

# Bombberman

## >> Настройка окружения.

1. Скачать темплейт проекта:
  - a. <https://github.com/kyaune/p5play-scheme>
  - b. Создать `main.js` в папке `/js` и подключить его в `index.html`
2. Установить в Atom:
  - a. сниппеты для `p5js` и `p5play`
  - b. автоформатирование `atom-beautify`. В настройках доскроллить до JavaScript и поставить галочку – `Beautify on Save`
3. Проверить работоспособность `p5js-toolbar` в Atom
  - a. Если не установлен тулбар, то установить как расширения в Atom
  - b. Должен запускаться локальный сервер
  - c. Открыть в Chrome — <http://localhost:8000/index.html>
  - d. Протестировать работоспособность через загрузку изображения из папки со спрайтами

### Важно !!

Когда вы будете работать над проектом и после сохранения `.js` файла, решите проверить результат в браузере. То возможно, что после обновления страницы ничего не изменится. Чтобы изменения отобразились используйте комбинацию клавиш `ctr + shift + r` или `shift + F5`.

Это связано с тем, что браузер кэширует старую версию и показывает её вам. Обновление страницы + `shift` – команда браузеру запросить с сервера весь сайт целиком.

## >> Создание сцены (игровое поле для bomberman)

1. В файле `main.js` выделить комметариями блок для глобальных переменных – так будет удобнее смотреть на код.
2. Объявить глобальные переменные:
  - a. для хранения файлов изображений спрайтов:

```
let stoneImg, grassImg, brickImg;
```
  - b. Имена для групп спрайтов:

```
let greenField;  
let wall;  
let bricks;
```
  - c. Количество строк, столбцов игрового поля и ширина одного блока:

```
let rows, cols, w;
```

3. Написать функцию для предзагрузки изображений, разместить между блоком с объявление глобальных переменных и сетапом:

```
function preload() {  
    stoneImg = loadImage("sprites/Blocks/SolidBlock.png");  
    ..  
}
```

4. Создать игровое поле и поместить его в div, у которого id = "game". Строку с этим div вы можете найти в index.html.

```
..  
function setup() {  
    let canvas = createCanvas(680, 520);  
    canvas.parent('game');  
    ..  
}
```

5. В сетапе определить значение  $w$  (ширина одного блока) и создать группы для спрайтов:

```
..  
w = width / cols;  
greenField = new Group();  
wall = new Group();  
bricks = new Group();  
..
```

6. Написать функцию создания игрового поля – `createScene()`:

```
..  
function setup() {  
    ..  
    createScene();  
}  
  
function draw() {  
    ..  
    drawSprites();  
}  
  
function createScene() {  
    ..  
}
```

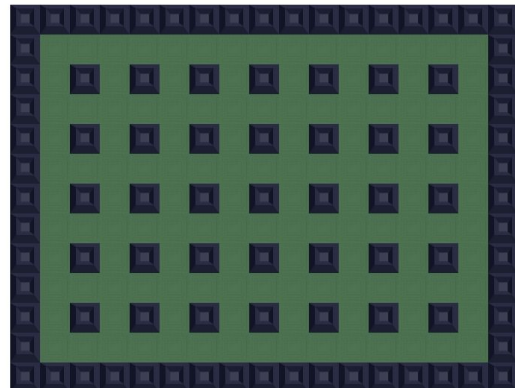


рис. 1

## Как сделать игровое поле?

1. У вас есть количество строк `rows` и столбцов `cols` – используйте их во вложенном цикле.
2. В цикле вы создаёте спрайт и определяете через индексы кем ему быть – камнем или травой. Вспомните шахматы :)
3. Добавить в спрайт изображение, изменить размер изображения, затем добавить его в соответствующую группу спрайтов.

## Шпаргалка

Методы и атрибуты, которые понадобятся для функции `createScene()`.

```
// Метод. Создать спрайт
element = createSprite(x, y, width, height);

// Метод. Добавить картинку из переменной stoneImg в спрайт
element.addImage(stoneImg);

// Атрибут. Изменить масштаб картинки в 2 раза
element.scale = 2;

// Метод. Получить размер изображения в спрайте
let widthSize = element.width;

// Метод. Добавить спрайт element в группу спрайтов wall
wall.add(element);
```

## Шпаргалка для самых ленивых.

Код функции `createScene()`.

Но даже в этой шпаргалке надо подумать — какими должны быть индексы, чтобы игровое поле соответствовало `рис.1`?

```
function createScene() {
  let x = w / 2;
  let y = w / 2;

  for (let i = 0; i < rows; i++) {
    for (let j = 0; j < cols; j++) {
      let element = createSprite(x, y, w, w);
      if (значение индекса соответствует позиции для камня) {
        element.addImage(stoneImg);
        element.scale = w / element.width;
        wall.add(element);
      } else {
        element.addImage(grassImg);
        element.scale = w / element.width;
        greenField.add(element);
      }
      x += w;
    }
    x = w / 2;
    y += w;
  }
}
```

## >> Создание бомбермена

1. В область глобальных переменных объявить бомбермена и добавить структуру для хранения изображений:

```
let bomberman;  
let bombermanImg = {  
  back: 0,  
  front: 0,  
  left: 0,  
  right: 0,  
};
```

2. Добавить в preload загрузку изображений для анимации:

```
bombermanImg.back = loadAnimation("путь/первыйКадр.png",  
  "путь/последнийКадр.png");
```

3. Добавить в setup создание спрайта для бомбермена и добавить анимацию в спрайт:

```
bomberman = createSprite(x, y, width, height);  
bomberman.addAnimation("left", bombermanImg.left);
```

4. Отмасштабировать бомбермена в setup через атрибут .scale

5. Написать функцию для управления бомберменом с помощью клавиш — bombermanWalkFunction(). В зависимости от направления меняется анимация.

6. Сделать collide бомбермена со группой стена:

```
bomberman.collide(wall);
```

7. Задать в setup() "слои", в которых будут отображаться спрайты. Спрайт с большим значение .depth будет отрисовываться поверх спрайта с меньшим значением:

```
bomberman.depth = 2;  
for (element of greenField) {  
  element.depth = 1;  
}
```

## Как сделать управление и менять анимацию?

1. В зависимости от нажатия клавиши выставить скорость по x или y
2. В зависимости от значения скорости по x и y выбирать анимацию

## Шпаргалка

Методы, атрибуты и функции, которые понадобятся для функции

`bombbermanWalkFunction()`.

```
// Функция keyDown(" ") для считывания клавиши
if (keyDown("w") || keyDown(UP_ARROW)) {
    .. спрайт идёт вверх
}
```

```
// Метод. Установить анимацию с лейблом front
// для спрайта спрайта bombberman
bombberman.changeAnimation("front");
```

```
// Методы для запуска и остановки анимации
bombberman.animation.play();
bombberman.animation.stop();
```

```
// Атрибут для задания скорости анимации
bombberman.animation.frameDelay = 2;
```

```
// Атрибут для задания скорости спрайта
bombberman.velocity.x = 1;
bombberman.velocity.y = 0;
```

## >> Добавляем кирпичи на игровое поле

Взрывающиеся блоки, которые расположены в свободных местах.

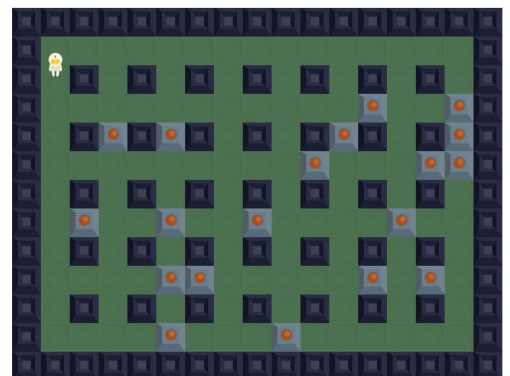
1. В функции `createScene()` мы добавляем спрайты в две группы спрайтов: `greenField` и `wall`. Для создания кирпичей необходимо наполнять ещё группу – `bricks`.
2. Требования к алгоритму:
  - a. Если в определённой точке расположен спрайт из группы `greenField`, то на этом месте можно поставить спрайт кирпич.
  - b. Кирпичи появляются в случайном месте, но занимают не более 20% зелёного поля
  - c. Сделать так, чтобы кирпичи не появлялись на месте создания бомбермена
  - d. В спрайтах группы `greenField` хранить атрибут `coveredByBrick` со значениями `true` или `false`.

```
..
if (индексы соответствуют каменной стене) {
    element.addImage(stoneImg);
    element.scale = w / element.width;
    wall.add(element);
} else {
    element.addImage(grassImg);
    element.scale = w / element.width;

    if (условия создания кирпича) {
        let elementB = createSprite(x, y, w,
w);

        elementB.addImage(brickImg);
        elementB.scale = w /
elementB.width;
        bricks.add(elementB);
        element.coveredByBrick = true;
    } else {
        element.coveredByBrick = false;
    }

    greenField.add(element);
}
..
```



## >> Создание врагов

1. В блоке глобальных переменных объявить `enemyImg`. В прелоаде загрузить анимацию:

```
..
let enemyImg = {
  back: 0,
  front: 0,
  left: 0,
  right: 0,
};
..
function preload() {
  ..
  enemyImg.back = loadAnimation("путь/первыйКадр.png",
    "путь/последнийКадр.png");
  ..
}
```

2. В блоке глобальных переменных объявить `enemies`. В сетапе создать новую группу:

```
..
enemies = new Group();
..
```

3. Необходимо разместить врагов на координатах `greenField` свободных от кирпичей. На этапе создания кирпичей мы сохраняли в `greenField[i].coveredByBrick` – информацию в формате `true / false`.

4. Создайте в блоке `setup()` массив координат, на которых нет кирпичей.

```
..
let positionOnGrassWithoutBricks = [];
for (element of greenField) {
  if (!element.coveredByBrick) {
    positionOnGrassWithoutBricks.push(element.position);
  }
}
..
```

5. Разместите 10 врагов в координатах из массива `positionOnGrassWithoutBricks` в случайном порядке. Добавьте анимацию + лейблы, установить размер спрайта в соответствии с размером блока, установите начальную скорость в случайном направлении.



## Шпаргалка

```
// функция для округления. используйте,  
// чтобы получить индекс в формате int  
let freeRandomPosition =  
floor(random(positionOnGrassWithoutBricks.length));  
  
// добавить анимацию спрайту  
enemy.addAnimation("back", enemyImg.back);  
  
// атрибуты скорости  
enemy.velocity.x = 1  
enemy.velocity.y = 0
```

У врага может быть только одно направление. Т.е. одна из скоростей по `.velocity.x` или `.velocity.y` должна быть равна нулю.

## Шпаргалка для самых ленивых.

```
for (let i = 0; i < 10; i++) {  
    let freeRandomPosition =  
    floor(random(positionOnGrassWithoutBricks.length));  
    let x = positionOnGrassWithoutBricks[freeRandomPosition].x;  
    let y = positionOnGrassWithoutBricks[freeRandomPosition].y;  
  
    let enemy = createSprite(x, y, w, w);  
  
    enemy.addAnimation("back", enemyImg.back);  
    enemy.addAnimation("front", enemyImg.front);  
    enemy.addAnimation("left", enemyImg.left);  
    enemy.addAnimation("right", enemyImg.right);  
  
    enemy.scale = w / 70;  
    enemy.setCollider("rectangle", 0, 0, wall[0].width,  
    wall[0].height);  
  
    if (random(10) >= 5) {  
        enemy.velocity.x = random(10) >= 5 ? -1 : 1;  
    } else {  
        enemy.velocity.y = random(10) >= 5 ? -1 : 1;  
    }  
  
    enemies.add(enemy);  
}
```

## >> Описываем поведение для врагов

1. Создайте функцию `enemyChangeDirection()` вне функций `draw()` и `setup()` – там будет код, которые будет изменять направления спрайта, когда он будет взаимодействовать с другими спрайтами:

```
..
function enemyChangeDirection() {

}
..
```

2. Добавьте в `draw()` взаимодействие врагов со стенами и кирпичами. Добавьте в колбек функцию смены направления для спрайта:

```
..
function draw() {
..
enemies.collide(wall, enemyChangeDirection);
enemies.collide(bricks, enemyChangeDirection);
..
}
..
```

Таким образом, при соприкосновении со стеной будет вызываться функция смены направления.

3. Напишите тело функции для смены направления при соприкосновении со стеной. Для обращения к спрайту используйте `this`. – так же работают все атрибуты и методы спрайта.

```
..
function enemyChangeDirection() {
  if (this.velocity.x === 1) {
    this.velocity.x = 0;
    this.velocity.y = 1;
  }
..
}
```

4. Реализуйте смену анимации в зависимости от направления:

```
if (this.velocity.x > 0) {
  this.changeAnimation("right");
} else if (this.velocity.x < 0) {
  this.changeAnimation("left");
}
```