI, true);

      ctx.fillStyle = "#ffffff" ;

      ctx.fill();

      canvas.onmousemove = function(event){

         var x1 = event.offsetX;

         var y1 = event.offsetY;

         ctx.beginPath();

         ctx.arc(x1, y1, R, 0, 2 \* Math.PI, true);

         ctx.fillStyle = "#ffffff" ;

         ctx.fill();

         ctx.closePath();

         ctx.beginPath();

         ctx.moveTo(x,y);

         ctx.lineTo(x1,y1);

         ctx.lineWidth = 2\*R;

         ctx.strokeStyle = "#ffffff";

         ctx.stroke();

         x = x1;

         y = y1;

      if(y1>599){

        canvas.onmousemove = null;

}

if(x1<1){

        canvas.onmousemove = null;

}

if(y1<1){

        canvas.onmousemove = null;

}

if(x1>895){

        canvas.onmousemove = null;

}

     }

      canvas.onmouseup = function(){

         canvas.onmousemove = null;

      }

   }

}

**Окружность.** При нажатии мышкой на холст в локальные переменные сохраняются координаты курсора и ссылка на состояние холста в данный момент с помощью метода *toDataURL()*,которое сохраняется как изображение в переменную *backgr* с помощью метода *new Image()*. Перед началом рисования окружности холст очищается, а затем восстанавливает своё состояние на момент клика мыши по холсту с помощью метода *drawImage(data, x, y, width, height)*, куда мы передаем сохраненное изображение состояния холста, а также параметры прямоугольника, на котором будет отображена данная информация. Окружность рисуется методом *arc()*, после чего процедура повторяется при перемещении курсора. Если мышь отпустили, в старую переменную *backgr* сохраняется новая информация о состоянии холста. С помощью флажка *mode\_flag* определяется режим рисование (обводка или заливка).

/\*\*

 \* dataURL - получение ссылки на изображение холста.

 \* backgr – переменная состояния холста.

 \* ctx.drawImage(backgr,0,0,900,600); отрисовка холста на "предыдущем шаге".

 \* dataURL = canvas.toDataURL(); (в onmousup) сохраняем холст и ссылку на его изображение для следующего рисования.

 \*/

function drawCircle(){

   canvas.onmousedown = function(event){

      var dataURL = canvas.toDataURL();

      var backgr = new Image();

      backgr.src = dataURL;

      var x = event.offsetX;

      var y = event.offsetY;

      canvas.onmousemove = function(event){

         var x1 = event.offsetX;

         var y1 = event.offsetY;

         var r = Math.sqrt(Math.pow(x-x1,2)+Math.pow(y-y1,2));

         ctx.beginPath();

         ctx.clearRect(0,0,900,600);

         ctx.drawImage(backgr,0,0,900,600);

         ctx.arc(x, y, r, 0, 2 \* Math.PI, true);

         if (mode\_flag){

            ctx.fillStyle = myColor;

            ctx.fill();

         } else {

            ctx.strokeStyle = myColor;

            ctx.lineWidth = 2\*R;

            ctx.stroke();

         }

     if(y1>599){

        canvas.onmousemove = null;

}

if(x1<1){

        canvas.onmousemove = null;

}

if(y1<1){

        canvas.onmousemove = null;

}

if(x1>895){

        canvas.onmousemove = null;

}

 }

      canvas.onmouseup = function(){

         dataURL = canvas.toDataURL();

         backgr = new Image();

         backgr.src = dataURL;

         canvas.onmousemove = null;

      }

   }

}

**Эллипс.** Аналогична предыдущей. Рисование окружности заменено на рисование эллипса методом ellipse(x, y, width, height, rotation, start\_angle, end\_angle, route), принимающим параметры положения курсора (x,y) в момент нажатия мыши, параметры прямоугольника в который будет вписан, наклон относительно горизонтальной оси, начальный и конечный углы и направление увеличения угла.

function drawEllips(){

   canvas.onmousedown = function(event){

      var dataURL = canvas.toDataURL();

      var backgr = new Image();

      backgr.src = dataURL;

      var x = event.offsetX;

      var y = event.offsetY;

      canvas.onmousemove = function(event){

         var x1 = event.offsetX;

         var y1 = event.offsetY;

         var width = Math.abs(x1-x);

         var height = Math.abs(y1-y);

         ctx.beginPath();

         ctx.clearRect(0,0,900,600);

         ctx.drawImage(backgr,0,0,900,600);

         ctx.ellipse(x, y, width, height, 0, 0, 2 \* Math.PI, true);

         if (mode\_flag){

            ctx.fillStyle = myColor;

            ctx.fill();

         } else {

            ctx.strokeStyle = myColor;

            ctx.lineWidth = 2\*R;

            ctx.stroke();

         }

      if(y1>599){

        canvas.onmousemove = null;

}

if(x1<1){

        canvas.onmousemove = null;

}

if(y1<1){

        canvas.onmousemove = null;

}

if(x1>895){

        canvas.onmousemove = null;

}

     }

      canvas.onmouseup = function(){

         dataURL = canvas.toDataURL();

         backgr = new Image();

         backgr.src = dataURL;

         canvas.onmousemove = null;

      }

   }

}

**Квадрат.** Аналогично предыдущему. Метод ellipse() заменен методом *strokeRect(x, y, width, height)* для обводки и *fillRect(x, y, width, height)* для заливки прямоугольника, причем параметры width и height здесь одинаковые по модулю. Добавлены проверки для более “интуитивного” рисования.

function drawSquare(){

   canvas.onmousedown = function(event){

      var dataURL = canvas.toDataURL();

      var backgr = new Image();

      backgr.src = dataURL;

      var x = event.offsetX;

      var y = event.offsetY;

      canvas.onmousemove = function(event){

      var x1 = event.offsetX;

         var y1 = event.offsetY;

         var width = x1-x;

         var height = y-y1;

if(y1>599){

        canvas.onmousemove = null;

}

if(x1<1){

        canvas.onmousemove = null;

}

if(y1<1){

        canvas.onmousemove = null;

}

if(x1>890){

        canvas.onmousemove = null;

}

         if((y1-y<0) && (x1-x<0)){

            ctx.beginPath();

            ctx.clearRect(0,0,900,600);

            ctx.drawImage(backgr,0,0,900,600);

            if (mode\_flag){

               ctx.fillStyle = myColor;

               ctx.fillRect(x, y, -Math.abs(width), -Math.abs(width));

            } else {

               ctx.strokeStyle = myColor;

               ctx.lineWidth = 2\*R;

               ctx.strokeRect(x, y, -Math.abs(width), -Math.abs(width));

            }

         }

         if((y1-y<0) && (x1-x>0)){

            ctx.beginPath();

            ctx.clearRect(0,0,900,600);

            ctx.drawImage(backgr,0,0,900,600);

            if (mode\_flag){

               ctx.fillStyle = myColor;

               ctx.fillRect(x, y, Math.abs(width), -Math.abs(width));

            } else {

               ctx.strokeStyle = myColor;

               ctx.lineWidth = 2\*R;

               ctx.strokeRect(x, y, Math.abs(width), -Math.abs(width));

            }

         }

         if((y1-y>0) && (x1-x>0)){

            ctx.beginPath();

            ctx.clearRect(0,0,900,600);

            ctx.drawImage(backgr,0,0,900,600);

            if (mode\_flag){

               ctx.fillStyle = myColor;

               ctx.fillRect(x, y, Math.abs(width), Math.abs(width));

            } else {

               ctx.strokeStyle = myColor;

               ctx.lineWidth = 2\*R;

               ctx.strokeRect(x, y, Math.abs(width), Math.abs(width));

            }

         }

         if((y1-y>0) && (x1-x<0)){

            ctx.beginPath();

            ctx.clearRect(0,0,900,600);

            ctx.drawImage(backgr,0,0,900,600);

            if (mode\_flag){

               ctx.fillStyle = myColor;

               ctx.fillRect(x, y, -Math.abs(width), Math.abs(width));

            } else {

               ctx.strokeStyle = myColor;

               ctx.lineWidth = 2\*R;

               ctx.strokeRect(x, y, -Math.abs(width), Math.abs(width));

            }

}

}

      canvas.onmouseup = function(){

         dataURL = canvas.toDataURL();

         backgr = new Image();

         backgr.src = dataURL;

         canvas.onmousemove = null;

      }

   }

}

**Прямоугольник.** То же самое что и функция для рисования квадрата, но параметры прямоугольника (width и height) уже не обязательно равны.

function drawRect(){

   canvas.onmousedown = function(event){

      var dataURL = canvas.toDataURL();

      var backgr = new Image();

      backgr.src = dataURL;

      console.log('save parametrs of canvas');

      var x = event.offsetX;

      var y = event.offsetY;

      canvas.onmousemove = function(event){

         var x1 = event.offsetX;

         var y1 = event.offsetY;

         var width = x1-x;

         var height = y1-y;

         ctx.beginPath();

         ctx.clearRect(0,0,900,600);

         ctx.drawImage(backgr,0,0,900,600);

         if (mode\_flag){

            ctx.fillStyle = myColor;

            ctx.fillRect(x, y, width, height);

         } else {

            ctx.strokeStyle = myColor;

            ctx.lineWidth = 2\*R;

            ctx.strokeRect(x, y, width, height);

         }

    if(y1>599){

        canvas.onmousemove = null;

}

if(x1<1){

        canvas.onmousemove = null;

}

if(y1<1){

        canvas.onmousemove = null;

}

if(x1>895){

        canvas.onmousemove = null;

}

   }

      canvas.onmouseup = function(){

         dataURL = canvas.toDataURL();

         backgr = new Image();

         backgr.src = dataURL;

         canvas.onmousemove = null;

      }

   }

}

**Ломанная.** Принцип работы основан на методах canvas *moveTo()* и *lineTo()*. Переменная *flag* нужна для определения номера нажатия мыши и последующего переопределения координат для методов.

function linedraw(){

   flag = true;

   var x;

   var y;

   canvas.onmousedown = function(event){

     var dataURL = canvas.toDataURL();

      var backgr = new Image();

      backgr.src = dataURL;

    if (flag){

         x = event.offsetX;

         y = event.offsetY;

         console.log('x: ',x,'y: ',y);

      } else {

         var x1 = event.offsetX;

         var y1 = event.offsetY;

       console.log('x1: ',x1,'y1: ',y1);

         ctx.beginPath();

         ctx.moveTo(x,y);

         ctx.lineTo(x1,y1);

         ctx.lineWidth = 2\*R;

         ctx.strokeStyle = myColor;

         ctx.stroke();

         ctx.closePath();

         ctx.beginPath();

         ctx.arc(x, y, R, 0, 2 \* Math.PI, true);

         ctx.arc(x1, y1, R, 0, 2 \* Math.PI, true);

         ctx.fillStyle = myColor;

         ctx.fill();

         x = x1;

         y = y1;

      }

      canvas.onmousemove = function(event){

       var x1 = event.offsetX;

       var y1 = event.offsetY;

     console.log('x: ',x1,'y: ',y1);

     ctx.beginPath();

        ctx.clearRect(0,0,900,600);

         ctx.drawImage(backgr,0,0,900,600);

      ctx.moveTo(x,y);

         ctx.lineTo(x1,y1);

         ctx.lineWidth = 2\*R;

         ctx.strokeStyle = myColor;

         ctx.stroke();

         ctx.closePath();

       ctx.beginPath();

         ctx.arc(x, y, R, 0, 2 \* Math.PI, true);

         ctx.arc(x1, y1, R, 0, 2 \* Math.PI, true);

         ctx.fillStyle = myColor;

         ctx.fill();

if(y1>599){

        canvas.onmousemove = null;

}

if(x1<1){

        canvas.onmousemove = null;

}

if(y1<1){

        canvas.onmousemove = null;

}

if(x1>890){

        canvas.onmousemove = null;

}

     }

     canvas.onmouseup = function(){

        dataURL = canvas.toDataURL();

         backgr = new Image();

         backgr.src = dataURL;

         canvas.onmousemove = null;

       }

   }

}

**Пипетка.** Принцип работы основывается на считывании с определённого пикселя информации о его состоянии (цвете пикселя) методом *getImageData(x, y, width, height)*, где (x,y) координата, а width и height размеры области, с которой считывается информация. В нашем случае - пиксель размерами 1х1 в точке холста, над которой была нажата мышь. Затем эта информация в формате rgb передается в переменную myColor, которая отвечает за текущий выбранный цвет.

function pipette(){

   canvas.onmousedown = function(event){

      var x = event.offsetX;

      var y = event.offsetY;

      var pix\_data = ctx.getImageData(x,y,1,1).data;

      var rgb\_base = 'rgb(' + pix\_data[0] + ','+ pix\_data[1] + ','+ pix\_data[2] +')';

      var r\_rgb = pix\_data[0];

      var g\_rgb = pix\_data[1];

      var b\_rgb = pix\_data[2];

      myColor = 'rgb('+r\_rgb+','+g\_rgb+','+b\_rgb+')';

   }

}

**Заливка.** Для инструмента использовался стандартный алгоритм закрашивания области, состоящей из ячеек. Для реализации используются 3 переменных, хранящие значения цветов в формате hex для дальнейшего их сравнения. Так переменная *boardColor* отвечает за цвет фона закрашиваемой области и считывается аналогичным способом, реализованным в инструменте «пипетка», за исключением формата хранения цвета (здесь формат hex). Перевод из формата rgb в формат hex осуществляет функция *rgb\_hex(r, g, b)*, где буквы – соответствующие числовые значение выбранного на холсте цвета в формате rgb. Переменная *pixColor* нужна для хранения цвета пикселя, который бы поверялся в текущей итерации на соответствие цвету «фона» области и несовпадением с текущем цветом (*myColor*). Для ускорения реализации алгоритма осуществляется проверка каждого второго пикселя по горизонтальному и вертикальному направлению, и последующей закрашивании области размерами 3х3 px.

function filling(){

   canvas.onmousedown = function(event){

      var x = event.offsetX;

      var y = event.offsetY;

      var pix\_data = ctx.getImageData(x,y,1,1).data;

      function rgb\_hex(r, g, b){

         return "#" + ((1 << 24) + (r << 16) + (g << 8) + b).toString(16).slice(1);

      }

      var boardColor = rgb\_hex(pix\_data[0], pix\_data[1], pix\_data[2]);

      var stack = [[x,y]];

      var pixel;

      ctx.fillStyle = myColor;

      while (stack.length > 0){

         pixel = stack.pop();

         if (pixel[0] < 0 || pixel[0] >= 900){

             continue;}

         if (pixel[1] < 0 || pixel[1] >= 600){

             continue;}

         pix\_data = ctx.getImageData(pixel[0], pixel[1], 1,1).data;

         pixColor = rgb\_hex(pix\_data[0], pix\_data[1], pix\_data[2]);

         if (pixColor == boardColor && pixColor !== myColor){

            ctx.fillRect(pixel[0]-1,pixel[1]-1,3,3);

            stack.push([

               pixel[0] -2,

               pixel[1]

            ]);

            stack.push([

               pixel[0] +2,

               pixel[1]

            ]);

            stack.push([

               pixel[0],

               pixel[1] -2

            ]);

            stack.push([

               pixel[0],

               pixel[1] +2

            ]);

         }

      }

   }

}

**Результаты**

При помощи скриптового языка программирования был реализован графический редактор «Paint». Использование множества различных функций и методов дало большой опыт программирования.

Ниже представлен пример результата работы программы.

****

**Заключение**

В ходе работы большое внимание уделялось удобству пользования программы и её дизайну, чтобы любой пользователь с наслаждением создавал свои рисунки и сохранял их себе на память. Благодаря курсовой работе мы научились применять знания, полученные в ходе прослушивания лекций, на практике.

В заключении хотелось бы сказать, что JavaScript – многофункциональный язык, который позволяет выполнять задачи различной направленности и сложности.

**Литература.**

Минник, Холланд «JavaScript для чайников».

shpargalkablog.ru.

htmlbook.ru.

learn.javascript.ru.

developer.mozilla.org.

jsfiddle.net.

github.com.

html5css.ru.

cat-in-web.ru.

stackoverflow.com.

habr.com.

javascript.ru.

 **Форма задания на выполнение
 курсового проекта**

Санкт-Петербургский политехнический университет Петра Великого

**ЗАДАНИЕ**

**НА ВЫПолнение курсового проекта**

студенту группы 3630103/90003 Михайлову М. А.

студенту группы 3630103/90003 Шульге М. Е.

 ***1. Тема проекта:*** написание графического редактора «Paint»

 ***2. Срок сдачи студентом законченного проекта***

 ***3. Исходные данные к проекту***: курс лекций по основам алгоритмизации и программирования за первый и второй семестры

 ***4. Содержание пояснительной записки*** (перечень подлежащих разработке вопросов): введение, основная часть (раскрывается структура основной части), заключение, список использованных источников, приложения.

Примерный объём пояснительной записки 12 страниц печатного текста.

***5. Перечень графического материала*** (с указанием обязательных чертежей и плакатов): отсутствует

***6. Консультанты***

 ***7. Дата получения задания***: «\_\_\_».\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_ г.

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

 *(подпись) (инициалы, фамилия)*

Задание принял к исполнению \_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

 *(подпись студента) (инициалы, фамилия)*

Задание принял к исполнению \_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

 *(подпись студента) (инициалы, фамилия)*