**命名规范：**

1. HTML、CSS中 id、class命名：

语义命名，单词使用小写，间隔使用”\_”连接，如页首div，id=”page\_head”;

1. JS中变量、函数命名：

语义命名，开头单词用小写，间隔单词首字母使用大写，如时间time、游戏控制器gameContorler、表单检测函数listCheck();

JS中jQuery对象命名规则与上述规则相同，只需在名称前加$符号即可，如页首div jQuery对象$pageHead。

**详细设计：**

1. 游戏变量（全局变量）：

SCREEN\_WIDTH

描述：游戏界面宽；

对象类型：number；

常量：

SCREEN\_HEIGHT

描述：游戏界面高；

对象类型：number；

常量：

SYS\_processor

描述： 进程类的实例，管理游戏中所有的游戏对象；

对象类型：processor；

常量：

SYS\_collisionManager

描述： 碰撞管理类的实例；

对象类型：collisionManager；

常量：

SYS\_timeInfo

描述： 帧数信息的实例；

对象类型：timeInfo；

常量：

SYS\_spriteParams

描述： canvasSprite对象的常用参数；

对象类型：Object；

常量：

1. playerControl 玩家操作类（即自行函数）：

说明：识别玩家的操作，并将其保存在游戏变量中。

**属性：**

**controlInfo**

描述： 玩家的操作信息；

对象类型：Object；

缺省值：{

singleClick：[0，{x，y}]， //1表示单击操作，xy表示单击点

doubleClick：[0，{x，y}]， //1表示双击操作，xy表示双击点

drag：[0，{x，y}，{x，y}] //1表示拖动操作，xy分别表示起点和终点

}；

1. vector2d js向量：

说明：新定义的js类型，相当于数学中的向量，即有方向又有长度，在设计中充当简化数据结构的角色，方便算法的设计。

传入参数：

x，y //以原点为起点的向量终点坐标

返回：

vec //属性，即生成的向量

**属性：**

**vec**

描述： 生成的向量

对象类型：Object；

缺省值：

**属性：**

**vx**

描述： 向量终点的x坐标

对象类型：number；

缺省值：x

**vy**

描述：向量终点的y坐标

对象类型：number；

缺省值：y

**方法：**

**negate**

描述：反向量；

参数：无；

返回： vector；

**length**

描述：求向量长度；

参数：无；

返回：number；

**normalize**

描述：该向量的单位向量；

参数：无；

返回：vector；

**dot**

描述：该向量的点积；

参数：无；

返回：number；

1. game 顶层类（即自行函数）：

说明：将所有游戏对象和游戏逻辑都捆绑在一起，响应游戏对象的信息，并控制游戏开始，结束。

**属性：**

**time**

描述： 游戏时间

对象类型：number；

缺省值：

**gameState**

描述： 游戏状态；

对象类型：string；

缺省值：“freeze”；

**score**

描述： 游戏分数；

对象类型：number；

缺省值：0；

**insects**

描述： 昆虫管理实例；

对象类型：insectManager；

缺省值：

**nets**

描述： 蜘蛛网管理实例；

对象类型：netManager；

缺省值：

**trees**

描述： 树枝管理实例；

对象类型：tree；

缺省值：

**gameOverFlag**

描述： 游戏结束标志；

对象类型：bool；

缺省值：ture；

**mySpider**

描述： 游戏的蜘蛛对象；

对象类型：spider；

缺省值：

**方法：**

**init**

描述：清楚画布，之后初始化各个管理器以及蜘蛛；

参数：无；

返回：无；

**gameOver**

描述：游戏结束函数，调用在之后设置游戏结束标签，清除画布并且设置游戏状态；

参数：无；

返回：无；

**gameCallBack**

描述：响应游戏过程中返回的信息，并作出处理；

参数：Object；

返回：无；

**gameLoop**

描述：游戏主循环体，没每15ms执行一次，通过gameState和gameOverFlag进行判断游戏状态，获取timeInfo，并且执行碰撞检测和主进程，或者执行游戏结束函数；

参数：无；

返回：无；

1. processor 进程类：

说明：对游戏中所有对象进行管理，包括新增对象，删除对象，对象位置改变。

**属性：**

**processList**

描述： 存放并处理所有游戏对象的列表；

对象类型：Array；

缺省值：

**addedItem**

描述： 上一次调用后新增加的游戏对象列表；

对象类型：Array；

缺省值：

**方法：**

**add**

描述：向addedItem中添加新增加的游戏对象，在下一次调用时添加至processList中，添加的游戏对象应为canvasSprite类型；

参数：canvasSprite；

返回：无；

**process**

描述：控制所有游戏对象进行移动，并通过游戏对象的remove标记控制游戏对象的删除，通过addedItem控制游戏对象的增加；

参数：无；

返回：无；

1. conllisionManager 碰撞检测：

说明：碰撞对象类，管理所有需要检测碰撞的对象和被碰撞的对象。

**属性：**

**listIndex**

描述： 碰撞检测中宽阶检测用到的网格在grid数组中的标记；

对象类型：number；

缺省值：0；

**grid**

描述： 网格的存放数组，网格为对象类型，网格中存放的是需要被检测碰撞游戏对象；

对象类型：Array；

缺省值：

**checkListIndex**

描述： 需要进行碰撞检测的游戏对象在checkList中的标签；

对象类型：number；

缺省值：0

**checklist**

描述：存放需要碰撞检测的列表；

对象类型：Object；

缺省值：

**gridWidth**

描述：碰撞网格的宽；

对象类型：number；

缺省值：

**gridHeight**

描述： 碰撞网格的高

对象类型：number；

缺省值：

**方法：**

**getGridList**

描述：通过接受画布上的x、y坐标来返回网格；

参数：number，number（x，y）；

返回：Object（grid）；

**newCollider**

描述：由需要和 其他对象发生碰撞的游戏对象调用，生成一个碰撞对象；

参数：number（colliderFlag主动碰撞标签），number（collideeFlags被动碰撞标签），Array（position的{x，y}），Function（callback回调函数）；

返回：Object（colliderObj）;

**checkCollisions**

描述：调用checkList中碰撞对象自己的checkCollisions函数来检测碰撞；

参数：无；

返回：无；

**colliderObj**

**属性：**

**list**

描述： 碰撞对象所在的网格；

对象类型：Object；

缺省值：

**indexStr**

描述： 碰撞对象在网格中的序号；

对象类型：number；

缺省值：‘‘+listIndex++；

**checkIndex**

描述： 碰撞对象在碰撞检测列表里的序号；

对象类型：number；

缺省值：

**colliderFlag**

描述： 碰撞对象的碰撞标签；

对象类型：number；

缺省值：colliderFlag；

**collideeFlag**

描述： 被碰撞对象的碰撞标签；

对象类型：number；

缺省值：collideeFlag；

**方法：**

**update**

描述：碰撞对象改变位置后，调用该函数进行更新碰撞对象在网格中的位置，以及碰撞对象刚生成时在网格中的位置（需要判断该对象是蜘蛛网或者其他）；

参数：无；

返回：无；

**remove**

描述：将碰撞对象从网格中移除，如果有collideeFlag标签为1，则还需从碰撞列表中移除；

参数：无；

返回：无；

**callback**

描述：碰撞对象发生碰撞时调用的函数；

参数：无；

返回：无；

**checkCollisions**

描述：碰撞对象扫描与偏离中心offsetX和offsetY距离网格内的碰撞对象是否发生碰撞；

参数：number（offsetX），number（offsetY）；

返回：bool；

1. canvasSprite canvas对象类：

说明：所有画布上的对象的基础类，包括画布对象的位移、渲染和动画。

参数：Object（SYS\_spriteParams）；

**属性：**

**listIndex**

描述：

对象类型：number；

缺省值：

1. timeInfo 速度控制类

说明：控制游戏帧数，同时能够提供游戏时间的相关信息。

参数：number（goalFPS目标帧数）

**属性：**

**oldTime**

描述：上一次调用时的时间；

对象类型：number；

缺省值：

**paused**

描述： 暂停标签

对象类型：bool；

缺省值：true；

**interCount**

描述： 内部计数对象；

对象类型：number；

缺省值：0；

**totalFPS**

描述： 总计的FPS数，用来计算平均FPS；

对象类型：number；

缺省值：0；

**totalCoeff**

描述： 总计的时间延迟系数，用来计算平均时间延迟系数；

对象类型：number；

缺省值：0；

**方法：**

**getInfo**

描述：计算上次调用之后所用的时间、FPS、时间延迟等，得到的信息来保证游戏FPS看起来一致；

参数：无；

返回：Object（{

elapsed：0； //上次调用经历的时间；

coeff：0； //时间延迟系数；

FPS：0； //当前帧数；

averageFPS：0； //游戏平均帧数；

averageCoeff：0 //游戏平均时间延迟系数；

}）；

**pause**

描述：暂停计数，即设置暂停标签，游戏中所有暂停都应调用该方法；

参数：无；

返回：无；

1. animEffect 动画类：

说明：动画效果的实现类，能放置动画，循环播放动画，并且能移除动画。

参数：x，y，imageList，timeout（动画产生的位置，动画图片，持续时间）；

（以上均为基础类，以下类均继承于以上的类）

1. insect 昆虫类：

说明：昆虫的基本信息，包括图片，位置，分数，血量等信息。

参数：x，y，kind{point，health}，frame，gameCallBack（位置，分数，血量，昆虫的大小，游戏信息返回函数）；

**属性：**

**x**

描述： 昆虫的位置；

对象类型：number；

缺省值：x

**y**

描述： 昆虫的位置；

对象类型：number；

缺省值：y

**z**

描述： 昆虫的垂直位置；

对象类型：number；

缺省值：1000

**radius**

描述： 昆虫的半径；

对象类型：number；

缺省值：frame

**that**

描述： 昆虫的canvasSprite对象实例；

对象类型：canvasSprite；

缺省值：

**that.catchFlag**

描述： 昆虫是否被捕捉到的标签；

对象类型：number；

缺省值：0；

**collider**

描述： 昆虫的碰撞对象；

对象类型：colliderObj；

缺省值：

**moveVector**

描述： 昆虫的移动向量；

对象类型：vector2d；

缺省值：

**方法：**

**that.getXY**

描述：返回昆虫的xy；

参数：无；

返回：{x,y}；

**that.getZ**

描述：返回昆虫的z值；

参数：无；

返回：z；

**that.remove**

描述：将昆虫从SYS\_processor，和碰撞列表中移除，并且播放动画；

参数：无；

返回：无；

**that.move**

描述：根据昆虫的移动向量来改变昆虫的位置，之后改变昆虫的移动向量，并减小z，使其做圆周运动，改变昆虫的图片，在昆虫的位置上画出昆虫；

参数：无；

返回：无；

**that.insertCollider**

描述：将collider根据参数初始化，并且添加到碰撞检测列表中；

参数：number（colliderFlag主动碰撞标签），number（collideeFlags被动碰撞标签）；

返回：无；

**that.collision**

描述：昆虫检测碰撞的函数，检测与蜘蛛网的碰撞；

参数：Object（碰撞对象的位置信息）；

返回：bool；

**that.hit**

描述：昆虫被碰撞之后调用的函数，需要根据碰撞对象的类型进行判断，如果是蜘蛛，则向游戏返回昆虫被吃的信息，并且调用that.remove()将昆虫移除，如果是蜘蛛网，则将catchFlag置为1，并将昆虫从碰撞列表中删除。

1. insectsManager 昆虫管理类

说明：统一管理昆虫的类，生成昆虫等；

参数：无；

**属性：**

**insectKinds**

描述： 昆虫的种类；

对象类型：Array；

缺省值：[

{point:，health:},

{ point:，health:},

{ point:，health:}

]

**newInsectList**

描述： 新昆虫的列表；

对象类型：Object；

缺省值：

**newListIndex**

描述： 标记新昆虫的序列；

对象类型：number；

缺省值：0；

**catchedInsectList**

描述： 被捕捉的昆虫列表；

对象类型：Object；

缺省值：

**catchedListIndex**

描述： 被捕捉昆虫的序列；

对象类型：number；

缺省值：0；

**that**

描述： insectManager的返回对象；

对象类型：Object；

**方法：**

**move**

描述：分别对newInsectList和catchedInsectList中的昆虫调用他们的move，并且检查newInsect的z轴，若为0则将其添加到碰撞列表，若小于0将其移除，以及newInsect的catchedFlag,若为1则添加到catchedInsectList，添加到碰撞列表,若标记为removed则将其从list中删除并解除settimeout；

参数：无；

返回：无；

**newInsect**

描述：产生一个昆虫，并 加入newInsectList；

参数：number；

返回：无；

**isInsect**

描述：检测一个点是否有昆虫；

参数：Object（{x，y}）；

返回：bool；

1. spider 蜘蛛类：

说明：蜘蛛的位置，移动向量，血量，包括织网，移动等动作。

传入参数：

gameCallback， //游戏响应函数，game对象方法

**属性：**

**health**

描述：血量；

对象类型：number ;

缺省值：100；

**x**

描述：蜘蛛x坐标；

对象类型：number；

缺省值：SCREEN\_WIDTH/2；

**y**

描述：蜘蛛y坐标；

对象类型：number；

缺省值：SCREEN\_HEIGHT/2;

**spiderRadius**

描述：蜘蛛半径；

对象类型：number；

常量：

**speed**

描述： 蜘蛛速度；

对象类型：number；

常量：

**collider**

描述： 蜘蛛碰撞实例，最后添加到SYS\_ collisionManager中的对象；

对象类型：colliderObj；

缺省值：

**state**

描述： 蜘蛛状态；

对象类型：Object；

缺省值：{

Moving：[0，{x，y}]， //为1则表示正在移动，xy表示移动位置

Neting：[0，[ {x，y}，{x，y}]]， //为1则表示正在织网或者正在赶去织网，

Eating：[0，catchedListIndex]， //为1则表示正在赶去吃虫子，

Static：0 //为1则表示正静止

}

**that**

描述： 蜘蛛的canvasSprite对象实例，最后添加到SYS\_processor中的对象；

对象类型：canvasSprite；

缺省值：

**moveVector**

描述： 表示单位时间移动方向和距离的向量

对象类型：vector2d；

缺省值：

**方法：**

**init：**

描述：初始化蜘蛛，包括改变蜘蛛canvasSprite对象实例的图片，画出蜘蛛，将蜘蛛的canvasSprite对象添加到SYS\_processor中；

参数：无

返回：无

**destory：**

描述：销毁蜘蛛，将蜘蛛从SYS\_collisionManager和SYS\_processor删除；

参数：无

返回：无；

**collision：**

描述：蜘蛛的碰撞检测函数，主要检测与昆虫的碰撞。

参数：insects；

返回：bool；

**hit：**

描述：蜘蛛检测到和昆虫碰撞之后的回调函数，加血加分等，要调用gameCallback；

参数：insects;

返回：无；

**that.move：**

描述：根据playerControl类判断玩家的操作，并对玩家的操作的合法性进行检查，即移动是否超出边界、织网的起点和终点是否在网和树枝上等等（若是捕食，则将蜘蛛的碰撞对象添加到SYS\_collisionManager中），并根据蜘蛛之前的状态对蜘蛛进行不同操作，辅以SYS\_timeInfo（控制游戏的帧数）、moveVector（蜘蛛移动的x和y方向上的距离）改变蜘蛛位置和蜘蛛状态，由SYS\_processor进行统一调用。

参数：无

返回：无

**getHealth：**

描述：获取蜘蛛当前血量；

参数：无；

返回：health；

**addHealth：**

描述：增加蜘蛛血量，并检查参数是否在合法范围（总血量不能超过100，若大于一百就保持在100），返回增加后的蜘蛛血量；

参数：number;

返回：health；

**reduceHealth：**

描述：减少蜘蛛血量，并检查参数是否在合法范围（总血量不能少于0，若小于0就保持在0，并调用gameCallback函数向游戏返回血量为0的信息），返回减少后的蜘蛛血量；

参数：number；

返回：health;

**getXY:**

描述：返回蜘蛛的位置；

参数：无；

返回：{x,y}；

1. net 蜘蛛网类：

说明：蜘蛛网的起点，终点等信息。

参数：sPoint{x，y}，ePoint{x，y}

**属性：**

**collider**

描述： 碰撞对象

对象类型：colliderObj；

缺省值：

**that**

描述： 蜘蛛网对象

对象类型：Object；

**方法：**

**that.hit**

描述：蜘蛛网被碰撞时调用的函数，暂时不用作处理

参数：无；

返回：无；

**that.draw**

描述：根据起点终点在画布上画出蜘蛛网；

参数：无；

返回：无；

**that.getPoint**

描述：返回网的起点和终点；

参数：无；

返回：[

{x，y}，

{x，y}

]

**that.remove**

描述：将直线移除，设置removed，并且将其从碰撞列表中移除；

参数：无；

返回：无；

**lineSplite**

描述：根据蜘蛛网的起点与终点，计算出蜘蛛网与碰撞网格的交点并返回；

参数：无；

返回：Array；

1. netManager 蜘蛛网管理类：

说明：管理蜘蛛网。

参数：无；

**属性：**

**netList**

描述： 蜘蛛网的列表

对象类型：Object；

缺省值：

**netListIndex**

描述： 蜘蛛网序号；

对象类型：number；

缺省值：0

**newNet**

描述： 正在织的网；

对象类型：net；

缺省值：

**that**

描述： 管理类的实例

对象类型：Object；

**方法：**

**that.move**

描述：将netList中的所有net都draw一遍，并将标记了removed的网移除，并且检测是否新的蜘蛛网已经完成，完成则将其添加到netList中，并将newNet置空；

参数：无；

返回：无；

**that.newNet**

描述：新生成蜘蛛网，并将其赋值给newNet；

参数：Object（sPoint{x，y}）,Object（ePoint{x，y}）

返回：无；

**that.onNet**

描述：判断一个点是否在蜘蛛网上，遍历所有netList中的网（初步方案）；

参数：Object（point{x，y}）；

返回：bool；

1. tree 树枝类：

说明：对树枝进行模拟，并且管理；

参数：无；

**属性：**

**treeMap**

描述： 树枝的映射表

对象类型：Object；

缺省值：{

‘0’：[0，1，2，3，4] //等价于0≤1X+2Y+3≤4

}

**that**

描述： 树枝的实例

对象类型：Object；

**方法：**

**init:**

描述：画出并初始化树枝；

参数：treeKind；

返回：无；

**onTree**

描述：判断一个点是否在树枝上；

参数：Object（point{x，y}）；

返回：无；