## 面向对象游戏案例：贪吃蛇

游戏演示

案例目标:

游戏的目的是用来体会js高级语法的使用 不需要具备抽象对象的能力，使用面向对象的方式分析问题，需要一个漫长的过程。

分析：

通过观察页面 发现三个对象 食物 蛇 map （抽象成三个对象 分别写它们各自的属性和方法），最后把三个对象集合在一个大的游戏对象里面 一起来统一调用

**功能实现**

1. 搭建页面
   * 1. 放一个容器盛放游戏场景 div#map，设置样式

#map {

width: 800px;

height: 600px;

background-color: #ccc;

position: relative;

}

* + 1. 分析对象
       1. 食物对象
       2. 蛇对象
       3. 游戏对象
    2. **创建食物对象**

食物food：

属性：

- x

- y

- width

- height

- color

方法： 随机创建一个食物对象，并输出到map上

* + 1. 创建Food的构造函数，并设置属性

var position = 'absolute';

var elements = [];

function Food(x, y, width, height, color) {

this.x = x || 0;

this.y = y || 0;

// 食物的宽度和高度(像素)

this.width = width || 20;

this.height = height || 20;

// 食物的颜色

this.color = color || 'green';

}

* + 1. 实现随机产生食物对象，并渲染到map上

Food.prototype.render = function (map) {

// 随机食物的位置，map.宽度/food.宽度，总共有多少分food的宽度，随机一下。然后再乘以food的宽度

this.x = parseInt(Math.random() \* map.offsetWidth / this.width) \* this.width;

this.y = parseInt(Math.random() \* map.offsetHeight / this.height) \* this.height;

// 动态创建食物对应的div

var div = document.createElement('div');

map.appendChild(div);

div.style.position = position;

div.style.left = this.x + 'px';

div.style.top = this.y + 'px';

div.style.width = this.width + 'px';

div.style.height = this.height + 'px';

div.style.backgroundColor = this.color;

elements.push(div);

}

* + 1. 通过自调用函数，进行封装，通过window暴露Food对象
    2. **创建蛇对象**
       1. 属性：
          1. width
          2. height
          3. body数组，蛇的头部和身体，第一个位置是蛇头
          4. direction 蛇运动的方向 默认right 可以是 left top bottom
       2. 方法：

把蛇渲染到map上

* + 1. Snake构造函数：

var elements = [];

function Snake(width, height, direction) {

// 设置每一个蛇节的宽度

this.width = width || 20;

this.height = height || 20;

// 蛇的每一部分, 第一部分是蛇头

this.body = [

{x: 3, y: 2, color: 'red'},

{x: 2, y: 2, color: 'red'},

{x: 1, y: 2, color: 'red'}

];

this.direction = direction || 'right';

}

* + 1. 把蛇渲染到map上

Snake.prototype.render = function(map) {

for(var i = 0; i < this.body.length; i++) {

var obj = this.body[i];

var div = document.createElement('div');

map.appendChild(div);

div.style.left = obj.x \* this.width + 'px';

div.style.top = obj.y \* this.height + 'px';

div.style.position = position;

div.style.backgroundColor = obj.color;

div.style.width = this.width + 'px';

div.style.height = this.height + 'px';

}

}

* + 1. 暴露Snake对象
    2. **创建游戏对象**

1. 属性：
   * 1. Food
     2. Map
     3. Snake
2. 方法：开始游戏，绘制所有的游戏对象

xii. 构造函数

function Game(map) {

this.food = new Food();

this.snake = new Snake();

this.map = map;

}

xiii. 开始游戏，渲染食物对象和蛇对象

Game.prototype.start = function () {

this.food.render(this.map);

this.snake.render(this.map);

// ========下面最后加====

runSnake(this);

bindKey(this);

}

### 游戏的逻辑

一、 在蛇对象上添加move移动方法

1. 让蛇移动起来，把蛇身体的每一部分往前移动一下

2. 蛇头部分根据不同的方向决定 往哪里移动

Snake.prototype.move = function (food, map) {

var i = this.body.length - 1;

for(; i > 0; i--) {

this.body[i].x = this.body[i - 1].x;

this.body[i].y = this.body[i - 1].y;

}

switch(this.direction) {

case 'left':

this.body[0].x -= 1;

break;

case 'right':

this.body[0].x += 1;

break;

case 'top':

this.body[0].y -= 1;

break;

case 'bottom':

this.body[0].y += 1;

break;

}

}

1. 让蛇自己动起来
   1. 私有方法

什么是私有方法？

不能被外部访问的方法

如何创建私有方法？

使用自调用函数包裹

在game.js中 添加runSnake的私有方法，开启定时器调用蛇的move和render方法，让蛇动起来

判断蛇是否撞墙

function runSnake(that) {

// 注意定时器里面的this有问题，在调用的时候传入this用that代替

var timerId = setInterval(function() {

that.snake.move(that.food, that.map);

// 在渲染前，删除之前的蛇

that.snake.render(that.map);

// 判断蛇是否撞墙

var maxX = that.map.offsetWidth / that.snake.width;

var maxY = that.map.offsetHeight / that.snake.height;

var headX = that.snake.body[0].x;

var headY = that.snake.body[0].y;

if (headX < 0 || headX >= maxX) {

clearInterval(timerId);

alert('Game Over');

}

if (headY < 0 || headY >= maxY) {

clearInterval(timerId);

alert('Game Over');

}

}, 150);

}

b) 每次在渲染蛇之前需要将之前的页面上的蛇删除掉

function remove() {

// 删除渲染的蛇

for(elements.length - 1; i >= 0; i--) {

// 删除页面上渲染的蛇

elements[i].parentNode.removeChild(elements[i]);

// 删除elements数组中的元素

}

elements=[];

}

c) 在game中通过键盘控制蛇的移动方向,并在start方法中调用

function bindKey() {

// 注意：这个函数里面的this有问题

document.addEventListener('keydown', function(e) {

switch (e.keyCode) {

case 37:

// left

that.snake.direction = 'left';

break;

case 38:

// top

that.snake.direction = 'top';

break;

case 39:

// right

that.snake.direction = 'right';

break;

case 40:

// bottom

that.snake.direction = 'bottom';

break;

}

}, false);

}

d) 在Snake的move方法中 判断蛇是否吃到食物

// 在移动的过程中判断蛇是否吃到食物

// 如果蛇头和食物的位置重合代表吃到食物

// 食物的坐标是像素，蛇的坐标是几个宽度，进行转换

var headX = this.body[0].x \* this.width;

var headY = this.body[0].y \* this.height;

if (headX === food.x && headY === food.y) {

// 吃到食物，往蛇节的最后加一节

var last = this.body[this.body.length - 1];

this.body.push({

x: last.x,

y: last.y,

color: last.color

})

// 把现在的食物对象删除，并重新随机渲染一个食物对象

food.render(map);

}

### 其它处理

自调用函数的参数

- 传入window对象

将来代码压缩的时候，可以把 function (window) 压缩成 function (w)

- 传入undefined

在将来会看到别人写的代码中会把undefined作为函数的参数(当前案例没有使用)因为在有的老版本的浏览器中 undefined可以被重新赋值，防止undefined 被重新赋值

整理代码

现在的代码结构清晰，谁出问题就找到对应的js文件即可。

通过自调用函数，已经防止了变量命名污染的问题

但是，由于js文件数较多，需要在页面上引用，会产生文件依赖的问题(先引入那个js，再引入哪个js)

将来通过工具把js文件合并并压缩。现在手工合并js文件演示

问题1

// 如果存在多个自调用函数要用分号分割，否则语法错误

// 下面代码会报错

(function () {

}())

(function () {

}())

// 所以代码规范中会建议在自调用函数之前加上分号

// 下面代码没有问题

;(function () {

}())

;(function () {

}())

问题2

// 当自调用函数 前面有函数声明时，会把自调用函数作为参数

// 所以建议自调用函数前，加上;

var a = function () {

alert('11');

}

(function () {

alert('22');

}())