软件工程
《子母鱼游戏》系统设计说明书

班　级：2015软件工程瑞典班

姓　名：王成、姚少峰、赵唯均

指导教师：王昭顺

主要的人员分工

一：报告编写的分工如下：
需求分析：赵唯均
可行性分析：姚少峰，赵唯均
概要设计：王成，姚少峰
详细设计：王成，姚少峰
测试：赵唯均

二：代码编写的分工如下：
思路整合：赵唯均
模块设计：王成，姚少峰
主要代码处理与实现：王成

目 录

引 言

如今，游戏机风行的程度，是第一台电子游戏机的研制者诺兰布什纳尔先生始料不及的。在全世界最大的城市，直至最小的村庄，从纽约最辉煌的游乐场，到高加索最小的乡镇儿童娱乐点，在千家万户，正在进行着千千万万这样的“战斗”，伴随着无数成功与失败，兴奋与懊丧。游戏机带来了一个全球性的疯狂症，其他任何娱乐与之相比都望尘莫及。然而，究竟是什么原因使游戏机如此风行呢? 在回顾了游戏机发展简史之后，我们不难悟出，技术进步在游戏机发展过程中起到了极大的促进作用。

但是，技术进步绝不是游戏机风行的唯一因素。随着终端设备开发能力的加强，作为娱乐终端的游戏机也得到了很大程度的发展。这也加速了游戏机在全球风行程度，所以对于游戏机的研究和设计具有很重要的意义，这也是本课题研究的来源。

以HTML5为基础，通过编程实现游戏的所有功能，模拟出单机版游戏大鱼吃饵食，随后不断喂给小鱼的情景。

HTML5已经出来很多年了，HTML5是一个基于浏览器的协作标准，可以让各种不同的素材在浏览器中流畅运行，它最大的优点在于跨平台性、易开发以及开发成本低。早在2010年的时候，乔布斯在封杀Flash的言论中，就预言HTML5将会成为取代Flash的下一波技术浪潮。特别是之前微信推出的一系列的基于HTML5的小游戏，如“围住神经猫”等的火爆，更是让我们看到了HTML5网页游戏方面的潜在市场。

1 系统开发的可行性分析

可行性分析以调查研究的结果为基础，经过可行性调研，进一步论证系统的必要性和可能性。子母鱼游戏的开发主要是从经济，技术，社会三个方面进行可行性研究。

1.1 经济可行性

子母鱼游戏是一种模拟大鱼吃小鱼的游戏，它的采集，规划，设计等都是在计算机平台环境下实施的，其表达载体是屏幕。在本课题中我们选取的仅仅是大鱼和小鱼的两个图片，再加上海藻，果实，漂浮物等不多的图片，所以从经济角度来看制作一个游戏是完全低成本高效率的项目，经济可行。

1.2 技术可行性

本系统可跨平台操作，开发采用HTML和JavaScript语言，运用Sublime作为系统开发软件，结合JavaScript语言通过网页展示给大家。

HTML语言具有安全、可移植性等特点，用HTML语言编译的软件不局限于某一个平台下，它可以跨平台运行，即一次编译，处处运行。

另外，开发环境对计算机的要求不是很高，开发成本低，软件对服务器配置要求也不是很高，为我们的研究提供了便利。

1.3 社会可行性

目前国内游戏公司已达到了200多家，市场上运营的游戏亦达250多款，但与欧美、韩国等国家的游戏发展程度还有一定差距。我国有广阔的游戏市场，宽阔的发展空间，无限的继续挑战和剧增的玩家队伍。随着网络的兴起，巨大的市场需求量使中国在短短几年，已经从无到有，从陌生到熟知，从掌握到运用。游戏必将成为网络竞争中的主角，他的商业利益和商业价值无法估量。中国，人口众多的国家，随着人们生活水平的提高，温饱已经不是最终目的，我们寻求的是更加多姿多彩的生活，那么我们怎么能错过游戏这一最佳休闲娱乐项目呢？所以我国游戏发展前景一片大好！

2 系统开发的需求分析

需求分析就是发现、求精、建摸、规格说明和复审的过程。为了发现用户的真正需求，首先应该从宏观角度调查、分析用户所面临的问题。也就是说，需求分析的第一步是尽可能了解当前情况和需要解决的问题。

2.1 游戏目的

子母鱼游戏是一个小型休闲游戏，可以锻炼人们的反应能力，尤其是在开发人们智力，锻炼反应能力方面，很受人们欢迎。本组的这个小游戏程序包括Html5，JavaScript，CSS3，Canvas等方面的知识。

编写本游戏设计与实现的需求分析报告，是为了以书面形式把用户对《基于HTML5的子母鱼游戏》使用的要求全面地描述出来，以作为下一步游戏开发设计的依据。该游戏的设计主要是减轻各类人群的生活、学习、工作压力。使人心情舒畅，提升工作效率等。

2.2 设计目标及功能

游戏主要由鼠标进行操作，鼠标操作大鱼吃果实，果实分两种，橙色和蓝色。橙色果实吃一个记一百分，蓝色是功能果实，吃一个蓝色果实后，再吃橙色果实会使分值加倍。大鱼初始颜色为白色，吃了橙色果实后，身体颜色会变红，吃了蓝色果实后，身体颜色会变蓝。小鱼初始颜色是橙红色，会随着时间的增长使自身颜色不断减弱，当减弱至白色时，游戏结束。但是大鱼在吃完果实后，可以对小鱼进行喂食，喂食后，小鱼减弱的颜色恢复至最初的橙红色，而大鱼不管是蓝色还是红色，统统恢复至初始的白色，未吃果实的大鱼无法对小于进行喂食。

3 系统开发的概要设计

3.1 功能设计

本次课题设计中该游戏具备的的基础功能有：

- 大鱼随鼠标移动：这里主要是定义了一个lerpDistance函数，这个函数有三个参数：目标（aim）、当前值（cur）和比率（ratio），然后返回

一个按照一定比率趋向于目标的值，所以运行起来会一直追随目标值，这样就可以使大鱼的坐标追随鼠标，而小鱼追随大鱼。

- 大鱼吃果实：这里主要是运用了一个碰撞检测的功能，当果实处于活着的状态时，使用坐标差就算大鱼和果实的距离，然后当这个距离小于一定值时，就说明果实被吃掉。
- 喂食：同大鱼吃果实类似，这里的喂食其实是一个大鱼同小鱼的碰撞检测模块。
- 颜色变换：这里是将各种颜色的图片置入一个数组，然后根据的时间的增加，每一帧绘制一张图。
- 果实出生：在图片底部有无数条海葵，果实是在海葵的头部出生，每一个海葵都有一个编号，然后让果实的图片随着时间的增加而在海葵上长大，当长到一定的大小时，让图片的y坐标不断的减小，这就造成果实上浮的动画。

另外还有通过绘制贝塞尔曲线定义的海葵，海面漂浮物动画，大鱼小鱼的摆动动画，画面特效以及计分系统等功能。

3.2代码设计

HTML部分的代码特别简单，基本是采用的HTML5规范的标记，然后只定义了一个div类，然后其中包含了两个canvas，用z-index将这两个canvas进行重叠，在上层canvas上绘制鱼，灰尘，特效和UI等内容，在下层canvas上绘制背景，海葵，果实等，Html部分主要代码如下：

```
<body>
	<div class="all_bg">
		<div id="allcanvas">
			<canvas id="canvas1" width="800" height="600"></
canvas>
			<canvas id="canvas2" width="800" height="600"></
canvas>
		</div>
	</div>
	<script src="js/main.js"></script>
	<script src="js/commonFunctions.js"></script>
	<script src="js/background.js"></script>
	<script src="js/ane.js"></script>
	<script src="js/fruit.js"></script>
	<script src="js/mom.js"></script>
	<script src="js/collision.js"></script>
	<script src="js/baby.js"></script>
	<script src="js/data.js"></script>
	<script src="js/wave.js"></script>
	<script src="js/halo.js"></script>
	<script src="js/dust.js"></script>

</body>
```

3.3页面模块设计

该游戏的页面部分设计的及其简便，仅使用了两个Canvas进行控制，然后用CSS进行一定的修饰，CSS部分的代码如下：

```
<style>
    body{
        padding-top: 10px;
    }
    .all_bg{
        width:          800px;
        height:         600px;
        margin:         0px auto;
    }
    #allcanvas{
        position:       relative;
        width:          800px;
        height:         600px;
        margin:         0px;
    }
    #canvas1{
        position:       absolute;
        bottom:         0;
        left:           0;
        border:         solid 1px black;
        z-index:        1;

    }
    #canvas2{
        position:       absolute;
        bottom:         0;
        left:           0;
        border:         solid 1px black;
        z-index:        0;
    }

</style>
```

4 系统开发的详细设计

4.1 系统各部分的实现方法

从系统功能模块划分中可以看出，系统总体上有两个功能模块组成。一个是静态显示，一个是动态控制。

4.1.1 游戏背景及海葵显示功能

这部分我们首先是绘制出整个游戏的背景，这个用一个drawImage的API即可完成，然后绘制海葵，根据画面大小我们定义其为50个，海葵长在海底部，所以所有海葵的y坐标是相同的，但是x坐标不同，这里我们使用一个

Math.random()函数进行随机的控制，然后给海葵一个高度，同样用随机函数控制，避免所有的海葵是同一个高度。

然后海葵其实是可以随着水流进行摆动，这里的摆动功能我们使用绘制贝塞尔曲线的方式进行控制，起始点和控制点不动，结束点用一个正弦函数(sin)控制，来回摆动，然后给每一个海葵一个振幅，避免所有海葵摆动幅度相同显得不真实。海葵部分的代码如下：

```
var aneObj = function(){
        this.rootx = [];//根部
        this.headx = [];//头部
        this.heady = [];
        this.amp = [];//振幅
        this.alpha = 0; //定义一个角度控制摆动

};
aneObj.prototype.num = 50;
aneObj.prototype.init = function(){

        for(var i = 0 ; i < this.num ; i++){
                this.rootx[i] = i * 16 + Math.random() * 20;//[0,1)
                this.headx[i] = this.rootx[i];//结束点的x坐标
                this.heady[i] = canHeight - 250 + Math.random() * 50;//结束点的y坐标，随机摆动
                this.amp[i] = Math.random() * 50 + 50;
        }
};

aneObj.prototype.draw = function(){

        this.alpha += deltaTime * 0.0008;
        var l = Math.sin(this.alpha);//[-1,1]
        ctx2.save();
        ctx2.globalAlpha = 0.6;//透明度是0.6
        ctx2.lineWidth = 20;
        ctx2.lineCap = "round";
        ctx2.strokeStyle = "#3b154e";
        for(var i = 0 ; i < this.num ; i++){
                ctx2.beginPath();
                ctx2.moveTo(this.rootx[i],canHeight);

                this.headx[i] = this.rootx[i] + l * this.amp[i];//当前海葵的具体位置
                ctx2.quadraticCurveTo(this.rootx[i],canHeight - 100,this.headx[i],this.heady[i]);
                ctx2.stroke();
        }
        ctx2.restore();
};
```

4.1.2 大小鱼的绘制及动画形式

大小鱼的绘制主要有几下几点内容：首先是将大小鱼的身体图片进行整合，然后显示在界面上，每条鱼都是由身体，眼睛和尾巴三部分组成；然后是使大鱼随着鼠标移动，小鱼跟着大鱼移动；最后是大小鱼的尾巴会匀速的摆动，眼睛随机进行眨动，身体的颜色随着吃到果实和时间的变化而变化。

首先第一部分，是将大小鱼的眼睛，身体和尾巴分别放入数组中，然后通过drawImage绘制在Canvas上，这一步非常简单。

然后第二部是定义一个鼠标的监听函数：

```
function onMouseMove(e){
        if(!data.gameOver){
                if(e.offSetX || e.layerX){
                        mx = e.offSetX == undefined ? e.layerX :
e.offSetX;
                        my = e.offSetY == undefined ? e.layerY : e.offSetY;
                }
        }

}
```

这样就获取到鼠标的实时坐标（mx，my），然后通过lerpDistance函数使大鱼的坐标不断的趋向于鼠标的坐标，再使小鱼的坐标不断趋向于大鱼的坐标，当然这其中还有个角度的问题，如大鱼和鼠标之间各有一个角度，这里可以利用Math.atan2反正切函数计算出这个角度差，由于atan2的返回值是在-PI到PI之间，所以这个角度值还要再加一个PI，然后让大鱼的角度通过lerpAngle函数不断趋向于计算出的这个角度，就可以使大鱼的头部始终对着鼠标而不至于特别生硬。其中的lerpDistance和lerpAngle函数定义如下：

```
function lerpAngle(a, b, t) {
        var d = b - a;
        if (d > Math.PI) d = d - 2 * Math.PI;
        if (d < -Math.PI) d = d + 2 * Math.PI;
        return a + d * t;
}

//lerp 使当前值cur趋向于目标值aim，可以让大鱼，趋向于鼠标的位置
function lerpDistance(aim, cur, ratio) {
        var delta = cur - aim;
        return aim + delta * ratio;//返回的是按照一定比率趋向于目标的值，所以
运行起来会一直追随目标值
}
```

然后，大鱼部分的代码如下所示，小鱼部分类似于大鱼，仅仅是跟随的目标有所区别：

```
var momObj = function(){
        this.x;
```

```
        this.y;
        this.angle;

        this.bigTailTimer = 0;
        this.bigTailCount = 0;

        this.bigEyeTimer = 0;
        this.bigEyeCount = 0;
        this.bigEyeInterval = 1000;

        this.bigBodyCount = 0;//计数器，当吃到果实的时候发生变化
}
momObj.prototype.init = function(){
        this.x = canWidth * 0.5;
        this.y = canHeight * 0.5;
        this.angle = 0;
}
momObj.prototype.draw = function(){

        //lerp x,y
        this.x = lerpDistance(mx,this.x,0.95);//目标是鼠标的x,y
        this.y = lerpDistance(my,this.y,0.95);
        //delta angle每一帧都要计算角度差
        var deltaY = my - this.y;
        var deltaX = mx - this.x;
        var beta = Math.atan2(deltaY,deltaX) + Math.PI;
        //lerp angle让大鱼坐标
        this.angle = lerpAngle(beta,this.angle,0.6);

        this.bigTailTimer += deltaTime;
        if(this.bigTailTimer > 50){
                this.bigTailCount = (this.bigTailCount + 1) % 8;
                this.bigTailTimer %= 50;
        }
        //bigEye
        this.bigEyeTimer += deltaTime;

        if(this.bigEyeTimer > this.bigEyeInterval){
                this.bigEyeCount = (this.bigEyeCount + 1) % 2;
                this.bigEyeTimer %= this.bigEyeInterval;

                if(this.bigEyeCount == 0){
                        this.bigEyeInterval = Math.random() * 1500 + 2000;
                }else{
                        this.bigEyeInterval = 200;
                }
        }
        ctx1.save();
        ctx1.translate(this.x,this.y);
```

```
        ctx1.rotate(this.angle);//大鱼旋转

        var bigTailCount = this.bigTailCount;
        ctx1.drawImage(bigTail[bigTailCount],-bigTail[bigTailCount].width * 0.5
+ 30,-bigTail[bigTailCount].height * 0.5);

        var bigBodyCount = this.bigBodyCount;
        if(data.double == 2){//判断是否是蓝色
ctx1.drawImage(bigBodyBlue[bigBodyCount],-
bigBodyBlue[bigBodyCount].width * 0.5,-bigBodyBlue[bigBodyCount].height *
0.5);
        }else{
                ctx1.drawImage(bigBodyOrange[bigBodyCount],-
bigBodyOrange[bigBodyCount].width * 0.5,-
bigBodyOrange[bigBodyCount].height * 0.5);
        }

        var bigEyeCount = this.bigEyeCount;
        ctx1.drawImage(bigEye[bigEyeCount],-bigEye[bigEyeCount].width *
0.5,-bigEye[bigEyeCount].height * 0.5);

        ctx1.restore();
}
```

4.1.3 果实的绘制及动画

果实部分绘制也有三个部分，但个模块与其他所有模块都是紧密相连的。首先果实的出生，果实出生是在海葵上，由一个点开始长大，我们设定的是屏幕上果实数量为15个，一旦有一个死亡，海葵上会立即有果实出生；然后是果实的漂浮，这点比较好理解，当果实的大小达到一定程度，果实的y坐标就会随着时间的增加而减小，这里我们采用一个随机函数来控制果实漂浮的速度，避免所有果实同步上升不太美观；其次就是果实的状态选择，果实分两种，蓝色和橙色，这也是由一个随机函数控制，当random()值小于0.3时我们让他产生蓝色果实，否则就是橙色；最后就是分值的统计，蓝色果实是加倍果实，本身不记分数，橙色果实一个100分，当吃到蓝色果实时候，橙色果实分值 * 2，这部分具体的代码如下所示：

```
var fruitObj = function() {
        this.alive = [];      //bool
        this.x = [];
        this.y = [];
        this.aneNO = [];
        this.l = [];              //果实的半径(大小)
        this.spd = [];          //每个果实自己的速度
        this.fruittype = [];    //果实类型
        this.orange = new Image();
        this.blue = new Image();
};
fruitObj.prototype.num = 30;
```

```
fruitObj.prototype.init = function() {

	for(var i = 0; i < this.num; i++) {
		this.alive[i] = false;
		this.x[i] = 0;
		this.y[i] = 0;
		this.aneNO[i] = 0;  //哪一个海葵
		this.spd[i] = Math.random() * 0.02 + 0.005;
		this.fruittype[i] = "";
	}
	this.orange.src = './src/fruit.png';
	this.blue.src = './src/blue.png';
};
fruitObj.prototype.draw = function() {
	for(var i = 0; i < this.num; i++) {
		var pic;
		if( this.alive[i] ) {
			if(this.fruittype[i] == "blue") {
				pic = this.blue;
			}else {
				pic = this.orange;
			}
			if(this.l[i] <= 15) {
				var NO = this.aneNO[i];
				//随时间变化果实长大，deltaTime使过程平缓些
				this.x[i] = ane.headx[NO];
				this.y[i] = ane.heady[NO];
				this.l[i] += this.spd[i] * deltaTime;
				ctx2.drawImage(pic, this.x[i]-this.l[i] / 2, this.y[i]-
this.l[i] / 2, this.l[i], this.l[i]);

			}else {
				//果实上升的速度变化
				this.y[i] -= this.spd[i] * 8 * deltaTime;
				ctx2.drawImage(pic, this.x[i]-this.l[i] / 2, this.y[i]-
this.l[i] / 2, this.l[i], this.l[i]);
			}

			if( this.y[i] < 10) {
				this.alive[i] = false;
			}
		}
	}
};
fruitObj.prototype.born = function(i) {
	this.aneNO[i] = Math.floor(Math.random() * ane.num);
	this.l[i] = 0;
	this.alive[i] = true;
	var fruRand = Math.random();   //随机果实类型
```

```
        if(fruRand < 0.3){
                this.fruittype[i] = "blue";
        }else {
                this.fruittype[i] = "orange";
        }

};
fruitObj.prototype.dead = function(i) {
        this.alive[i] = false;
};
function fruitMonitor() {
        var num = 0;
        for(var i = 0; i < fruit.num; i++) {
                if (fruit.alive[i]) {
                        //记录存活的果实数量
                        num++;
                }
        }
        if ( num < 15) {
                sendFruit();
                return;
        }
}
function sendFruit() {
        for (var i = 0; i < fruit.num; i++) {
                if( !fruit.alive[i] ) {
                        fruit.born(i);
                        return;
                }
        }
}
```

4.1.4 动画特效

特效这里我们引入一个“物体池”的概念，比如一个池子，里面有30个特效圈圈，每个圈圈都有一个状态，闲置或者正被使用，当鱼吃果实的时候，就去池子里找一个圈圈出来，这时要进行检查，若找的这个圈圈的状态是闲置的，即可拿出去进行绘制。

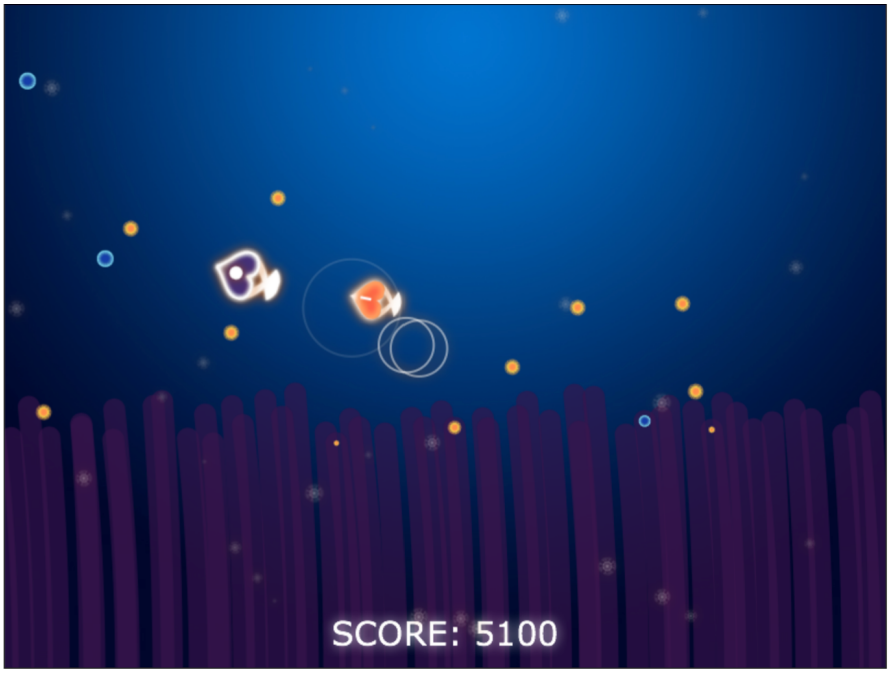
在绘制的过程中，圈圈的半径会随着时间的增加而变大，而颜色会随着半径的增加而减弱，而圈圈的圆心位置，既是碰撞检测中获取的果实与大鱼碰撞的坐标和大鱼喂食小鱼的坐标，具体代码如下所示：

```
var waveObj = function(){
        this.x = [];
        this.y = [];
        this.alive = [];//是否闲置,false是闲着，true是忙着
        this.r = [];
}
```

```
waveObj.prototype.num = 10;

waveObj.prototype.init = function(){

	for(var i = 0 ; i < this.num ; i++){
		this.alive[i] = false;
		this.r[i] = 0;
	}
}
waveObj.prototype.draw = function(){

	ctx1.save();
	ctx1.lineWidth = 2;
	ctx1.shadowBlue = 10;

	for(var i = 0; i < this.num ; i++){
		if(this.alive[i]){
			//draw
			this.r[i] += deltaTime * 0.02;
			if(this.r[i] > 50){
				this.alive[i] = false;
				break;
			}
			var alpha = 1 - this.r[i] / 50; //颜色与半径成反比
			ctx1.beginPath();
			ctx1.arc(this.x[i],this.y[i],this.r[i],0,2 * Math.PI);
			//ctx1.closePath();
			ctx1.strokeStyle = "rgba(255,255,255," + alpha + ")";
			ctx1.stroke();
		}
	}
	ctx1.restore();
}
//判断是否闲着
waveObj.prototype.born = function(x,y){
	for(var i = 0; i < this.num ; i++){
		if(!this.alive[i]){
			this.alive[i] = true;
			this.r[i] = 10;
			this.x[i] = x;//出生时就获取到圆心的位置
			this.y[i] = y;

			return;
		}
	}
}
```

4.1.5 界面设计

整体的界面大概如下所示：



5 系统开发测试

5.1什么是软件测试

软件测试是在软件投入运行前,对软件需求分析、设计规格说明书和编码的最终的复审，是软件质量保证的关键步骤。确切的说，软件测试就是为了发现错误而执行的过程。一般分为两个阶段：

1.单元测试：在编完一模块后进行测试；

2.综合测试：在开发完软件后进行综合测试。

5.2软件测试的目标与方法

G.Myers给出了关于测试的一些规则，这些规则可以看作是测试的目标或定义：

1.测试是为了发现程序中的错误而执行程序的过程；

2.好的测试方案是极可能发现迄今为止尚未发现的错误的测试；

3.成功的测试是发现了至今为止尚未发现的错误的测试。

测试任何软件都有两种方法：黑盒测试和白盒测试法。

黑盒测试法（又称功能测试法）是把程序看成一个黑盒子，完全不考虑程序的内部结构和处理过程，是在程序接口进行的测试，它只检查程序功能是否按照规格说明书的规定正常使用。

白盒测试法（又称结构测试法）是把程序看成装在一个透明的白盒子里，也就是完全了解程序的结构和处理过程，这种方法按照程序内部的逻辑测试程序，检验程序中的每条通路是否都能按照预定要求正确工作。

5.3系统的不足和展望

本次做的子母鱼休闲小游戏有很多的不足和不完善的地方，在功能的实现上有些操作由于我们水平的限制不够便捷灵敏，游戏范围局限在同一个背景之中，游戏模式较为单一，除此以外在界面上的设计上也较为简陋，美观度有待提高。

之后可对游戏增加一些升级闯关的功能，随着级别的增长，果实的速度越来越快，或者可以达到一定分数之后，鱼的状态发生进化，由最开始的小鱼，变成大鲤鱼，然后梭鱼等等，直到最后变成鲨鱼，这样的话，可玩性会增加很多，但由于时间的关系，这些内容将随后慢慢进行完善。