Science and Thinking

Jerry Chaos

2025 年 11 月 18 日

目录

Chapter 1

奇妙的数学

1.1 思考

数学中好多问题的解答，可能更希望是乘除运算而非加减运算，而一个简单的加减变乘除的方法就是：提取公因数。下面就是核心的用法：

$$ab - 13a = 0 \tag{1.1}$$

$$a(b - 13) = 0 \tag{1.2}$$

看似简单，但我们需要用的灵活，比如：

$$ab - 13a - 13b = 0 \tag{1.3}$$

$$a(b - 13) - 13b = 0 \tag{1.4}$$

$$a(b - 13) - 13(b - 13) = 169 \tag{1.5}$$

$$(a - 13)(b - 13) = 169 \tag{1.6}$$

如果说题目限定了a与b都是整数的话，那么就可以分解169为整数的乘积，从而分类讨论。

相加叫和，相减叫差，相乘叫积，相除叫商，商的整数部分叫模，余数部分叫余，指数结果叫幂，对数结果就叫对数。

欧拉公式 李永乐老师

$$e^{i\theta} = \cos\theta + i\sin\theta \tag{1.7}$$

$$\sin(a + b) = \sin a\cos b + \cos a\sin b \tag{1.8}$$

$$\cos(a + b) = \cos a\cos b - \sin a\sin b \tag{1.9}$$

$$x = e^{\ln x} \tag{1.10}$$

$$x = \ln(e^x) \tag{1.11}$$

Lambert W function：$W(xe^x) = x$

我们可以看这个视频：Harvard Entrance Examination Question — Solve for ‘b’ 。一般在题目中突然冒出来一个魔法值（具体的数字），大概率意味着它有变化，要么拆成几个数字之和，要么拆成乘积，具体要看整个表达式的表现形式，比如这个click here

$$x^2 - x^3 = 12 \tag{1.12}$$
$$x^2 - 2^2 - x^3 - 2^3 = 0 \tag{1.13}$$
$$(x-2)(x+2) - (x^3 + 2^3) = 0 \tag{1.14}$$
$$\text{Extract the common factor:} \tag{1.15}$$
$$(x+2)[(x-2) - (x^2 - 2x + 2^2)] = 0 \tag{1.16}$$

在解决数学题期间，我们需要尽可能的“拼凑”整齐的表达式，这些表达式可能是既有公式的形式，也可能是便于计算的形式（W函数是Lambert W函数，与ln或者log计算一样，可以使用计算器求得，计算器中一般叫做lambertW或者productLog函数，故此答案中直接使用W是能接受的）。下面的计算中在两边同时乘以ln(2)，拼凑了完整的LambertW函数参数的形式：

$$(5-x)2^{(5-x)} = A \tag{1.17}$$
$$(5-x)(e^{\ln(2)})^{(5-x)} = A \tag{1.18}$$
$$\ln(2)(5-x)e^{\ln(2)(5-x)} = A\ln(2) \tag{1.19}$$
$$W\left[\ln(2)(5-x)e^{\ln(2)(5-x)}\right] = W(A\ln(2)) \tag{1.20}$$
$$\ln(2)(5-x) = W(A\ln(2)) \tag{1.21}$$
$$x = 5 - \frac{W(A\ln(2))}{\ln(2)} \tag{1.22}$$

1.2 verlet integration

https://www.youtube.com/watch?v=-GWTDhOQU6M

Verlet integration与Euler intergration的本质区别是速度的计算，正因为Euler方法存在误差，是因为其使用了“瞬时速度”去计算“存在加速度下的一段时间”的距离，无论使用开始时刻还是结束时刻的瞬时速度，都是不准确的。而Verlet integration是利用此刻与前一刻的位置变化去计算速度，这种计算出来的速度是“平均速度”。

$$x_{n+1} = x_n + (v_n + a\Delta t)\Delta t \tag{1.23}$$
$$x_{n+1} = x_n + (\frac{x_n - x_{n-1}}{\Delta t} + a\Delta t)\Delta t \tag{1.24}$$
$$x_{n+1} = 2x_n - x_{n-1} + a\Delta t^2 \tag{1.25}$$

韦氏积分只需要保存前一次与本次的位置，即可计算未来的位置，不在需要保存速度信息。有人想到使用前后两者速度的平均值（见附录），这样一来会使verlet interation更加精确，当然计算量也会上升。这样的Verlet integration叫做速度V···I···

$$x_{n+1} = x_n + v_n\Delta t + \frac{a}{2}\Delta t^2 \tag{1.26}$$
$$\mapsto \tag{1.27}$$
$$x(t+\Delta t) = x(t) + v(t)\Delta t + \frac{a(t)}{2}\Delta t^2 \tag{1.28}$$

1.3 Taylor series

在Dr Tom Crawford的频道中，注重从Taylor系数入手，介绍其系数的由来。下面是$f(x)$在a处展开后的表达式：

$$f(x) = f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \cdots + \frac{f^{(k)}(a)}{k!}(x-a)^k + \cdots$$

如果$a = 0$，也就是在原点展开的话，Taylor Series也可以叫做Maclaurin Series，故：Maclaurin Series是Taylor Series的特殊形式：

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(0)}{n!}x^n$$

为什么要有“展开点(expansion point)”？Taylor能用多项式的形式无限逼近真实的函数图像，但这种逼近是有代价的，这里的代价要从两方面理解：

- 计算量指数增长的代价：导数越高阶，幂级数越高，自然算量就需要越多；
- 逼近指数增长的代价：并非相同步长的多项式和就能“固定”增加一定程度的函数图像“样子”，而是越想相似就需要指数级增长的多项式项去拟合；

逼近图像是从两端延申的，展开点就是蔓延的中心，级数项越多圆圈的半径就越大。如果你就想知道在原点展开的泰勒级数在$f(6)$附近的样子，与其选择大量项去计算多形式和，还不如直接在6处展开。这里有个Desmos的示例：

1.4 综合除法

1.5 向量值函数（Vector-valued Function）

向量值函数（vector-valued function）是指输入一个或多个实数，输出一个向量的函数。简单来说，它是一个以标量变量t或多个变量u, v为自变量、以向量为值的函数。

1.6 Jacobian determinent

https://www.youtube.com/watch?v=YqMelRryG8U

雅可比行列式用于空间变化后的系数计算，要理解其变换，我觉得一个重要的切入点是**偏导数**，当然这也是微积分的基础。回想一下一维函数的一阶导数，它表示因变量随着自变量的变化率，如果乘以dx，意味着：dx个单位的自变量引起多少个单位自变量的因变量变化，也就是$dy = f'dx$

如果在一维函数积分中，需要变换微分对象，那么就需要计算这样的变换系数，比如：

$$\int_0^2 (x)dx = \frac{x^2}{2}\Big|_0^2 = 2 \tag{1.29}$$

if x=2t: (1.30)

$$dx = (2t)'dt = 2dt \tag{1.31}$$

$$\int_0^1 (2t)2dt = 2(t^2)\Big|_0^1 = 2 \tag{1.32}$$

那么在二维积分中该怎么算这个比例呢？这就用到了Jacobian Determinant. 回到前面说的，我们需要正确认知导数的含义，在二维积分中，自然就是偏导数了（partial derivative)。Let's first define what a partial derivative is: If a function is a multivariable function, we use the concept of partial differentiation (aka: partial derivative) to measure the effect of a change in one independent variable on the dependent variable, keeping the other independent variables constant. To apply the rules of calculus, at a time generally, we change only one independent variable and keep all other independent variables constant. In this way, we only look at the partial variation in the function instead of the total variation.

从微分推导的角度看待

$$z = f(x, y) \tag{1.33}$$

$$\rightarrow \tag{1.34}$$

$$dz = \frac{\partial f}{\partial x}dx + \frac{\partial f}{\partial y}dy \tag{1.35}$$

$$\text{def: x=g(u,v), y=h(u,v)} \tag{1.36}$$

$$\int dxdy \tag{1.37}$$

$$= \int \left(\frac{\partial g}{\partial u}du + \frac{\partial g}{\partial v}dv\right)\left(\frac{\partial h}{\partial u}du + \frac{\partial h}{\partial v}dv\right) \tag{1.38}$$

$$= \int \left(\frac{\partial g \partial h}{\partial u \partial u}du^2 + \frac{\partial g \partial h}{\partial u \partial v}dudv + \frac{\partial g \partial h}{\partial v \partial u}dvdu + \frac{\partial g \partial h}{\partial v \partial v}dv^2\right) \tag{1.39}$$

$$\approx \int \left(\frac{\partial g \partial h}{\partial u \partial v}dudv + \frac{\partial g \partial h}{\partial v \partial u}dvdu\right) \tag{1.40}$$

$$= \int \left(\frac{\partial g \partial h}{\partial u \partial v} - \frac{\partial g \partial h}{\partial v \partial u}\right)dudv \tag{1.41}$$

上面有2个重要的地方：

- 忽略了高阶无穷小du^2与dv^2项，因为它们更快趋向于0；
- 加号变减号，这个与微分几何学中叉乘的顺序有关，前者是$dudv$，后者是$dvdu$，变成一致的顺序后叉乘需要改变方向[1]。

从向量变化与叉积的角度看待

设想有这么一个积分运算及其参数的变换函数，如何求取**J**？

$$\text{def: } \begin{cases} x = g(u, v) = 2u + v \\ y = h(u, v) = u + 2v \end{cases} \tag{1.42}$$

$$\text{solve: } \iint dxdy = \iint \mathbf{J}dudv \tag{1.43}$$

$\frac{\partial x}{\partial u} = 2$意味着**u方向上增加du长度，x会在u方向上增加2du个长度**，同理v方向也是一样的，只不过增加相同的长度。我们取任意uv空间下的点$< a, b >$，向着u与v方向增加du与dv个极小的长度，

[1] 具体细节需要进一步了解，查看微分几何学、函数行列式等内容。

UV空间变化量	X投影到UV空间的变化量	Y投影到...
$\langle du, dv \rangle$	$\langle 2du, dv \rangle$	$\langle du, 2dv \rangle$

那么x也会有一定的变化量，按照刚才提到的，x在u上的投影会增加$2du$个长度，而在v上的投影会增加$1dv$，同样我们也会知道y投影到UV空间的变化量$< du, 2dv >$，简单记作：

从线性代数的角度看[2]，UV空间变化的微面元面积：

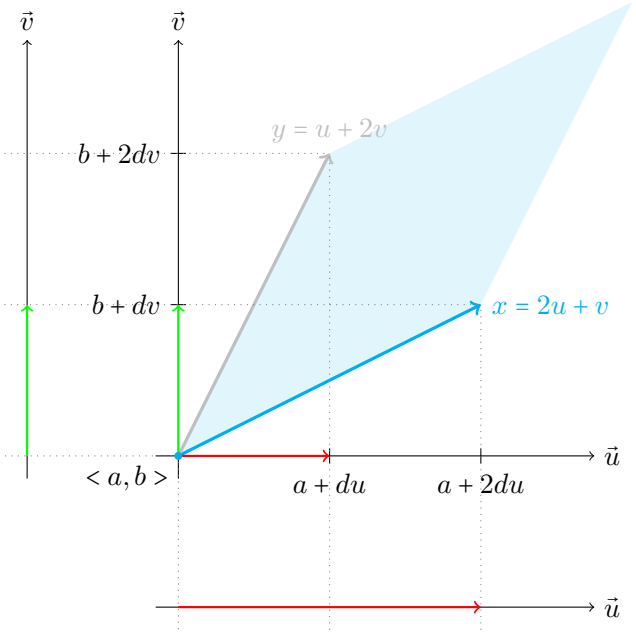
$$du \times dv = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \times \begin{pmatrix} 0 \\ 1 \end{pmatrix} = 1$$

而XY空间它们各自的变化量投影到UV空间下的面积是：

$$dx \times dy = \begin{pmatrix} 2 \\ 1 \end{pmatrix} \times \begin{pmatrix} 1 \\ 2 \end{pmatrix} = 3$$

这里重点强调了**投影到**，这意味着上述计算的结果是在同一个空间，故此它们的数值具有可比性[3]。上述是在向量空间，在标量下的意义就是$dxdy = 3dudv$，因此计算的J应该是3，整个计算过程使用偏导数进行，组成的矩阵就是Jacobian matrix，其行列式就是Jacobian determinent.

$$\textbf{Jacobian matrix} = \frac{\partial x}{\partial u}\frac{\partial y}{\partial v} - \frac{\partial y}{\partial u}\frac{\partial x}{\partial v}$$



从图中我们知道新的空间$B ::= 0XY$，是定义在原来空间$A ::= 0UV$下的，因为只有存在了A，才能定义新的标架，以及它们的方向与大小。经过向量叉积我们知道x, y向量在$0UV$空间下的面积是：$\hat{x} \times \hat{y} = (2, 1) \times (1, 2) = 3$ 如果我们将视野搬到空间B下，面积会是多少呢？显然还是用叉积计算：向量$\bar{x}$在B空间下是其一个基向量，也就是$(1, 0)$，同理，$\bar{y}$会是$(0, 1)$，那么它们的面积是1，那

[2] 导数变化率在微观上具有线性近似性

[3] 不同空间下的数值没有可比性，比如战国时期各个国家都有自己的度量衡，同一个物件赵国说3米，魏国说2米，虽然3大于2，但这种比较没有任何意义，必须是同一个度量衡，也就是这里说的投影到UV空间。

么我们答案是$dxdy = 3dudv$，得出基本结论：

$$\textbf{if:} \begin{cases} x = g(u,v) \\ y = h(u,v) \end{cases} \tag{1.44}$$

$$\textbf{then:} \iint 1dxdy = \iint \begin{vmatrix} \frac{\partial g}{\partial u} & \frac{\partial g}{\partial v} \\ \frac{\partial h}{\partial u} & \frac{\partial h}{\partial v} \end{vmatrix} dudv \tag{1.45}$$

1.7 最小二乘法（least squares method）

最小二乘法，又称**最小平方法**，就是在多个样本中，找到一个表达式，使其能尽可能拟合所有给出的样本，在此之后，就可以用该表达式计算未知参数的结果。基本的思想就是利用偏导数等于0，计算参数的值。

$$(1,6)\Delta(2,5)\Delta(3,7)\Delta(4,10)$$

$$y = \beta_2 x + \beta_1$$

$$\beta_1 + 1\beta_2 = 6 \tag{1.46}$$
$$\beta_1 + 2\beta_2 = 5 \tag{1.47}$$
$$\beta_1 + 3\beta_2 = 7 \tag{1.48}$$
$$\beta_1 + 4\beta_2 = 10 \tag{1.49}$$

$$S(\beta_1,\beta_2) = [6-(\beta_1+1\beta_2)]^2 + [5-(\beta_1+2\beta_2)]^2 + [7-(\beta_1+3\beta_2)]^2 + [10-(\beta_1+4\beta_2)]^2 \tag{1.50}$$

最小值可以通过对$S(\beta_1,\beta_2)$分别求β_1和β_2的偏导数，然后使他们等于零得到。

$$\frac{\partial S}{\partial \beta_1} = 8\beta_1 + 20\beta_2 - 56 = 0 \tag{1.51}$$
$$\frac{\partial S}{\partial \beta_2} = 20\beta_2 + 60\beta_2 - 154 = 0 \tag{1.52}$$
$$\beta_1 = 3.5 \tag{1.53}$$
$$\beta_2 = 1.4 \tag{1.54}$$

$$y = 3.5 + 1.4x \tag{1.55}$$

1.8 emtpy

一个n维空间下的向量，怎么投影到另一个n维空间？投影之后各个分量是多少？

1.9 deep learning

1.9.1 损失函数 loss function

$$\ell^{(i)} = \frac{1}{2}[\hat{y}^{(i)} - y^{(i)}]^2$$

1.10 common equations

$$a^2 - b^2 = (a - b)(a + b) \tag{1.56}$$
$$a^3 + b^3 = (a + b)(a^2 - ab + b^2) \tag{1.57}$$
$$a^3 - b^3 = (a - b)(a^2 + ab + b^2) \tag{1.58}$$

二元一次方程的解：

$$ax^2 + bx + c = 0 \tag{1.59}$$
$$\Rightarrow \tag{1.60}$$
$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{1.61}$$

logarithm[4]:

$$\log_n(MN) = \log_n(M) + \log_n(N) \tag{1.62}$$
$$\log_n(M^p) = p\log_n(M) \tag{1.63}$$
$$\log_n(M) = \frac{\log_a(M)}{\log_a(n)} \tag{1.64}$$

[4]more equations to: https://en.wikipedia.org/wiki/Logarithm

Chapter 2

chapter 2

2.1 求点集中指定方向的最远点

如何计算点集中指定方向上最远的那个点？这是计算机科学中一个很基础的技术，基础到哪里都会用到。

这是个很重要的基础技术，比如GJK，gift-wrapping 等算法都要用到。如图1.1所示，2D平面空间中有随机定义的一堆平面点（点集）。

Alt text 图1.png

假如我指定的方向总是平行于x轴，且方向与x轴正向相同。这样的话我只需要依次比较每个点的x坐标大小即可，最大的自然是指定方向最远的点，因为每个点的x值表征的就是这个方向上的远近（大小）。

图2.png

倘若方向不平行于坐标轴呢？如图2所示，蓝色箭头方向$d0(6.5, -3.8)$

最基本的思路是，将所有的点向这条方向（射线）做投影，那么可以比较投影的的某一项属性从而比较大小。如图3。

图3.png

问题可能出来了：我们知道点积（内积）可以让线段向另一线段投影，点投影貌似没有既定的公式。怎么办？我们就假想一个线段来来向蓝色方向投影。

思考这样一个东西(x, y)，它是什么。我们既可以说它是指定坐标系下的一个平面点，也完全可以说它是指定坐标系下的一个方向（起始点为原点，终点为(x, y)的有向线段）。如图4。

图4.png

现在我们就可以遍历点集中每一个点，假想它们是起点为原点的有向线段d1，然后与指定方向d0做点积（内积）向量运算，得到的结果便是d1投影到d0上的线段大小（带方向）。然后比较这个值的大小即可。如图5所示。

图5.png

2.2 differential equation

近期打算复习一下高等数学，一来巩固自己对高数旧有的认识、修复错误理解，二来学习新的知识。更重要的是希望能够拓展，这就是第三个目的。

2.2.1 概念

初等函数都是自变量与因变量之间的有限简单组合的等式关系，但是当该等式外加其导数关系的时候，整个关系式（方程）便演化为微分等式。微分方程就是其等式中含有自变量、因变量的方程，形如以下一般等式：

$$F[x+\varphi(x)+\varphi(x)^{(1)}+\varphi(x)^{(2)}+...++\varphi(x)^{(n)}]=0$$

对于这样的等式，当我们努力找出其中不含有微分成分的内容后，即仅仅是自变量与因变量的普通等式，就算是进了一大步，这个新等式叫做该微分方程的通解，该过程自然就是解微分方程。所以我们研究微分方程，其重点就是尽量找出x与$\varphi(x)$的关系，以后便是普通函数的关系问题。

2.2.2 思考

微分方程是数学中很重要的一个知识面，它的出现自然不是偶然，而是人类知识进步的必然。随着人类对客观事物更加精确计算的需求，微分方程开始展露头角。

初等数学中的题目基本都是建立在更加理想化的情况之下，比如喜欢边抽水边放水的路人甲来说，只要开关启动，水便马上就可以从出水口流出来，明显需要时间的一个过程偏偏马上就出来；再比如天上掉下来个馅饼，其速度明显不能越来越大但一定要说是均加速。当我们稍微具体点分析这些问题的时候，就会发现倘若没有进一步的知识便对其无可奈何。

我们简单分析一下掉落馅饼的情况吧：馅饼要下落，速度会因重力加速度而慢慢增大，逐渐增加的速度会产生相比之前阻力更大的空气阻力，导致速度的增长受到一定制约，制约后的速度依然产生不同的空气阻力，阻力依然要制约速度。。。如此看来整个过程便是个圈，于是我们就陷入这样一个困境中：**问题依赖于结果**

我们可以粗略的想出来这样一个方法：一开始速度为0，1s后速度变为g（重力加速度），然后将这1s内的所有空气阻力算个总账，速度g因空气阻力变为$g(1-k)$；依次类推后面的速度。这种方法在科学的计算中当然是错误的，但经常用在物理碰撞的时间步中，就因为其理解简单：最后算一个总账。但问题是空气阻力不是恒定的，而是变化的，随时都在变化的。这种变化亦不是恒定的。于是我们想着细分：半秒算一个总账，甚至1微妙算一个总账，这样做的用处除了让你或者计算机更加难受以外，没有其他意义，因为无论我们怎么细分，时间点都是离散点，都是可以更加细分的，没有最小，只有更小，都是试图用一个点阵算出精确的值，这显然逻辑上比较难以被认同。

要跳出上面的困境，一般是找一个第三者，大家都向第三者看齐，而不必彼此互看。这里的第三者便是时间，因为时间一旦确定，什么速度、阻力、加速度都会在那一刻成为一个定值。

2.2.3 具体问题

思考这样一个具体的问题（同济大学《高等数学》第七版上册 p302 第七题）：

一个半球体形状的雪堆，其体积融化率与半球面面积A成正比，比例系数$k>0$，假设在融化过程中雪堆始终保持半球体状，已知半径为r_0的雪堆在开始融化的3小时内，融化了其体积的$\frac{7}{8}$，问雪堆全部融化需要多少小时？

假设体积为V，他肯定是时间t的函数，因为随着时间体积会变小；

$$V=f(t)=\frac{2}{3}\pi r^3$$

其中半径r也是时间的函数，因为随着时间半径减小；$r=\phi(t)$题目说体积融化率是半球面积正比；

$$V^{(1)} = kA = 2k\pi r^2 = 2\pi r^2 r^{(1)}$$

上式可得：

$$r^{(1)} = k \tag{2.1}$$

$$r = kt + C \tag{2.2}$$

利用初值条件得到$C = r_0$至此半径与时间的函数关系$r = kt + r_0$就剩余一个常量k了，要让$r = 0$的话，$t = -\frac{r_0}{k}$，为题演化为求$\frac{r_0}{k}$的比值问题，再利用3小时融化$\frac{7}{8}$体积这个初始条件，很容易得出$t = 6$

2.2.4 齐次方程

许多数学科目中都提到“齐次”这个概念，笼统来说可以参见网络解释，这里需要强调的是，微分方程中的“齐次”指的不仅仅是各项变量次数之和相等，而是一定要化成下面形式。

$$\frac{dy}{dx} = \varphi\left(\frac{y}{x}\right)$$

求解一阶齐次微分方程的基本原则就是利用变量替换，将不能“变量分离”的状态，转换成能变量分离的状态。从上式可见，这个替换方案就是$u = y/x$，如此一来$\frac{dy}{dx}$可变成$\frac{du}{dx}x + u$，带入方程就得到“可分离变量的微分方程”；

2.2.5 一阶线性微分方程

对于未知函数y及其导数是一次的方程叫做一阶方程，如果常数项又不涉及y，那么就是一阶线性微分方程方程

数学上常用**常数变易法**来求解这种方程，常数变易法让我始终有意无意的联想到仿射空间下的移动：尽管常数项可以是自变量的函数，但毕竟与因变量y无关。纵观整个微分方程，其通解都会出现额外常数项，从而形成‘微分方程积分曲线‘，这条曲线会随着常变量的不同而平移到不同的位置，原来的一阶线性微分方程的解会不会是常数项在某一具体值下的图像。

整个**常数变易法**的过程，就是先求的齐次线性方程的通解，再替换常数项为自变量x的函数$u = \varphi(x)$，从而假设这个关于y的新的普通方程是原来一阶线性微分方程的一个特解，带入原方程从而解得变量u，就得到了方程的通解；

2.3 vector spaces

数学是基础学科，推动着几乎所有的其他学问前进，学好数学很必要，其目的是建立尽可能正确的底层逻辑。之前记录过自己对于空间变换的一些思考，这里再记一记关于不同空间的认识。

2.3.1 标量场

我们自从认识世界以来，较之向量——标量肯定出现的更早。人们使用标量无非记录各种数据，比如一个房间里面有各种家具，做家具的材料密度大多，桌子什么尺寸，室内多少温度，空气成分比例等等都可以用标量来表示，这个房间里充满了各种数字。我们把这些数字集合起来构

成了一个新的数学概念A，通过研究发现A中的任何两个数字相加或者相乘之后，其结果依然是概念A中的标量，于是大家称呼概念A为**标量场（scaler field）**。后来人们意识到有没有个标量加上自己或者乘以自己依然还是自己，这便引入了加法单位元0，与乘法单位元1的概念。

2.3.2 向量空间（线性空间）

现在我们可以用一个标量场表示这个房间中的任何一个具体的数字，诸如上面所说的什么尺寸、密度、温度等。现在你在房间外边，需要告诉另一个人去这个房间取出来一张桌子，你会怎么形容？如果限制在标量场下，你会告诉他：房间里面有一个50x40x23的桌子，你帮忙拿出来。倘若房间中恰巧有两张这个大小的桌子，你可能会额外加上木头密度为0.5的那一张，或者其他什么信息来表征你所想要的那张唯一的桌子。问题是：如果你告诉他密度了，也不能简单的靠肉眼看出来！于是慢慢的向量产生了。

认知向量空间一定抓住向量的一些性质：只有方向与大小，没有位置。向量空间是标量场的一个超集，不但有标量还外加了由有限的标量组成的向量。说到向量一定要说‘基‘，没有基的话，向量无从定义。‘基‘是n个线性无关的向量组成的，二维向量空间n就是2，三维就是3。向量空间对向量的运算也有一定的定义：

- 一个标量可以与一个向量相乘，结果是向量长度增加标量倍；
- 两个向量可以相加，结果是另一个向量；

就这两条（向量相减是相加的一种），现在身处房外的你可以说：给我拿来50x40x23的**头朝西**的那张桌子，明显更好点不是么。

2.3.3 仿射空间

我想大家应该也想象到仿射空间扩展了什么，没错就是’点’，点表征了位置，仿射空间定义说：

- 一个点可以与一个向量相加，其结果是另一个点；
- 两个点可以相减，其结果是一个向量；

仿射空间的到来使得人们不但可以让朋友进去拿东西，而且能更加精确的拿，更能让他将桌子朝着某个方向移动多少个单位等等。中学里我们见到了坐标系（二维或三维的），可想而知它是仿射空间下定义的，原因就是它里面有点的概念，而点就是在仿射空间下引入的。但同样中学里的坐标系也是个特殊的仿射空间，因为其三个基向量互相垂直（正交）。欧几里德空间我没有细看，听说增加了“距离”的概念，我想向量不是有大小的概念嘛，不能用来表示距离吗？oh，好复杂，就这样吧。

仿射加法

从上面得知仿射空间中没有定义**两个点相加**会是个什么情况，大想下也是，两个点相加是另一个点呢还是向量呢还是标量呢？就计图而言，我们在给两点（假如他们定义了一个线段）插值的时候，其表达式的形式就是类似两点相加（我们把结果强迫认为是另一个点），比如：$Q = P + n\vec{v}$（P沿着v的方向移动n个v的长度到达Q点），$\vec{v}$可以用$Q - P$表示，于是等式变为$Q = P + n(Q - P)$，进而$Q = nQ + (1 - n)P$，这在形式上就好像两个仿射空间中的点相加，因为仿射空间没有这样的定义，人们称其为‘仿射加法‘；

上面所言可能看起来有些不知所云，前提是对这些概念有一定了解。他们都是我在认知数学概念时候产生的各种想法，然后用比喻的形式试图合理解释这种概念的产生，纯属自己思维的认知历程。还是那句话：错不要紧，及时改正就是了。

2.4 inverse matrix

昨天在计算机图形学群中聊天，有人问下面的矩阵怎么求逆。看完觉得这岂不简单，用伴随矩阵就可以做到，但担心他可能有其他想法，便说“伴随矩阵啊，还是你想知道什么更简便的方法?”，后面就闲聊了几句。突然另一朋友写出这样一个公式，一开始给我愣在那里，什么意思一时间就是看不懂，等静下心来完整推导后一切明了，公式不错，点赞。

$$\begin{bmatrix} R, T \\ 0, 1 \end{bmatrix}$$

$$\begin{bmatrix} R, P \\ 0, 1 \end{bmatrix} \begin{bmatrix} R^{\top}, X \\ 0, 1 \end{bmatrix} = \begin{bmatrix} RR^{\top}, RX + P \\ 0, 1 \end{bmatrix}$$

因为R为旋转矩阵，所以$RR^{\top} = 1$，如果我们能使$RX + P = 0$的话，上面等号左边的第二个矩阵就是第一个矩阵的逆矩阵了。

$$RX + P = 0; RX = -P; X = -R^{\top}P;$$

将平移矩阵X带入上面的矩阵，于是就得到了逆矩阵：

$$\begin{bmatrix} R^{\top}, -R^{\top}P \\ 0, 1 \end{bmatrix}$$

我觉得这个方法应该源自矩阵分块化求逆的推演，分块化矩阵求逆有这样的公式（注意前提条件：A、B为可逆矩阵）：

$$\begin{bmatrix} A, C, \\ 0, B \end{bmatrix}^{-1} = \begin{bmatrix} A^{-1}, -A^{-1}CB, \\ 0, B^{-1} \end{bmatrix}$$

这里取矩阵B为1阶单位阵，再加上旋转矩阵是正交矩阵的特点，很容易得出与上面相同的逆矩阵。

2.5 微分中值定理与导数的应用

2.5.1 微分中值定理

费马（Fermat）引理

设函数$f(x)$在点$x0$的某邻域$U(x0)$内有定义，并且在$x0$处可导，如果对任意的x属于$U(x0)$，有$f(x) <= f(x0)$或$(f(x) >= f(x0))$; 那么$f'(x0) = 0$;

罗尔（Roll）定理

如果函数f(x)满足

1. 在闭区间$[a,b]$上连接；

2. 在开区间(a,b)内可导；

2. 在区间端点处的函数值相等，即$f(a)=f(b)$;

那么在(a,b)内至少有一点c，使得$f'(c)=0$;

拉格朗日（Lagrange）中值定理

如果函数$f(x)$满足：

1. 在闭区间$[a,b]$上连续；

2. 在开区间(a,b)内可导；

那么在(a,b)内至少有一点$xi;(a<xi;<b)$,使等式$f(b)-f(a)=f'(xi)(b-a)$;

泰勒中值定理（Lagrange中值定理的推广）

柯西中值定理

如果函数$f(x)$及函数$F(x)$满足：

1. 在闭区间$[a,b]$上连续；

2. 在开区间(a,b)内可导；

3. 对任一x属于(a,b)，$F'(x)!=0$;

那么在(a,b)内至少有一点$xi;$,使等式$(f(b)-b(a))F'(xi;)=(F(b)-F(a))f'(xi)$;

2.5.2 洛必达法则

当遇到求解$\frac{0}{0}$或者$\frac{\infty}{\infty}$的极限时（未定式），我们可以根据一定的前提条件，利用洛必达法则求解。

定理一

设：

1. 当$x->a$时，函数$f(x)$及$F(x)$都$->0$；

2. 在点a的某去心邻域内，$f'(x),F'(x)$都存在且$F'(x)!=0$;

3. $\lim_{x\to a}\frac{f'(x)}{F'(x)}X(^{-}:0\text{œ}$（不存在还计算它做什么）；

则：

$$\lim_{x\to a}\frac{f(x)}{F(x)}=\lim_{x\to a}\frac{f'(x)}{F'(x)}$$

定理二

设：

1. 当$x->\infty$;时，函数$f(x)$及$F(x)$都$->0$；

2. 当$|x|>N$时$f'(x)$与$F'(x)$都存在，且$F'(x)!=0$;

3. $\lim_{x\to\infty}\frac{f'(x)}{F'(x)}X(^{-}:0$（不存在还计算它做什么）；

则：

$$\lim_{x\to\infty}\frac{f(x)}{F(x)}=\lim_{x\to\infty}\frac{f'(x)}{F'(x)}$$

1	$e^x \sim 1+x$
2	$e^x - 1 \sim x$
3	$sinx \sim x$
4	$tanx \sim x$
5	$ln(1+x) \sim x$
6	$x - ln(1+x) \sim \frac{1}{2}x^2$
7	$1 - cosx \sim \frac{1}{2}x^2$
8	$arcsin(x) \sim x$
9	$arctan(x) \sim x$

表 2.1: 等价无穷小

2.5.3 泰勒公式

麦克劳林公式：

实例：

2.5.4 曲率

弧微分公式

$$ds = \sqrt{1+y'^2}dx$$

曲率圆，曲率中心，曲率半径；

2.5.5 方程的近似解

根的隔离，隔离区间，

- 二分法：找出一个隔离区间，折半查找f(x)的值，根据符号缩小隔离区域，继续折半查找、计算，直到得出想要的精度；
- 切线法：找出一个隔离区间。计算二价导数，从其符号确定选取的端点（函数值与二阶导数相同的符号的端点）。计算切线与y=0的交点，缩小区间，重复计算；

2.5.6 总复习题

2.5.7 不足

- 证明不等式能力弱，可能需要劲量寻找具体的函数;
- 有逻辑就按照方法证明;

2.6 math thinking

前一段时间在乱翻高数的时候看到了笛卡尔坐标系下的曲线微分公式，或者叫弧微分公式：$ds = \sqrt{1+y'^2}dx$，当时并没有想到极坐标下的形式。最近在某数学群中有人说起极坐标下的弧微分公式，并且给出了公式，说该公式很普遍，于是我想推导一番。

2.6.1 微元

微分中自变量与因变量的微元是什么意思？就是一个极小的变化量，他们之间还有一定的关系，这个系数就是导数。比如函数$y = 4x$，微元之间的关系就是$dy = 4dx$，也就是说自变量变化一个微单元，因变量就需要变化4倍于它的单元，并且这个变化是线性的（从函数曲线图中就能看出来）。再比如函数$y = x^2$，微元之间的关系就是$dy = 2xdx$，此时的自变量微元变化与因变量微元之间不是线性的，而是随着自变量具体值而不同。具体点说就是，取x=3，此时微元之间的关系是$dy = 6dx$。说这些的重点是：微元之间有关系，这种关系可能随着自变量的不同而不同。如论如何：微元就是变化量。

下面我简单记录下自己对其的推导。如右图所示，假设该极坐标系下的方程为$\rho = a\theta(a > 0)$，微元等式为$d\rho = ad\theta$，在任意θ与$\theta + d\theta$时刻该曲线上的两点BC如图2所示，图中已经标出了相关的变量。因为圆周弧长、半径与夹角的公式为$s = r\theta$，在微观情况下，其弦长与弧长是相等的，所以我们认为图中DB长度就是$DB = da = \rho d\theta$，又因为$d\theta$足够小，所以图中DB与DC是垂直的（不要在意图中的情况），这样一来弧长为：

$$CB = ds = \sqrt{da^2 + db^2} = \sqrt{\rho^2(d\theta)^2 + db^2}$$

公式中db是什么？就是我们上面分析的因变量的变化量，即：自变量从θ开始，增加一个$d\theta$后，因变量便从ρ增加了自变量的$\rho'(\theta)$倍，现在全部带入公式中。

$$ds = \sqrt{\rho^2(d\theta)^2 + db^2} = \sqrt{\rho^2(d\theta)^2 + \rho'^2(d\theta)^2} = \sqrt{\rho^2 + \rho'^2}d\theta$$

这样我么便得到了极坐标下弧长微分公式，其形式与笛卡儿坐标系下的弧长公式相同。从公式中我们很容易能推导出圆周长公式，设圆的半径为R，那么该圆在极坐标系下的方程为$\rho = R$，带入公式得到$ds = \rho d\theta$，积分得到周长为：$C = 2\pi R$

Chapter 3

computer science

3.1 2013-12-27 21:51:15 Git学习、复习随笔

这篇文章是在我*2013*年博文《*Git*安装随笔》的基础上修改完成的，没有大动，整理时候调整了用词、术语等。

3.1.1 概括

按照公司要求，新的3D项目及其库项目使用Git作为版本管理系统，之前有同事集中普及过一次Git的基本操作，这次受命于公司也算与Git[1]有缘吧。这里记录下自己学习Git的相关笔记与学习、复习心得；

Git是一款开源软件，采用分布式管理的做法，不再集中版本，而是将诸如SVN，CVS等管理版本的做法“移植”到了本地，这种做法有助于扩大开发者的自由度，理论上可以本地无限制的创建与销毁分支而不会影响到其他的合作开发者。这里说的是‘不再绝对集中’意思是Git虽然是分布式管理系统，但依然可以通过服务器来控制汇总，让合作开发者能够同步各自的最新进展。

Git服务端的安装我暂时不太清楚也暂不深入计较，但在使用本地Git时你需要知道这样一件事：如果你打算克隆一份Git服务器上的仓库或是提交你的程序到Git服务器，你都需要向服务器证明：‘你是谁?’，如果你不这样做服务器是不会接受你的提交请求(pull request)的。

如何要让服务器知道你是谁呢？Git的设计者做了这样一个策略：所有想要克隆或者提交程序的客户端都需要到Git服务器注册自己的信息，注册时粘贴由本地Git客户端生成的一串公开字符串（称为公钥‘public key‘，我们暂时称为A串）即可，这字符串便是客户端的ID。这样每当Git客户端提交文件的时候，会连同生成的另外一串字符串（称为私钥（private key），我们暂时称为B串）一起提交，服务器收到后经过算法计算出全新的一个字符串（我们暂时称为C串）后，与你之前提交的A串进行比较，如果相同就证明你是之前提交过公开字符串的客户端，他就会接受而不拒绝本次提交请求。

3.1.2 TortoiseGit.exe

TortoiseGit[2]的安装与使用无异于TortoiseSVN，除了部分界面不同以外，其他界面基本相同，推荐从SVN转过来的程序员使用；需要说明的是TortoiseGit是运行在Git的命令行的基础上的，也

[1] http://git-scm.com/

[2] http://tortoisegit.org/

就是说TortoiseGit是个外衣，其核心依然是Git客户端。这就要求我们如果打算使用TortoiseGit的话，需要先安装Git客户端。

3.1.3 Git.exe

Git客户端的安装相比TortoiseGit要复杂一些（仅仅是这两个软件的安装而言）。windows系统下的安装过程中会让你选择具体的内容，比如是否安装Git命令行'Git-bash'给其他pc机用户，是否创建PATH环境变量，是否安装Git界面(Git-GUI)（没错，Git开源组织也开发了Git的外衣，还有好多，比如github客户端），是否安装高级什么什么交互的插件,是使用openSSH还是使用PuTTY SSH等，各取所需，无须赘述。今天除了全面了解、安装Git以外，还熟悉了基本的Git-bash命令：

```
# 注册基本信息:
git config --global name="JerryChaos"
git config --global email="email@domain"
# 需要指定服务器git地址，首次使用后会在~/.ssh下生成know\_hosts文件，保存服务器Git信息，以便下次使用
git clone remote-git-server-URL
拉取服务器数据： git pull origin-repository
查阅命令帮助： git help <cmd name>
新建分支： git branch new-branch-name
删除分支： git branch -d exist-branch-name -d参数表示强制删除未合并分枝;
切换分支： git checkout exist-branch-name
查看有几个源： git remote -v
查看当前Git客户端版本： git --version
查看历史提交记录： git log
查看所有交互记录（包括历史提交记录）： git reflog
```

3.1.4 关于Gerrit审核服务器的启动

```
比如gerrit安装在: c:/gerrit/
启动: c:/gerrit/bin/gerrit.sh start -d c:/gerrit/
-d或者--gerrit\_site
```

3.2 2014-2015

这两周偶尔开会讨论我们工作的质量，就是针对3D库程序，开会期间有公司大牛参与我个人倍感荣幸，我觉得有必要记录下他们在分析、解释过程中说出来的一些让我受教的经验；这些经验对我来说有些是彻底全新的，有些便是加强了我对程序的认知。

不常用的成员方法参数应在函数外部管理（即将参数提升为类成员变量），并通过get、set方法进行赋值和访问；
把握好预处理（序列化）阶段，不要让其在运行时继续做大量解析工作；
引擎要做的是将复杂的、大量重复的计算、流程等统一起来，给引擎使用者提供一种非常简单，容易理解，方便使用的环境；
实现可以乱，但接口含义不能乱；
你不能把一个或者两个底层API简单的在自己的成员方法中封装一次，或者简单封装几句，这样的

话，我调用你的接口与直接调用底层API没有区别！于是你提供的所谓接口根本没有意义，功能更是扯淡；
统一的库，统一的引擎，定义要统一，不要在A类中dispose()是销毁函数，B类中destroy()是销毁方法；
既然使用了若字典，那就是说你对其有需求，不要在使用过程中用的跟个普通字典似的，那还不如一开始就使用普通字典，这样还快点；
我（开发者）要执行个函数，你（库程序）还不给我执行吗？
成员变量可以通过赋值为 null 来清空，但成员方法不应如此处理，否则显得不伦不类。记住：简化接口绝不是在这个上面；
“事件涉及的程序应该是异步的。”这句话我不太同意，同步也有使用事件的必要，比如在发布-订阅模式中，模型变化后派发事件，订阅者得到事件后调整界面显示或者更新自己模块的数据都可以使用事件机制。
起名依然还是准确并且精简的好，两个词：准确，精简；

3.3 2015-06-23

下面是些我此前零星的对编程的体会，并不连贯，是从我各个笔记中找到的，暂且归整于此，不排除继续追加的可能。

在写一个函数逻辑的时候，偶尔会看到某条语句在逻辑中间被调用，但其实际执行时间却在整个逻辑之后，比如一个延迟调用的语句。对于这种情况，我觉得应该将这条语句放在整个逻辑的最下面，虽然他们实际上没有区别，但对于阅读该逻辑的人员来说，放在下面就能让其非常直观的认为这条语句是最后执行的。这种将语句放在最后的做法我称之为：明确引导。

在阅读《祖玛》代码的时候，发现作者将路径坐标计算放在了游戏开始初始化的时候，此时计算出来的值是小数，而在用户玩游戏的时候，其计算大部分是整数计算，这的确是一种良好的做法。由此在开发的时候完全可以在“可被接受”的前提下，扩大小数到整数，从而在大部分计算中让计算机处理整数。比如“可被接受”的小数位为.001，那么我们就用1000扩大所有的这些小数，从而变成整数。

“如果他没错，就不要动他”。这是句在计算机科学中流传的话，说这句话的人可能年代非常久远（因为这句话一开始不是给计算机说的，后来计算机科学借用），所要表达的意思也可能绝非计算机行业，但新人新译。这句话是想表达在计算机编程里面，如果程序运行OK、没有错误的情况下，即便程序的实现着实非常恶心，但开发者或者维护者尽量不要尝试去随意重构程序，不然可能出现事与愿违的结果。我将这句话联想在编程变量的初始化中。当一个变量（全局或者局部）无论它现在是什么值，只要下一步不打算用该变量，那就尽量不要去重新为其赋值，这样做是有好处的：

既然都要初始化，那我在需要的时候初始化与最开始全部初始化在性能上没有不同，为何不把初始化的代码与其使用的代码放在一起，从而阅读清爽呢？如果你销毁了一个对象的资源数据，但他的逻辑数据（比如变量的值）还在一段时间内需要用到，此时如果你改变了里面的值，可能会产生新错误；不排除其他好处；注意我这里的意思绝非是否定程序一开始对变量的初始化，而是想表达一种编程思想；

写一个设置坐标的函数，用百分数表示从左向右总长的百分之多少，如果用负百分数表示从右向左，因为这样显然更统一些；

之前写对象池的时候，总是习惯于在类文件中写一个静态push函数，一个get函数，还有一个数组，用户总是调用get静态方法得到一个对象，方法自然是先尝试从数组中pop出来一个对象实

例，没有的话就new一个新对象返回。而当用户不用的时候我便调用push静态方法将其推进数组。还有一种方法就是维护一个free指针，它始终指向对象池中“空闲”状态的对象，池中每一个对象也维护一个指向池中另一个对象的指针，最后一个对象指向null，当free指针是null的时候，对象池就需要扩大。 Box2D扩大对象池的方法是全新创建一块2倍与之前池子大小的新池子，初始化里面全新的对象彼此之间的链条式引用，然后将之前池中对象的所有属性值拷贝过来。这样做的优点就是新的对象池中对象之间依然是一种顺序表的结构，遍历起来快，尽管插入删除慢，但是从需求来看基本不会有这样的操作。而我的方法势必打一开始就是个链表结构，适合经常需要插入、删除操作需求的应用；

封装一个功能可以有很多方法，比如用一个全新的数据结构去与这个功能模块交互，该模块不能与其他任何模块交互，这样就可以。但这也属于治标的方法，封装核心还是对外尽可能提供优质的、简明扼要的接口，从而自我封闭。

如果几个函数都使用同一个变量，那么我们至少有两种选择：1、将变量写成全局的形式；2、将变量作为参数传递；细细品味1、2种方法你就会发现，使用第一种确实能提高编写效率，但同样将该变量暴露给了更多的函数，倘若编写不慎可能会导致该全局变量在无意间被全新赋值；其次一个明显的问题就是我一向支持的编写方式，就是任何代码要给阅读者（包括以后阅读该代码的自己）一种尽可能简单、明确、清楚的引导（被我称作“明确引导”）。这种以全局变量写的方式明显没有以显示的给函数传递参数的方式更容易让读者明白“可能”会发生什么。比如int min（int，int）一看就能理解它是想返回参数中最小的。然而要是写成int min（），或者int min（int）；你便会很郁闷它是试图在与谁或者哪些数字进行比较；

3.4 2016-07-28

变动一个方法的时候，可以给新方法添加一个‘怪异’的行为，以便能在逻辑中很直观的被察觉到。这样你就可以第一时间察觉到哪些旧有的逻辑还在被使用；
内建类用来作为单例类创建时的唯一参数，这样就能客观保证不出第二个单例类实例；
销毁对象的时候记得只清除必要的连接，比如addEventListener或者dispose，不必要每个内部对象都执行销毁、断开引用，这样有助于提高性能；
记得固定字符串、数字等用常量表示，免得debug困难，因为编译器不会判断两个不同的字符串对于String类型有什么不同；
可以大胆的使用一个变量，但绝不可大胆的改变一个变量，除非你对其非常熟悉；
不要把逻辑写成一片，这样不方便阅读与重构，而且逻辑也势必很死板；
一次网络行为希望能有一条消息完事，不必有过多的消息；听同事说《xxx》项目的吃经验丹消息是一条对应吃一颗，这样的话如果玩家一次吃99颗经验丹，就意味着陆陆续续会收到至少99条消息，作死的姿态。
为什么不写成至返回一条信息呢？基于安全的考虑？我们这样分析下：如果说陆陆续续来99条消息比整体来1条安全，那么这99条的安全势必是基于“每一条都是安全的”为基础的，既然每一条都是安全的那么就一条好了。带宽、手机流量，前端流畅性都会有大幅度提升；
给变量起个优秀的名字；
什么叫逻辑与UI分离？就是如果你有哪怕一丝的逻辑未完成就千万不要参与UI的操作；
不要进行不必要的操作；如果一个变量经过之前的逻辑必定不会是空值，那么诸如先用if操作去判断其是否空值就起到了浪费一条或者几条指令的作用。
指定明确的变量类型；动态类对象成员在其寻址内容过程中势必没有静态类对象快，所以尽可能的去用静态类对象代替动态类对象；

集中同一种行为，有助于理解逻辑结构，使思路更加清晰；

```
clearItems();
var i:int=0;
Global.removeAll();
var item:MailItem;
for(i=0;i<mailList.length;i++){...}
```

上面程序除了Global.removeAll()的话，其他的估计是一种行为，但在执行一种行为的程序中间插入一个看上去完全没有关系的Global逻辑，导致上下文逻辑各种穿插，没有章法。阅读起来感觉思路不清晰，想到什么写什么，绝非优秀的程序；

虽然强转或者用as关键字将数组中的对象指定为具体的对象会比直接读取慢一点（实测：1千万对象会慢200ms，视软硬件环境而定），但强转后大部分IDE会显示出其属性与方法，方便我们调用，不容易造成bug。坏处就是程序员过度依赖IDE，以至于使用Sublime、Vim等编写的时候各种忘记类名、函数名等等，不过这些编辑器有各种强大的插件，功能慢慢齐全倒也不是问题；如果一个界面中两个UI控件（比如按钮）不可能同时被显示的话，为什么要创建2个然后再隐藏一个？完全可以视当时环境给唯一的控件指定不同的业务逻辑；常犯的错误：判断布尔型变量，赋值布尔型变量；

```
bool a=true;
bool b;
if(a)b=true;
else b=false;
```

底层库接口的设计更加偏向于大众化，比如下面三个接口函数，对于库接口的设计而言，明显第三个好。但当接口的设计提高到应用层的时候，封装出来诸如第一、二个函数是没有问题的。这是编程哲学上的思考，没有对与错；

```
function changeColorToRed():void;
function changeColorToBlue():void;
function changeColorTo(color:uint):void;
```

不方便阅读或者需要用户使用前先计算参数值的接口不是个好接口；
不可否认，需要的时候再创建的好处是：
节省不必要的资源开销；
增强前端表现的流畅性；
有可能不会用到它为什么一定要创建它；
倘若子类没有用到直接父类的任何成员方法与成员变量，就不要继承它，除非有必要，比如统一处理；
换一个角度看待战斗系统的话，就是：一个活动的更频繁、更复杂、更有序的UI界面，所以完全可以用相同的一套底层逻辑；
对外接口最好不要让全部参数都附带缺省值，这样无助于开发者对接口参数的理解，也不方便其他开发者阅读；如果全部附带默认参数，直观的视觉反馈是一个不需要参数的函数调用，从而给大脑错误的信息；
不要把底层接口基本没做任何处理的封装一层丢给高层使用，那完全等于脱裤子放屁；
注释也是一门学问，比如进入场景地图可能有好几次调用不同的函数才能完成进入，此时就不要用

类似“进入地图”，“进入地图前的逻辑”，“真正的进入场景”的语句去注释不同的函数，不但第三者迷惑，自己时间长了也会忘记；SP：Ruby语言哲学是“可视化”编程，就是用Ruby写的程序你甚至都不需要注释都能读得懂；
使用ADT编程，这也正是学习数据结构的目的。使用接口与组合编程，这正是OOP编程的本质；

3.5 2015-03-22

这里有个简单的基础题目：如何计算物体坐标系O中某点$p(a,b)$在世界坐标系W中的坐标？

估计大部分童鞋会说乘以矩阵，但问题是我们是如何得到这个计算方法的？或许是从书中看到，或许是从前辈口中得知，倘若你还没有进行过思考，建议你先琢磨琢磨，因为我觉得里面有无穷的乐趣，我可不希望自己的思想被他人先入为主。

我不管O坐标系中某个点在哪里，只要O与W重合，那么O中的点的坐标就是W中的坐标，这必须是一致的，此时我们可以说点$p(a,b)$的具体坐标值就是其在W中的坐标值。倘若O坐标系要离开W坐标系，比如要往W坐标系的x轴正向移动3个单位，那么O中的所有点便会被动的随着O坐标系的移动而同样向着W坐标系的x轴移动3个单位。如此一来O中的点$p(a,b)$便与W中的点$(a+3,b)$重合。再然后O坐标系向上（y轴正向）移动5个单位，理同上，于是上面的点对应W坐标系中的点$(a+3,b+5)$。紧接着O坐标系围绕W坐标系原点旋转theta度（逆时针方向），解释都是一样的：O中的所有点被迫旋转θ度（相对于W）。

当我们把这一系列联系起来看的时候会发现：所谓O坐标系中的点，一开始都是与W坐标系中的点同值对应（不是一一对应）的，只不过有的O坐标系像大陆漂移一样通过具体的变换（平移+旋转+缩放+切变等）跑到了其他地方，导致整个O中的所有点都经过如此变换。但是：O中的点的坐标对于O来说还保持的“漂移”之前的数据。于是我们假设这个O中的点的坐标是在W坐标系中定义的，那么这个点再经过上述“漂移”变换后应该与O中这个点在位置上重合。思路就理清了：想要将O中的点转换成W中的点的坐标，就需要乘以该“漂移”变换，要将W中的点转化为O中的点坐标就乘以“漂移”变换的逆变换；

更进一步，我们知道旋转矩阵是正交矩阵，但是平移矩阵（平移数据作为四维矩阵中的第四个维度）不是正交矩阵，于是我们一般通过邻接矩阵法去计算其逆矩阵，而不能使用转置矩阵。要知道用邻接矩阵法计算一个四维矩阵的逆矩阵是相对直接转置慢的多的计算过程，优化这个问题我们可以使用一个新的抽象数据类型，他包含了旋转矩阵与平移向量，我们使用仿射空间去计算子空间的移动问题，如此一来，要计算一个变形（变换）的逆变换的时候，可以先平移到W空间的原点，后乘以转置矩阵，再移回原来的位置。

3.6 2016-01-02

$u = Mv$表示：u标架是如何从v标架转换的。此时如果u标架下有一个点P，如果想要求得其在v标架下的坐标表示，就需要P右乘M的转置矩阵。

3.7 2018-03-06

3.7.1 性能优化

- 在追求性能的同时，建立扩展性强的软件结构；
- 配置序列化，有助于快读解析；

- 资源转码，至少有3个目的：1、适应具体硬件环境的架构；2、压缩；3、加密；
- 采用流式资源打包方式，减少硬盘的寻址次数与时间；
- 具体语言的特征，基于编译后汇编代码的的执行指令条数；
- 子线程分担主线程压力，也可以使用分帧的方式降低单帧的集中处理压力；
- 优化重要算法，采用必要数据结构加快针对性问题的处理速度；注意2、8原则；
- 不要偏激于单一方面的性能；
- 采用profiler；

1. AVL树的单旋转与双旋转 2. 伸展树（splay tree） 3. 二叉树
摊还时间。

3.7.2 哈希表

1. 分离链接法（separate chaining） 2. 探测散列表（probling hash tables） 1. 线性散列； 2. 平方散列； 3. 双散列；

3.7.3 二叉堆

1. 父节点的键值总是大于等于（或者小于等于）任何一个子节点的键值； 2. 每个子节点的左右子树也同样是一个二叉堆；

3.7.4 排序

name	时间复杂度	说明
插入排序（insertion sort）	$O(n^2)$	
希尔排序（Shellsort）	$O(n^2)$	分割了几次的插入排序，难点在于分割值的选取
归并排序（Mergesort）	$O(nlogn)$	
快速排序（quick sort）	$O(nlogn)$	

表 3.1: 排序时间复杂度

3.7.5 STL

STL三大件（容器、算法、迭代器）；容器与容器之上执行的算法之间无需彼此了解。这种戏法是通过迭代器实现的。迭代器提供了一种算法与容器相互协同工作的机制。一个容器可以生成一对迭代器来指定容器中的一个元素序列（可以是全部序列，也可是一个区间），而算法则对该序列进行操作。采用这种机制，使得容器与算法既可以紧密协作，又可以保持彼此“不知情”，这种“不知情”的好处，乃是C++高级编程领域反复强调的主题。除了上面的内容以外，STL还定义了大量辅助性的功能，STL的另一个优越性在于高度可配置性。

3.7.6 引用是别名而非指针

引用是一个现有对象的别名。用对象来初始化引用之后，那么对象的名字或引用的名字可以用于指向该对象。指针与引用之间的区别：

- 不存在空引用；
- 所有引用都要初始化；
- 一个引用永远指向用来对他初始化的那个对象；

3.7.7 数组形参

函数定义中，参数需要一个数组，但是实际传入中，只需要一个指向该数组的指针即可，这种从数组到指针的自动转换被赋予了一个迷人的技术术语：“退化”，即数组退化成一个指向其首元素的指针；(用函数名字做参数传递，也会退化成一个指针)，**指向类成员的指针并非指针，指向类成员函数的指针也不是指针**

3.7.8 Factory Method

工厂方法的本质在于，基类提供一个虚函数“挂钩”，用于产生适当的“产品”。每一个派生类可以重写继承的虚函数，为自己产生适当的产品。

3.7.9 条款31 协变返回类型（covarient return type）

一般来说，一个重写的函数与被它重写的函数必须具有相同的返回类型；但是协变返回类型提供了一种在编译器进行检查的机制，尽可能减小运行时类型转化后可能出现的问题。C++中子类不能继承父类的构造函数；父类的友元关系也不能被继承；

3.7.10 Framework

1. 通信；（模块间通信、与网络的通信）；
2. 本地文件读取与保存；
3. 如何封装；
4. 底层库的决策（库更新，变动等）；
5. 设计先行、文档同步跟进；
6. 发布文件的格式定义，预留字段方便扩展；
7. 抽象

酷爱Scala，喜欢造轮子，对函数式编程、并行程序开发、系统设计架构、代码之美等相关问题也有着浓厚的兴趣，同时非常希望能够靠写代码赚钱养老

3.8 2015-06-27 开发总结-从Cocos游戏《消灭星星》到Mirage引擎下的版本

几天前Boss叫我到办公室，说是希望将Cocos版本的《消灭星星》移植成使用我们Mirage引擎的版本。我当然欣然接收，因为我喜欢具有挑战行的工作，并立下军令状：一个星期搞定。在移植过程中，免不了磕磕碰碰，因为你总是在“原作者应该是这么个意思”的思想模式下进行，时间长了、见识的框架多了后，这思想便会稍微好点，一来自己认为的基本正确，二来就算错了用自己的思路也可以成事。如果你够牛逼，原作者的程序漏洞你都能发现并且修复，但有些坑是非踩不可的，比如Cocos的坐标系是在游戏窗口左下角的，而AS的确是在左上角。我整个工作是从简单了解Cocos框架开始的，紧接着将必须做的事情先行搞定，比如资源，我可不希望这些东西在我思考逻辑的时候来烦我。最后就开始拜读游戏逻辑。几遍下来后自然收获不小，开始琢磨哪些需要大改那些不需要。最后的结论是：我希望按照Cocos那种框架来搭建AS版本的逻辑，但着手大改关于特效与动作的逻辑（因为两者相差太大）。于是开始写AS关于Cocos结构基本相同的功能模块、逻辑，比如Cocos的Director、Scene等。一开是自然经行的顺利，当游戏可以在Mirage运行的时候，当初一厢情愿的错误想法这便就爆发出来了：这些错误基本全部来自细节方面。细节的东西就需要断点一步步的查找，因为他们是Cocos与Mirage的在架构上的不同，不了解的开发者根本不可能预知。慢慢感觉自己一周时间说的有点唐突（虽然结果确实是一周），于是我不得不拿出大量的自己非工作时间去延长移植时间。总结下来，至少有这么几点：

- 不了解的东西，不可一厢情愿的认为他是怎么样，有时候它可能会很糟糕；
- 对于未知事物的判断，可以从不同的方面采样，比如同事的口碑，网络的言语，自己的理解，产品出现的时间，官方对其的重视度等；
- 对这些事情时间的预估，尽量从自己的热情度、熟悉程度、处理内容的数量方面考虑，同时不要忘记墨菲定律；

3.9 2015-10-03 11:40:33 《乐三消》开发思维历程

这四天来写《开心消消乐》的程序，有所心得，记之。到目前为止，程序应该是被彻底推翻了两次，现在的是第三次。第一次失败的原因是自己对整个系统的抽象度不够，一开始便着眼于‘大局’，于是每创建一个新的类或者新的方法、变量，都试图考虑他们在全局下的定位。刚开始一路顺风，毕竟程序才刚刚起步，感觉自己‘大局观’很强，站的挺高，看的挺远，棒棒哒。可随着程序模块的铺开，游戏系统的复杂度这才慢慢体现出来，于是每次下手编码总感觉碍手碍脚，顾此失彼，导致的结果可能是拆了东墙补西墙，倘若拆了是往好的方向修补倒也罢了，问题在于系统的复杂度是以指数增长，复杂到一定程度后大脑势必应付不过来，也就不肯能‘往好的方向发展了’，于是这次失败了。

第二次我便缩小系统开发，着眼于具体功能上。一开始宏观的规划还是有的，但在初步确定后，我便将精力集中于‘下落系统’的上，所谓‘下落系统’，顾名思义就是开心消消乐游戏中的小动物肯定要按照一定的规则在每次消除后下落。我初步完成了类似倒沙子一样的算法，就是一个方块落地后，后继的方块成金字塔形慢慢摆放。demo运行良好，我用一张大的二维数组记录每个格子的状态，又用另一个二维数组记录小动物的状态等等。基本全部以矩阵数组记录任何有用的或者必要的数据。当我写完“失联格子”动态查找其他连通格子的算法后，虽有bug，但整体表现良好，可之后的测试开始让我头疼：表现良好的情况是消除一个格子的时候，当我尝试同一时刻消

除2个及更多的时候，整个系统便濒临奔溃，好比倒塌的砖块一样。我花了一个下午尝试去解决这个问题，但没有成功。于是我将问题暂时依然归结到没有较好抽象。

貌似我会情不自禁的将问题复杂化，这可能源自人类对完美的追求，但这是人之本性，因此这不是问题的关键，那么我为什么会将问题慢慢复杂化呢？漆黑的背景，下面一张水平面，一个球从高处下落，途中有个静态障碍物，这个小球会发生什么运动。我一遍一遍的想着这个画面，反思着自己。突然想到此前买的《复杂性思考》这本书，那是一本非常薄的书，还没来得及看。但从书名我立刻意识到是不是自己处理问题的方法不对，或者不够好，我缺少对问题复杂性的思考。我立马又联想到公司大牛此前在QQ群里面随口说的数据结构比算法重要云云的话（可能意思变了，此处不代表大牛真实本意）。我想到自己此前移植工作内容较多，忽视了自我对问题的思考过程。这绝对是愚蠢的行为!

想了好多，天也开始黑了，没有开灯，径直躺在床上。抽象、数据结构移植绕着我打转。不用多说，第二次失败的原因是因为自己对一个系统的抽象度不够，如果能深入抽象，正确抽象，并且配合恰当的数据结构，复杂问题就会变的越来越具体，从而各个突破；慢慢我有了今天第三套方案的思路。

下落的时候还有个问题就是哪个实体被先计算，因为先计算的话就需要决定它是否还能继续下落，不能的话就要锁定这个格子从而不让其他格子进入；这个问题一直困扰着我，是我将问题集中在了对entity的排序上，当然这的确是个解决方案。我的想法是写一套逻辑，让越是子孙节点的entity越排在前面以便越早计算他们的下落数据。这个问题最后得到了解决（但不是完美解决）。问题出在当下落的格子（小动物）与周边的格子发生连续三消后，新的格子该如何插入之前建立的消除序列中。要知道在我的逻辑中，建立起来的队列是不保存entity关于节点之间谁是子，谁是父的逻辑关系的。这个问题烦恼我半天。

在我读更新逻辑的时候，我始终感觉这个东西应该很好解决，总有一种呼之欲出的感觉。我脑海中始终觉得所有的entity既然都是同一帧更新到一个新的位置，那么它下一步能不能继续下落应该能计算出来，或者有办法得知的。庆幸的是突然“下一帧”这个概念蹦出了我的脑海，我想既然entity同一帧到达一个新的位置，这一帧肯定是不会继续下落，但是所有的entity在这一帧肯定是要被计算的。如此一来我可以在下一帧开始的时候决定新一轮的entity能不能继续下落不就可以了嘛。这样一来的话，我就不需要之前那个复杂的逻辑去建立entity之间的先后顺序。但是在之后的思考中马上否定了这个想法，因为一个entity能否继续下落是需要去判断的，判断的话就需要前后顺序，否则还是徒劳；

本来1.3版本打算把毒液与初级毛球弄出来，但是在一遍遍的测试过程中发现下落子系统存在严重的一个问题，就是1个动物会尝试占据另一个已经被占据的节点，从而引发错误。这几日我努力尝试查出这个bug的来源，但是一无所获。我越来越感觉这是个不得不优先解决的问题。问题的出现是在魔力鸟与直线特效组合使用的时候，因为这个时候你不确定哪些动物会被消除，而且消除后会有250ms的延迟。这个时候最容易导致重叠的出现。

源码与截图可以看这里[3]，如果你还能找到flash player播放器（高一些的版本），应该能运行。

3.10 2015-11-23 17:50 Mirage UIMaker使用指南（0.8.5）

3.10.1 介绍

Mirage UIMaker是一款基于Mirage引擎，纯AS3语言开发的开源游戏UI界面编辑器。它具有市面上UI编辑器的常用功能，同时兼具windows10全新的界面布局。其最大的特点是轻巧，它可以

[3]https://github.com/zspark/lsx

快速启动、快速编辑、快速发布，删除了程序集成开发的功能，使得UIMaker更加专注于设计。

3.10.2 特点

- 快速启动：UIMaker轻量级的设计使得工具启动速度得到了很大提升；
- 专注设计：大胆的去掉了程序与界面集成的传统编辑器，把精力集中于界面的设计上，细分软件系统的开发；
- 预览功能：设计界面的过程中可以及时预览，更加方便复杂组件的设计；
- 工程层级直观可见：从更大的尺度把握工程开发，及时掌控全局资源与层级关系；

3.10.3 使用指南

UIMaker界面上分布着8个区域，见图3.1。彩色文字与对应颜色区域明确表示，里面还有两个黑色区域，一个左上角的菜单区（点击后出现）、另一个项目管理区（点击后出现）。首次打开UIMaker，需要创建编辑项目，点击“新建项目”按钮（或者在菜单列表中选择“新建项目”按钮），在此输入项目名、项目位置、默认尺寸（默认值为960*640像素）点击确定即可完成创建项目。此时在“资源列表”选项卡中会出现刚才创建的项目，设计区“已存在项目”栏目也出现这个项目的显示。多个项目中，标有（**A**）的表示当前激活项目，在这里你可以选择点击不同的项目从而在他们之间切换；

图 3.1: 超闪（superflash） UIMaker工具截图

项目创建完成后，UIMaker会在指定的目录下创建以项目名为名字的子目录，然后在该子目录下面又创建四个子目录：

- **editor**：保存UIMaker的编辑文件，以.editor为扩展名，可再次打开继续编辑；
- **resource**：保存该项目使用过的或者正在使用的资源文件，当用户从磁盘其他地方指定资源后，UIMaker会拷贝该资源到本目录下，方便以后制作图集；
- **view**：该文件是界面最终在项目中使用的，扩展名.view；

- **view_cfg**：生成view子目录下所有view文件的配置文件，所谓以后项目所有资源的一项子配置而存在；

当前有激活项目时，UIMaker才允许用户创建编辑文件（.editor）。用户可以点击工具栏中的新建文件按钮或者菜单栏中的“新建文件”按钮，从而创建一个全新的编辑文件，此时设计区界面切换进入编辑状态。打开文件可以从项目管理区中editor目录下双击完成，需要说明的是：组件实例属性的变化是及时生效的，不需要用户点击类似“执行”按钮。

当创建了一个新文件后，用户可以从UIMaker左边的组件区中拖入想要使用的组件到设计区就可以创建一个该组件的实例，同时编辑器的实例层级区会刷新以显示最新的实例层级关系，编辑器右边的实例编辑区也会显示当前选中的组件实例的可编辑属性条目。

1. 文本输入条目：内置可以输入任何字符，具体组件有特别类型要求除外；
2. 数字输入条目：内置只能输入数字相关字符，比如0-9、-、.；
3. 布尔条目：只能勾选与不勾选，该条目定位的属性只有两种状态；
4. 定位条目：右边有个按钮，点击后，可针对不同的属性定位到不同的对话框，提供比较复杂的属性编辑；
5. 下拉条目：是为“多选一”而存在的，点击条目中按钮部分任意位置，可弹出下拉列表，选择并点击想要的子条目即完成操作；

用户对各种组件的属性编辑就是靠这五种属性编辑条目来完成的。当选定一个组件实例后，UIMaker会生成其属性对应的编辑条目供用户编辑，针对不同的组件，条目的数量与功能可能是不同的。

容器组件

容器组建是最基本的组件之一，它类似Mirage中的DisplayObjectContainer实例，具有组件最基本的诸如x、y、alpha、visible、rotation、mouseEnabled、scaleX，scaleY等8个属性，见图五。容器组建没有width与height属性，其大小是由容器内部子组件的位置与大小决定的。

动画组件（Movie组件）

动画组件是用来显示用MirageAnimator编辑器编辑的动画文件的，属性条目相较容器组件多了movieName可输入文本条目，这是用来告诉编辑器显示哪个动画。播放动画就需要用到动画资源（即序列帧资源），这些资源在设置movieName条目的内容之前需要导入UIMaker，否则UIMaker会弹出提示框。

现在说下如何导入动画配置文件与动画资源。点击菜单选择“设置动画配置文件”，在话框中“主配置文件名”是需要用户输入用ResTool工具编辑的资源总配置文件（具体操作参见ResTool使用指南）。“Movie配置文件Id”是指动画配置文件在主配置文件中的id编号（每一个用Animator编辑并且导出的‘动画配置文件’都需要在主配置文件中进行配置，从而得到该动画配置文件唯一的ID），“资源根目录”是指上述主配置文件所在的路径。当必要的路径、文件名指定完毕后，点击“确定”按钮，UIMaker会尝试将数据导入内存，成功、失败都会有提示。（注意：土豆库会自动配置好地区与语言目录，所以用户这里只能指定项目根目录，最终*URL*路径可以从“主配置最终*URL*”查看。）

图片组件

图片组件的类型是Bitmap，Bitmap有个显著的特点就是：可以指定九宫格放大，如果九宫格为null的话，就是缩放操作。Bitmap编辑属性多了texture与grid，此二者都是定位条目，需要点击右边绿色按钮方可进入编辑对话框。需要缩放的话，可以选择在“实例编辑区”对应的条目编辑，也可以鼠标左键按住编辑区域周边的八个蓝块之一拖动，因为我们暂时没有指定图片的九宫格数据，所以此时拖动的话是缩放原始图片以适应大小。接下来我们设置九宫格数据，点击grid9条目右边的绿色按钮出现“编辑九宫格信息”窗口，这里用户可以拖动窗口中的4条白线已达到九宫格定位的效果，定位好后点击“确定”按钮，回到设计区。此时界面中图片组建的效果发生了变化，当指定九宫格数据后，图片的高旷不再以缩放表示，而是使用九宫格纹理。

按钮组件

按钮组件中有类型图片纹理设置的定位条目，比如upTexture、downTexture、selectTexture、gird9，他们的设置与图片一样，这里不赘述。唯一与之前介绍组件不同的是多出来个textFormat定位条目，点击定位按钮出现“TextFormat属性编辑”对话框，里面有三个编辑条目，依次指定按钮上文本的**颜色、字体、大小**三个属性，点击颜色定位按钮出现如图所示调色板，其使用与经典软件如photoshop，flashprofessional类似。

输入文本组件

顾名思义，该组件会在运行时允许用户动态输入文本内容。其特别的编辑条目就是maxChar、mutLine与restrict，各自说明如下：

maxChar	表示用户最大可以动态输入几个字符
mutLine	表示该输入文本框是否允许多行输入
restrict	是正则表达式的字符串形式，目前使用字符串转换正则表达式的方式 可能会因为转义字符的存在而有bug，该功能新版本可能会屏蔽

滑块组件

滑块组件可编辑条目如上图所示，其中direction是个下拉条目，内容只有两条，表示用户想要该组件垂直可滚动还是水平可滚动。backTexture与backGrid9表示指定用户想要的滚动条底图纹理及其九宫格；handleTexture与handlGrid9表示指定用户想要的拖拽头。

文本组件

文本组件不同于输入组件之处是它不能与用户直接交互，除此之外，两者暂无明显显示上的区别；

3.10.4 多选

设计区的组件实例都可以通过左键单击来激活的，但也选择了一个，想要在UIMaker中多选的话，需要在点击其他实例之前按住shift键。多选之后的一处变化是实例编辑区变成了**左对齐、垂直居中对齐、右对齐、上对齐、水平居中对齐、下对齐**等控制按钮，而不再是具体的组件编辑条目。

快捷键	说明
Ctrl+Enter	预览
Ctr+W	关闭预览
Ctrl+C	拷贝
Ctrl+V	粘贴
Ctrl+X	剪贴
Ctrl+S	保存
Ctrl+Z	撤销
Ctrl+Y	反撤销
Ctrl+Minus	缩小设计区
Ctrl+Equal	放大设计区
Delete	删除激活的组件
Up	当前激活组件向上移动一个像素
Down	当前激活组件向下移动一个像素
Left	当前激活组件向左移动一个像素
Right	当前激活组件向右移动一个像素
Shift+Up	当前激活组件向上移动十个像素
Shift+Down	当前激活组件向下移动十个像素
Shift+Left	当前激活组件向左移动十个像素
Shift+Right	当前激活组件向右移动十个像素

表 3.2: UIMaker快捷键

3.10.5 实例层次结构

如果用户在编辑过程中发现组件A被组件B压在下面，但我们想要A在B上面，这个时候就需要改变组件之间的层级关系（实例层级区的组件越往下越在上面）。用户可以在实例层级区拖动任意一个到另一个上面释放，即可完成层级改变，具体来说分两种：

1. 如果将组件A拖至非容器组件B上面释放，则将添加A到组件B的上面；
2. 如果将组件A拖至容器组件C上面释放，UIMaker会弹出提示，告诉用户想要的更具体行为；

层级结构区还有其他两个小功能：

1. 左键单击可选激活对应编辑区的实例，方便定位；
2. 显隐按钮功能不同于各组件的visible属性，它是一种给设计者隐藏冗余组件的友好设计罢了；

3.10.6 快捷键

UIMaker目前支持快捷键操作，暂不支持自定义快捷键，全部按键参见表3.2。

3.10.7 文件预览与使用

当用户编辑完了界面之后，可以点击预览按钮查看整体效果，此时某些在设计区无法查看的属性也会表现出其效果，比如visible属性。UIMaker在预览的时候会在项目目录的view子目录下生

成tmp.view临时文件，可随时删除。想要保存的时候，点击主界面工具区保存按钮，会出现“保存文件”对话框，输入文件名并点击“确定”按钮后，UIMaker会在激活项目的view目录下生成对应的.view文件，这个文件就是具体项目使用的文件。使用过程很简单，主要两部分：第一步载入.view文件，第二步对文件中摆放的组件实例添加触发逻辑，比如按钮按下、文本框输入等。

```
//第一步载入界面
var url:String = "test.view";
UIMaker.load(url, this);
//第二步获取实例
var btn:Button = UIMaker.getDisplayObject(url, "btn") as Button;
btn.addEventListener(GestureEvent.CLICK, func);
```

大家可能想知道上面代码中“btn”是什么意思，这个就是获取view文件中具体实例的标志，它是在UIMaker中的编辑区给想要以后添加逻辑的实例自定义命名的字符串，如图3.8.1所示。也就是指定id属性的内容即可；

3.10.8 关于软件

UIMaker在MIT开源软件协议下发布。被授权人有权利使用、复制、修改、合并、出版发行、散布、再授权及贩售软件及软件的副本。被授权人可根据程序的需要修改授权条款为适当的内容。同时在授权人在其软件和软件的所有副本中都必须包含版权声明和许可声明。

3.11 2015-11-26 15:01 编辑器开发日志

在整理MirageAnimator（超闪序列帧动画编辑器）编辑器的时候，之前的同事对View（各个显示的模块）的编号不是固定的，而是在程序初始化的时候动态指定的，从0一直++。这种写法的优点就是不会造成编号的重复，而且容易在必要的时候插入一个View。回过头来想想我们在对实例设置编号的时候，一般都是将编号保存在一个类似id的变量里面，之后的逻辑一般或者全部是针对id来进行的，这就是说程序员，包括计算机本身不关心id的具体内容，既然如此为什么不在编程的时候动态指定这些内容的。我们完全不要叫真这中设计会对效率有多少影响，因为它仅仅是一种“面向程序员”的编程。

这两天重构了2个编辑器的代码，唯一的原因就是老代码对编辑器不友好，具体体现在IDE不能连接到源、程序员不能方便读写程序，甚至连第三方库程序的“可读性”都比不了。库程序好比另一个集团，好歹客户知道其集团地址，各个对外的部门。而这种不友好的编程风格就像一个黑洞，什么都看不到。之前公司大牛说他的编程经验之精华就是“敏捷开发”，唯快不破，也许大神戏言，因为他的言行最觉得是个“怪人”，但我对此不太同意。我总是有意识的将程序的架构与社会的建构联系起来，比如社会有管理机构，有公路，有公司，也有个人，这就好比程序中的各种模块，以及模块的功能与之间的通道，人就好比模块间通信的数据源。往抽象里说社会还有法律，有道德，这就是程序中对一个模块的定义，有时候一个模块的功能绝对不能超过该定义而执行逻辑，这就是法律，而有时候某些功能是互相协作的模块间谁都可以参与的，这就类似社会中的道德；这是对大型项目的拟社会比喻，但对于小型项目或者一个仅仅为了计算某个结果的程序而言，对社会的比喻就需要缩小为公司甚至更小。从我这个角度去思考的话，倘若编程的核心是“敏捷开发”的，那么社会结构有是什么？

出于flashAMF协议的方便性，公司各种编辑器经可能的使用它去序列化一些数据，其中包括编辑器编辑完成后将要输出的文件。如果使用这种方法的话，我承认对最终数据的保存与读取会是

#	类型	长度	说明
1	int	4	一共有多少个块
2	int	4	UTF编码下的显示对象类型名长度
3	UTFString	-/-	显示对象类名
4	Dictionary	-/-	属性的键值对，保存与字典
5	int	4	UTF编码下的显示对象实例id
6	UTFString	-/-	显示对象实例id
7	int	4	自己的编号
8	int	4	父容器的编号

表 3.3: .view文件格式，其中2到8是一个完整的块内容，之后依次重复n次。

相当方便，只要在程序中对相关的数据类注册一个别名，将数据与别名序列化进文件中，在正式项目使用该文件的时候反序列化读取即可。方法确实简单，但问题亦是可见。最明显的问题就是需要向客户暴露编辑器最终文件的数据结构，因为如果不暴露的话反序列化工作不能进行，暴露之后面临着结构化编程中对封装概念的无视与对安全性的无视，当然还有其他不好的地方就是：有时候，可能大部分情况下用户根本不需要你这个结构文件；

于是我决定抛弃对结构文件的序列化支持，而采用自定义序列化过程。这里记录下各个编辑器最终文件的序列化结构，一方面整理下结构本身，二来好方便查阅。

3.11.1 MirageUIMaker

主体由许多“块”组成，每一个块即是一个显示对象，块中包含固定的一些数据，比如类型，成员变量的键值对，自己的编号，父容器编号，自己的id（**id与编号不是一个概念，编号主要用来映射谁是谁的父容器，id用来在运行时获取该显示实例**）；

3.11.2 MirageEffector

从效率与文件大小2个方面考虑的话，我决定去掉对2D粒子属性名（一个字符串）的记录，仅仅记录属性的值，映射过程是固定死了的，比如第一个读出来的数据固定就是粒子初始红色，第二个固定是初始绿色，以此类推；

类型	长度	Mirage中属性名	说明
uint	4	-/-	记录粒子相关metadata【注一】
Short	2	-/-	记录该2D粒子属性个数
uint	4	-/-	UTF编码下的纹理名长度
UTFString	-/-	texture	该粒子纹理名（带扩展），路径固定
float	4	duration	粒子系统持续时长（秒）
float	4	emitterX	粒子x坐标
float	4	emitterY	粒子y坐标
float	4	emitterXVariance	x坐标浮动
float	4	emitterYVariance	y坐标浮动
float	4	speed	速度
float	4	speedVariance	速度浮动
（下续）			

表 3.4: .pex文件格式（1/2）

【注一】:
uint第一个字节记录当前该文件的版本，与编辑器版本不同，该版本是独立的，从1到255；
uint第二个字节第一位记录该文件是否为模板文件（1为模板文件，0反之），模板文件不允许通过编辑器正常保存途径保存，编辑后只可另存为；
其他位带需求逐渐补充；

3.11.3 MirageAnimator

Animator只有编辑文件没有显式的导出，编辑文件是能再次导入到编辑器中继续编辑的数据状态文件，而Animator最终导出的隐式文件全部只有一个（要么重新指定），它就是potato中MovieAsset类需要加载的动画配置文件，是个“人可读”的txt文件；这里主要说下编辑文件（.mvd)。 Animator没有大动，按原版继续。

3.11.4 坑

- 我序列化完毕的文件却不能显示出来，几番打印信息发现需要显示数字0的地方，有的显示0，有的显示0.00，马上意识到原数据可能是字符串，经过排查，问题就解决了，确实是字符串。
- Adobe的AS3编译器有这样的bug，就是变量名如果与这个文件import的其他类所在的包名一样的话，编译器会报告 *Warning:(130, 0) [MirageAnimator]: Definition name is the same as an imported package name. Unqualified references to that name will resolve to the package and not the definition.*‘警告，此时编译是可以通过的，但是如果你后续逻辑是使用该变的属性或者成员方法，编译器就会跑去那个包相关的地方找，然后就是报错了‘*Error:(131, 0) [MirageAnimator]: Call to a possibly undefined method xxx through a reference with static type Object.*，一个佐证就是在IDE中完全可以连接到正确的位置；

3.11.5 经验

当一个引擎对焦点的支持只有一个的时候，在底层库框架的设计上就一定要注意鼠标划入触发的“焦点对象”与文本输入时的“框焦点”的对象两个概念。这两个概念不能混淆，可以想象当鼠

类型	长度	Mirage中属性名	说明
（上接）			
float	4	lifespan	生命周期
float	4	lifespanVariance	生命周期浮动
float	4	emitAngle	发射角度
float	4	emitAngleVariance	
float	4	gravityX	水平重力
float	4	gravityY	垂直重力
float	4	radialAcceleration	法（径）向加速度
float	4	tangentialAcceleration	切向加速度
float	4	radialAccelerationVariance	
float	4	tangentialAccelerationVariance	
float	4	startColorRed	岂始红色
float	4	startColorGreen	
float	4	startColorBlue	
float	4	startColorAlpha	
float	4	startColorRedVariance	
float	4	startColorGreenVariance	
float	4	startColorBlueVariance	
float	4	startColorAlphaVariance	
float	4	endColorRed	结束红色
float	4	endColorGreen	
float	4	endColorBlue	
float	4	endColorAlpha	
float	4	endColorRedVariance	
float	4	endColorGreenVariance	
float	4	endColorBlueVariance	
float	4	endColorAlphaVariance	
float	4	maxNumParticles	粒子最大数（注意一个pex文件是一个粒子系统的）
float	4	startSize	起始大小
float	4	startSizeVariance	
float	4	endSize	结束大小
float	4	endSizeVariance	
float	4	emitterType	发射器类型
float	4	maxRadius	最大半径
float	4	maxRadiusVariance	
float	4	minRadius	最小半径
float	4	rotatePerSecond	旋转速度
float	4	rotatePerSecondVariance	
float	4	startRotation	起始旋转角度
float	4	startRotationVariance	
float	4	endRotation	结束旋转角度
float	4	endRotationVariance	
float	4	blendMode	混合类型，枚举
float	4	emissionRate	发射速率
(结束)			

类型	长度	Mirage中属性名	说明
Dictionary	-/-	-/-	用字典记录全部

表 3.6: .mvd文件格式

标划入一个list后原来可输入的文本框焦点丢失是一种什么样的体验。

3.12 2015-06-08 祖玛游戏相关技术思考

3.12.1 路线

祖玛的路径可以有好多方法，比如用线段模拟，这就意味着需要大量内存数据，或者使用贝赛尔曲线，这就需要大量计算。鉴于目前计算机性能与该游戏的复杂度，两者随便都可以，不过我还是倾向于使用贝赛尔曲线，因为它任意位置可导。而离散的点是不可导的，只能是最大程度的逼近。

由于祖玛的线路不是一条二次贝赛尔曲线能全部展示的。所以它可能是一个贝赛尔曲线的集合，我们可以使用一个数组记录该集合，数组中线路信息在载入内存解析的时候

3.12.2 球的方位

球在移动的时候，需要计算其最新的坐标与方向，两者肯定都是根据线路信息而计算。对于离散点线路而言，可以给每一个关键点外加法向信息（或者切向信息），这样可以在两个离散点之间进行线性法向插值与坐标插值，从而决定两个关键点之间球的方位。Beizer曲线的话可以让球记录一个插值系数，从而实时计算最新的位置，方向的话也可以通过两次计算得出。

3.13 2013-12-23 10:19:15 划线游戏demo

工作之余，简单做的一款划线小游戏，仅仅是个demo，在安卓手机上试了下，感觉不错，毕竟用手的其操作方式比鼠标舒服很多。

3.14 2013-12-29 19:09:11 GJK algorithm学习

GJK算法[4]，是由Gilbert、Johnson、Keerthi三人共同开发的一类迭代算法。该算法并不对参数本身做改变，而是利用参数计算明科夫斯基差，将两个点集的距离计算转化为一个点集与原点的计算问题，其迭代过程是不断判断逻辑上是否存在更有可能会包含原点的明科夫斯基子空间。

3.14.1 算法的理论基础

我们计算平面中两个点A，B距离的时候，最常见的方法就是用A向量减去B向量，然后再计算这个全新向量C的模，当我们进一步思考的时候就会发现新向量C的模其实又是向量C与原点O的距离。如此一来的话，我们可以简单的推导出这样一个事实：要计算平面中A、B两点的距离，可以尝试找到另一个点C，然后计算C与原点O的距离，也就是直接计算点C的模（三维空间亦是如此）；这句话的重点并不是如何去计算点C，而是说两个变量之间的一些外部关系可以变换到一个变量的

[4]https://graphics.stanford.edu/courses/cs448b-00-winter/papers/gilbert.pdf

内部关系；这是一个质的变化。现在我们扩展到点集中，假设有两个点集SetA(n)与SetB(m)，由于其差集中的任何一个点必然是从SetA与SetB中计算而来，那么当差集中有一个点与原点重合的话，就意味着SetA与SetB中至少存在一对点Pa与Pb，它两的距离为0（重合在一起），不然的话差集中这个点从何而来？这就是GJK算法的理论基础，它的核心思想就是尝试计算差集中的点与原点的位置关系，从而确定点集A、B之间的位置关系；计算几何中一个基本术语明科夫斯基差（Minkowsky minus）就与其相关。

边缘点	明科夫斯基差空间下形成凸包的点
差形	SetA中的点减去SetB中的点，所构成的明科夫斯基空间
差集	构成差形的点集

表 3.7: GJK算法相关术语

3.14.2 明科夫斯基和（Minkowsky plus）

先说说明科夫斯基和，其本质就是两个变量的加法运算，变量可以是但不限于一维、二维、三维向量，其和就是向量的对应分量和，结果自然也是个同纬度的向量；明科夫斯基和在几何上的直观表现就是SetA的凸包以SetB的凸包形状移动之后的扫略图形。当然会有一些修正，但这不是重点，也易于理解。明科夫斯基差可以看作是SetA与SetB关于原点对称点集SetB’的和，深层含义可以参考平面中两点距离的思考。

3.14.3 支持函数（support function）

我们现在知道两个点集$SetA(n)$，与$SetB(m)$，SetA中有n个空间点，SetB中有m个空间点；倘若SetA中的任何一个点去与SetB中的任何一个点尝试计算Minkowsky Minus，会是一种什么情况：就会产生nm个空间点，而这些点中必然有许多不是在$SetC(nm)$的凸包点集中（而在差形中间），这就意味着去计算这些点的Minkowsky Minus是一种既无助于我们最后的判断，又浪费大量的计算时间的操作，我们肯定要尝试去减少这样的计算，最好没有一个多余的计算，于是支持函数就这样出现了。我们使用支持函数去试图找到理论上一定在差形边缘上的点，最好是其顶点。怎么做到呢？我们知道，凡是差集中的点都是一个点集中的点去减去另一个点集中的点，倘若SetA在右边，SetB在左边，那么我用SetA中最右边（x值最大）的点减去SetB中最左边（x值最小）的点，其形成的点的x坐标肯定是最大（没有比它更大的），那么这个点p肯定就在差形边缘上或者就是个顶点，这样我们就找到了一个有价值的点。同理，当我们用SetA中最右边的点减去SetB中最右边的点，形成的新的顶点肯定也在差形边缘，因为再没有其他点的x值会比这个点的小。上下方向也是一样，我们甚至也可以用倾斜的方向。尝试提出他们的共性：都是给一个具体的方向，得出这个方向下SetA中的最远的点（或者最近的点），然后得到这个方向下SetB中最近的点（或者最远的点），他们做Minkowsky Minus计算后，结果点肯定在差形的边缘。

支持函数的具体逻辑

```
private function supportFn(SetA:Array,SetB:Array,dir:Vector2D):Vector2D{
        var mostFarInSetA:Vector2D=getFarthestPoint(SetA,dir);
        var mostNearInSetB:Vector2D=getFarthestPoint(SetB,-dir);

        var mkwPoint:Vector2D=mostFarInSetA-mostNearInSetB;
```

```
        return mkwPoint;
}
```

3.14.4 单纯形

有了支持函数后，我们便更近了一步，因为从此之后我们得到的任何一个点都会是在差形的边缘（或者是其顶点），接下来就是判断差形与原点的位置关系了。如下图，我们计算出了差形中的一个点M，但我们难以仅以M与原点O有位置关系判断出整个差形与原点的位置关系，所以需要我们再得到更多的边缘点。一般的做法就是简单的反转得到点M的方向，然后计算另一个点，这样做是有一定原因的，其一：反转一个向量的方向要比全新创建一个向量速度快；其二：从目前仅有的点M我们无法知道整个差形的形状（甚至是大概形状），如下图，差形可能与原点有好多关系。所以在这样信息不全的情况下，我们没有必要大费周折去计算一个自认为更优秀的点。那就仅仅反转下方向，计算第二个边缘点吧。有了点N 要想形成一个平面形状，至少要有三个点，组成一个单纯形，然后判断原点是否在其中，在的话表明SetA与SetB相交。

3.14.5 向量三重积

一般学习线性代数都会遇到三重积，但那个是标量三重积，意思就是先计算两个向量的叉积，其结果再与最后一个向量做点积，最后的结果是个标量，可理解为空间菱状体的体积；而向量三重积意思是三个向量全部使用叉积计算，其最终结果依然是个向量，就我所知的计算机领域（计算机图形学，碰撞检测）里面其用途主要用来计算方向；当我们找到2个差形边缘点后，第三个点的查找便有了一定的依据，因为我们的目的就是判断原点是否在差形里面，所以第三个点的查找肯定是向着原点的方向，否则便是徒劳。如图下图所示，红色箭头就是希望寻找的方向，它与A，B两点垂直，要计算该方向需要用到向量三重积，新的方向下会计算出来一个新边缘点，这样就形成了一个三角形，现在问题是如何判断这个单纯形（三角形）包含原点呢？

3.14.6 Voronoi域

Voronoi域是计算几何分支的一个重要概念，这里仅说二位空间下的Voronoi域。当我们将一个平面凸图形的每条边延长后会发现整个二维空间被分割成几个区域如图所示，这些区域中有的占有原来图形的一条边，有的仅占有一个点，有的却占有整个面，除此之外别无其他，这三种不同的区域就是三种Voronoi特征域，依次被称作边域、点域、面域。当我们计算出单纯形中的三个点后，能肯定的是其他边缘点肯定不会在点域，因为如果存在的话，该点域所在的唯一点便是单纯形中的一个内部点，这根使用支持函数计算边缘点相矛盾，故此倘若再有其他边缘点肯定不会落于当前单纯形的顶点域中，这样一来我们的判断就可以分割成判断原点是否落于面域之内。判断点落于那个Voronoi域最简单的做法就是判断点与组成图形边的位置关系，问题便简化为点与线段的计算，更进一步便是点积的正负问题。

3.14.7 迭代计算

当我们判断出原点位于某个点域内，便能一口得出SetA与SetB不相交；而位于面域内便是一定相交。边域的话就需要进一步计算，此时就要判断应该丢掉第一个得出的边缘点还是第二个（第三个是在第一、二个边缘点的存在下得出的，故此它比前两个更具有有效性，试想如果丢掉第三个点，再次计算不还是得到第三个点嘛）。判断的依据依然是原点落在了哪个Voronoi边域内，如图，

落在A域的话，自然丢掉A点；丢掉后便是第一次递归计算了，与之前一样找出点三个点，然后全新的判断一次Voronoi域域原点的位置关系，直到肯定的退出或者逻辑有个迭代上限而退出。

3.14.8 参考资料

3.15 2015-06-25 凸包及其算法

学习物理碰撞相关的理论，凸包是绝对逃避不了的一个术语。

3.15.1 凸包的概念

在一个实数向量空间V中，对于给定集合X，所有包含X的凸集的交集S被称为X的凸包。

上面概念就是说：凸包是凸集的子集，且是包含集合X的最小的子集；按照坊间通俗的话来说就是把所有的点用橡皮筋圈起来，橡皮筋形成的那个凸多边形就是这些点的凸包。

这样的直观解释不能说不对，但显然不太考究，他会将读者的理解集中在“橡皮筋”之上，认为那个多边形的边就是凸包的实体。但是我们从概念上理解的话明显实体还要包括里面的东西（最小的凸集）。故此凸包是整个面而不仅仅是边；

3.15.2 凸包的应用

凸包是计算几何中基本的概念，既是基本概念其应用自然不是具体的，主要看使用者对凸包的理解。最常见的应用比如使用凸包去计算围绕树林的篱笆墙的最短距离，深入一些的话比如在《探求二维凸包及其应用》[5]论文中作者提到用凸包区分两个点集是否可能被一条直接分开，再比如铸造新合金问题，乍看第与凸包没有关系，但作者还是巧妙的应用了凸包知识进行了计算。

这里简单说下凸包在CD(collision detection)中的应用。一个物理碰撞引擎驱动的肯定不止是简单的规则图形，而对于不规则图形，我们一般使用图形的顶点去定义它。对于三角形，三个顶点可以任意顺序[6]，但是顶点数量大于3的话，他们就必须要有一定的顺序，否则不但不能计算正确的表面法线方向，而且不能正确计算面积等属性。如图2所示，点集顺序按字母顺序。

倘若不做处理，碰撞逻辑误以为AB、BC、CD、DA是四条边，这显然是不对的。故此需要先对该点集进行凸包计算，我们并不是为了显示（绘制）凸包的轮廓而去计算它，而是使用凸包计算过程中形成的统一的极点顺序（极点就是在凸包边上的顶点）。假如我们从A点开始计算，那么能在凸包边上的CW方向的下一个顶点肯定是D，然后是B，最后是C。

3.15.3 凸包算法

凸包算法有好多种，比如卷包裹算法(gift wrapping algorithm)、快包算法(Quick hull algorithm)、格雷厄姆扫描算法(Graham-scan algorithm)、等。

卷包裹算法

如图3所示，假如我们有这样一个点集

首先计算出肯定在凸包边上的一个极点（极点就是肯定在凸包边缘上的点），假设我们取最左边的点I，然后我们的工作集中在如何得到第二个极点；

[5]徐瑞广，余志伟.中国矿业大学(北京）资源学院（100083）

[6]3个顶点的多边形虽然不可能造成边计算的错误，但仍然有可能造成表面法向方向的错误。故此依然需要排序，以方便用统一的逻辑处理；

从图4中可以看出，I的临近极点肯定是与直线$w = I(x)$夹角最小的点，图中向量u与w夹角为17.12度，而其他的点比如B则是48.68度，一次遍历下来后，夹角最小的肯定是H点，于是H点便是CCW方向下与I临近的下一个极点；后面便是个循环，因为我们找到了下一个极点H，再找它的下一个极点与之前逻辑一样了。直到我们找到的下一个极点与第一个I相同，此循环便结束；

这里有个问题需要说明的是，图5中点I、A、E好像在同一条直线上。当然感觉可能是未必准确，但如果他们确实共线的话，这就需要我们自己决定点A的去留了，一般情况下点A是去掉的，因为他就是线段IE上的一个普通点而已，就算碰撞解决，也是与IE做线段碰撞，并非与点A做点参与的碰撞。有时候我们为了提高性能，做一些取舍计算，假如点I、A、E不公线，但是角IAE（A是顶点）大于一定的阈值，几乎为180度，这个时候完全可以近似为他们共线，从而省去一部分计算量。

具体实现（AS3）

```
/**
 * 输入点集，得到在凸包上的点集；
 * @param pointSet
 * @param hullPointSet
 * @return true表示经过了计算，false表示没有经过计算；
 */
public static function handle(pointSet:Array,hullPointSet:Array):Boolean{
 if(pointSet==null || hullPointSet ==null)return false;
 if(pointSet.length<=2)return false;
 hullPointSet.length=0;

 //找到最左边的点；这个点肯定是个极点；
 var firstPoint:Vector2D=LAMath.getFarestPoint(pointSet,new Vector2D(-1,0));
 hullPointSet.push(firstPoint);
 //每次遍历的开始点，是上一个已经确定的极点；
 var Point_a:Vector2D;
 //每次遍历开始，是随机点集中的一点，该点在每次遍历之后是个极点；
 var Point_b:Vector2D=firstPoint;
 var d0:Vector2D=new Vector2D();
 var d1:Vector2D=new Vector2D();

 do{
  Point_a=Point_b;
  Point_b=pointSet[0];
  d0.resetComponent(Point_b.x-Point_a.x,Point_b.y-Point_a.y);

  for each(var v:Vector2D in pointSet){
   if(v==Point_a)continue;
   d1.resetComponent(v.x-Point_a.x,v.y-Point_a.y);
   var result:Number=LAMath.cross2D(d0,d1);
   if(result>0){
```

```
    //大于零，表示有更左边的点；于是跟新Pont_b为这个点，同时更新d0;
    Point_b=v;
    d0.resetComponent(Point_b.x-Point_a.x,Point_b.y-Point_a.y);
   }else if(result==0){
    //在同一条直线上;选择远端的点;
    if(LAMath.dotProduct(d0,d0)<LAMath.dotProduct(d1,d1)){
     Point_b=v;
     d0.resetComponent(Point_b.x-Point_a.x,Point_b.y-Point_a.y);
    }
   }
  }

  //一次遍历结束，Point_b记录的肯定是最左边的点（相对了d0方向）；推入极点数组;
  hullPointSet.push(Point_b);
  //如果最新得到的极点不是最初的极点，说明圈还没有封闭，继续找下一个极点;
 }while(firstPoint!=Point_b)
 return true;
}
```

3.15.4 算法分析

从上面算法中可以看出，整个过程至少包含嵌套的2次循环。外圈while循环用来判断本次do逻辑里面计算出来的极点是否为第一个极点；而do中的for循环主要是遍历所有顶点，如此一来Gift-wrapping算法的时间复杂度便是$O(n \times l)$，其中n是凸包极点数量，l是点集数量。进一步分析可知如果点集中的点全部在凸包上（比如圆），那么算法时间复杂度便是最大$O(n^2)$,所以卷包裹算法不适合计算极点密集型点集。

3.15.5 快包算法(quick-hull-algorithm)

看过《实时碰撞检测算法技术》中的快包算法与维基百科[7]中的算法，其中虽有部分出入，但本质一样：都是由内而外，膨胀到点集的边缘。不过《实时碰撞检测算法技术》一书中说为了快包算法健壮性，需要考虑两点：

- 首次逼近四边形计算的时候，需要考虑到其退化情况，比如一个三角形或者一条直线；
- 最远点不止一个的问题；

书中说为了考虑到四边形有可能退化的问题，需要算法注意首次逼近构造四边形的退化情况。这条明显过虑了，因为后期当图形被迭代‘充气’的时候，都是计算距离边（三维的话是面）最远的点，如此一来的话，先前的四边形就是画蛇添足，完全可以一开始就仅仅构造一条直线（边），然后向两边“充气”。第二条在我们熟悉了算法之后再看；我们还是使用前面的点集，来具体看一下快包算法的过程，点集如下所示。

首先计算出肯定在凸包上的2个点，比如是顶点H、E，它们构成初始边g，此时所有的其他顶点被分割成2部分：绿色与蓝色，（见图2）。后边要做的就是2部分，一部分遍历绿色的点直到完毕，

[7] https://en.wikipedia.org/wiki/Quickhull

另一部分就是遍历蓝色的点直到完毕，就拿蓝色来说。先找到距离直线g最远的顶点F，F肯定也是在凸包上的点，这个计算完毕后，肯定形成2条新的边h与i，后边的计算就与初始边的计算一样了，依次计算新的边对应的最远顶点，我们就得到了点C与B，至此蓝色点群遍历完毕。下面就是计算绿色的点，步骤与上面一样。

3.15.6 注意点

- 新形成的边（比如h边），它是怎么确定自己寻找的方向是朝向C而不是反方向？
- 寻找最远点的过程中，是要遍历点集的，如何减少遍历？

在我经验里面，所有涉及到“哪一边”的图形图像问题，一定要把握好**初始点**与**结束点**还有**方向**即可，比如我们明确在这个快包算法中，一律使用‘左边’，那么第一次找到的点F如果说要在初始边g的‘左边’的话，那么此时的初始边肯定是E为起始点，H为终点的边；当我们产生2条新边h与i之后，要保证以后检测的方向依然是“左边”的话，需要明确边h的起始点是E，终点是F，边i的起始点是F，终点是H；大家肯定能想到当计算绿点群的时候，只需要将初始边方向改变就行了。所以没有必要调用2次函数，甚至写相同逻辑两边，当然这不是重点；现在说下《实时碰撞检测算法技术》提到的第二条。见图4

当我们为某条边（图中是边g）寻找最远点的时候，发现除了点F外还有M, K, L这三个点并驾最远，这时快包算法的策略是取距离这条边（边g）两个顶点$\Phi E, H\Psi$最近的点，从图中看出，距离E最近的是L，距离H最近的是M，此两者选其一即可。有朋友可能会想到如果点K与点L距离点E一样远怎么办，比如E位于K，L两点的垂直平分线上？稍微分析一下就知道这种情况不可能发生，如果有的话当初的点E就不可能被找到，代之的应该是图中的点L了。专业点说就是当E，H确定了后，其他所有的点都肯定位于边g的面域中，而非点域。

3.15.7 具体实现（AS3）

3.15.8 算法分析

快包的时间复杂度最好的情况下是：

$$T(n) = 2T(n/2) + O(n) \tag{3.1}$$

$$= O(nlogn) \tag{3.2}$$

上面公式计算起来可能不太容易，可参考《数据结构与算法分析–C语言描述》书中对于快速排序的时间复杂度计算。话说回来，该算法正因为叫快包算法，其快‘quick‘就是来自‘快速排序quick sort‘，两者原理相同。最差的情况下是：

$$T(n) = T(n-1) + O(n) \tag{3.3}$$

$$= O(n^2) \tag{3.4}$$

3.15.9 质疑

这里[8]有篇关于凸包的博文，其中提到关于快包算法的时间复杂度最快能够达到$O(n)$的级别，我对此很质疑，快包算法就计算第一条分割线而言，怎么着也要$O(n)$的时间复杂度，更何况递归才刚刚开始。

[8] http://www.cnblogs.com/Booble/archive/2011/03/10/1980089.html

3.15.10 相关资料：

3.16 2015-10-27 Mirage开发环境配置

1. 从官网[9]下载Mirage Engine并安装，环境变量中也会出现MIRAGE_ENGINE变量。
2. 去github上下载MirageFBPlugin包（在线安装不需要）。
3. 以管理员身份运行Flash Builder 4.7，进入`help -> Install New Software…`
4. 首先需要添加一个新site，点击图中的Add…按钮，进入Add Repository对话框（**建议在安装插件前点击“Available Software Sites”，把下图勾选的两个无用项先删掉**）
5. 如果是下载到本地安装，选择图中的Local…按钮，并选择MirageFBPlugin所在的文件夹；如果是在线安装，则拷贝URL地址到Location，在Name框中输入site名，如Mirage，点击OK按钮。
6. 在上面第4项的Work with列表框中选择新添加的site，会出现插件。
7. 点击next，直到出现许可协议，选择同意。
8. 点击Finish，出现下面的签名提示对话框，选择OK。
9. 安装完成后，重启Flash Builder 4.7。至此，插件已经安装完成。
10. 打开Flash Builder 4.7的首选项页查看已安装的Flex SDKs，如下图，会出现Mirage SDK条目。如果MirageEngine安装有问题，这里没出现Mirage SDK项，也可以手动添加Mirage SDK。
11. 打开Flash Builder 4.7，进入创建项目向导可以看到Mirage目录。
12. Mirage Library Project项目可以创建一个类库项目，提供给其他项目引用并生成swc文件，Mirage Mobile Project项目可以创建手机项目。具体操作与flash类似，按照向导指引即可。

3.17 2016-04-04 面试杂记

在超闪工作了三年半，期间学到很多东西，但随着自己对职业生涯规划的逐渐明确与清晰，越来越感觉到是时候需要行动起来蹦向认准的方向了，虽然在超闪的工作任务目前不是很重，但这绝对不是好事。最近面试一家做引擎开发与游戏开发的公司，初试与复试就隔工作日的一天，这当然是我比较欣慰的地方。期间先是技术主管A的面试，随后是人事B，复试的话是技术VP，姑且称其为C。

3.17.1 技术主管

A的面试主要针对简历中明确写出来的内容。首先就是自我介绍，一来缓和求职者的情绪，二来准备进入面试交流。此期间我观察到A的举动：他并没有在我简述期间看简历，而是认真聆听，这比较另我意外，自然我觉得他比较尊重求职者。

图形图像的渲染管线流程

这个当是我的菜，这里就不记录了，有兴趣翻阅博客CG类文档。

[9] http://www.mobimirage.com/resources.html

引擎与原生flash的不同，是不是生成的swf文件就可以直接用引擎跑

Mirage与flash除了在显示对象上很大程度不一样外，其他对象与flash基本一致，而显示对象没有像flash那样有很冗长的继承层次结构，反而更加平面化，这样降低了Mirage对显示对象的管理难度。A听完很是兴奋，他好像得到了什么宝贝是的按照他的思路说了一遍我刚才的内容，我说是，A立马说“不错，非常好的架构”，有种想拍桌子的意思。这让我们的谈话开始变得轻松起来，我此时马上意识他A是个geek。

优化的做法？

- 从架构层开始就应该设计一种伸缩性强的框架。我在说这点的时候汉语“伸缩性”忘记了，说了英语flexibility，A补充说“伸缩性”。A听完后说这个与优化有什么关系，他感觉怪怪的。我当时的想法是这样才能方便重构，方便优化，这是优化的基本条件。初试结束后我感觉这点确实不太重要，甚至恰恰相反，好的伸缩性是因为让渡了性能的结果，面试期间说出来可能出于自己在超闪工作中要经常性的优化来自PC页游的项目程序，感性上认为“你伸缩性强点，好让我方便点，不然我的工作会进行的好艰难”。

- 配置文件序列化，直接将其在内存中的存储格式输出到硬盘保存，方便读取，至少省去了解析这么一个步骤；

- 将资源文件打包处理，打包的目的至少有三条：1）减少资源量，便于网络传送，因为要经过压缩；2）转码格式，以适应具体应用的最佳要求；3）统一IO处理；

- 如果某一帧压力过大的话，先确定压力来源，有可能来自渲染、有可能来自CPU的计算，有可能来自IO（A说还有一种，后来谈话说是cg），于是我们可以采用分帧或者多线程的方式；

- 针对不同的语言进行语言级的优化，比如Mirage引擎中用来创建对象要比new Object()性能高；

flash中多线程是怎么样的

flash中多线程与我们Mirage多线程重要的区别是，我们的没有消息通知，所以就必须一遍遍轮询子线程检测处理是否结束。由于多线程方式在具体应用中会出现非原子访问的操作，一般需要加上互斥锁。鉴于互斥锁的开销比较大，并且我们完全可以不让子线程参与游戏应用逻辑的处理这两点，我们便仅仅将资源从本地硬盘载入内存的工作交付子线程。子线程不能参与尤其与显示列表相关的操作，即不能让子线程创建个Sprite实例，添加到显示列表并且期望能正确显示出来。Mirage子线程轮训检查的工作会不停的增加内存碎片，具体的表现是内存不停增长，直到因Mirage从系统申请的内存不够导致gc逻辑的执行，才会释放掉这些临时的内存碎片。

开发AS3库都做了什么

人物、地形、摄像机、技能、资源管理、手势管理。

“有较强的文档编辑能力”是什么意思。

这是我简历中自我评价的一句话，因为我习惯写博客，对于排版、文字组织等方面应该较之他人强点。

能记住的就这些重要的问题了，A在结束后总结了我的现状：对AS熟悉，熟悉计图，但不了解JS，我说是，没几句后他便离开。我觉得他应该或多或少问问关于设计模式、数据结构等方面的知识，我觉得首先这些知识是哪门语言基本都需要的，其次我面试的岗位的工作内容就是开发龙骨动画编辑器，如果对设计模式熟悉的话，肯定是大有助于框架的搭建，至少能够更容易理解他们的框架。

3.17.2 人事的面谈

与人事的交流显得更加轻松，期间我说了句题外话“你的工牌怎么与你本人不像”以缓和当时思考棘手的人事问题的大脑，她的问题是“如何让人事在面试期间能判断技术人员到底（在技术方面）行还是不行？” 这个问题我最后的答案是“还是交给技术人员面试吧”，当然也出了点主意比如与其博弈，说白了就是诈唬。此后B介绍了招聘公司的组织结构，也简单聊了聊我目前公司的结构，B的问题就是很传统的那些：

- 为什么要离职；确定了个人发展的技术方向，就要努力靠近，超闪暂时不提供这样的岗位。
- 竞业协议；
- 你对加班的看法：这个问题我此前几年就思考过了，我的回答从两个方面入手，第一：加班本身是不合理的，因为8个小时工作结束后，就需要进行一些人际之间的交流，大脑的放松，体力的恢复，或者进行自我技能的修炼与新技能的学习；第二：加班不可避免，也不能抵触，但要合理加班，干个通宵，第二天睡觉算什么加班啊，应该是拿出自主的8小时中的一些时间（不要影响休息）去做原本计划2天或者1半做的工作，从而在一个相对较长的时间跨度上从宏观的尺度缩小工期，这才是合理的加班；
- 希望的薪资与现在的薪资；
- 还有什么问题问我；我问了薪资提升空间与职位提供空间；社保、公积金及其缴纳额度；节假日、周末上班情况。

与B的谈话结束后，被告知5天左右给回复，我便离开。回去后坐了不久，通知说与当天相接的下一个工作日复试。

3.17.3 VP面试

C是招聘公司的合伙人，性杨，据此后人事说在国内flash方面很有名。C显然比初试的A与B更加老气、沉稳。事后回忆他的问题也更加宽泛与尖锐，C并没有集中在具体的某一个技术点上，主要问了些Mirage引擎相关的问题，鉴于对公司产品品牌的维护，下面大部分仅提及C的问题而不作答，当然他的目的是了解我对Mirage的熟悉程度，从而得出一些他认为更加客观的结论(这当然是我的理解)。

- 你最得意的项目是哪个，具体都干了什么？
- Mirage就是个flash虚拟机？
- Mirage有什么坑？干了三年多，自我感觉都形成了避坑意识，工作的时候，下意识的就避让了坑的存在，感觉这样才是正常的，。。。让我说点什么好。
- Mirage目前瓶颈是什么，还有哪些可提高的地方？

- Mirage有什么优势？Mirage的优势我觉得主要是其前生来自8位，16位红白非智能机时代，大师对底层硬件的理解非常深入，能够最大程度的让硬件处理必要的计算，整体表现为内存占用相对低，渲染性能快等优势。其他的优势比如3D引擎方面Mirage是直接原生实现诸多引擎的接口，高层只需要专注游戏逻辑，而flash的3D接口，提供的函数更加类似OpenGL提供的硬件层的接口（当然OpenGL也有提供软接口的，它毕竟工作在图形显下驱动程序之上），如此一来需要在AS层首先搭建一个引擎（诸如away3D，flare3D等）程序，才能更快捷、方便的开发应用程序，这一层搭建是AS层面的，比起Mirage的C++/C层面来说，在性能上肯定会稍逊一筹。

- 如果我投入像Unity那样的人力与财力，你认为Mirage会成为下一个Unity吗？这个问题我从Mirage之父[10]李大师的性格出发，说了自己的看法。C强调：我们只谈技术。

- 给你一个问我问题的机会，你最想问什么？我最关心的其实就是能不能从flash平滑过渡到底层开发，当然我本次面试的工作内容应该涉及到一些底层。不过从C一再强调我不会JS的样子来看，可能不太顺利，至少一时间是这样。其实我觉得对我这不是问题，学习新技能是我乐此不疲的事情，更何况JS与AS有着同源ECMAScript。（20160411补充：今天与同事聊天说起这事，同事马上否定了我，说这个回答简直跟狗屎无异，用人公司想要听的是你有多大的才能能为其所用，而不是把他们当作技术的跳板。的却事后回忆这部分回答着实欠妥，其实我就是想真实的表达自己，说些违心的话非我所愿，但实际上：大家都爱听好话。）

- 你的优势。我从物理学专业出发，强调自己在引擎开发中的物理碰撞、图形渲染中的光照、光线追踪、矩阵、向量等线代方面有着科班出生的优势。第二个优势就是：真喜欢写程序，并不是“被逼的”。想想十一长假在家写7天程序的感觉那叫一个爽。

- 了解我们公司吗？

- 简历上写是项目经理，那么你可能不太多的涉及到技术研发？超闪的项目经理就是带领一些技术人员做开发。C秒懂。

与C的交流中，有二个问题是直接否定的，一是JS，二是谈话期间的一个关于Mirage作为一款产品的问题。面试结束后，另外一个人事（不是B）与我简单聊了聊，此后我我便离开。整体下来我感觉还行，姑且不说脾气秉性他们能不能接受，就单从问答来说感觉拿到offer的概率是65%左右，我的推理是这样的，招聘公司要求精通AS与JS，我暂时不懂JS的话，折一半，但同时我熟悉计图这个加分项，姑且加1分（满10分计），同时招聘信息写的明确，是招聘AS高级工程师，并不是JS高级工程师或者AS、JS双精通高级工程师，这样看来可能更加注重AS的能力，所以上面的折半补回来1分，这样7：3. 这是我尽可能客观的推测，我也不叫真（谁知道加分项是加1分还是半分，AS与JS是不是稍微偏重AS等）。

3.17.4 总结

面试之初我比较担心自己在面谈期间可能说的不会太多，一来我不善言谈，二来我不愿进行不必要的交流，所以问题回答了我不会再说写无关痛痒的东西。也不知道从哪里突然奔出来一句“平常心”，瞬间感觉良好很多。在“平常心”的光环下，我自然不太在意最后的结果，但一定要说说收货。这次面试最重要的心得就是：人一定要折腾，不折腾像死水一般，你会静静死去，折腾能给你带来活力，是生命欣欣向荣的体现。不要把这里的折腾理解的狭隘，应该宽一点，比如多交流，

[10]他自己这么称呼

多沟通，多与外界沟通，多参与活动（收费的、志愿的）等。折腾可能会给自己带来挫折，不过我觉得鼓励与精神层面的能力提高是更多的，这就是所谓的情商。

3.18 2016-08-21 Ogre此前电子笔记的拾零性整理

这种扁平化的架构思路让我想到了Ogre引擎对场景图的设计，基本上市面上已知的任意一款游戏引擎都有场景图这个概念，场景图用于管理场景中所有的对象（这里的对象不要与flash中的‘显示对象’联系起来），比如模型、粒子系统、公告板、地形等。场景图的设计一般都是多叉树的结构，就是一个根节点，挂载相应的一堆一级子节点，一级子节点又挂载二级子节点...。在非Ogre引擎的设计中，开发人员使用的Mesh类必须继承自子节点类，这样层次关系就会比较长，不便与程序重构、调整与功能的变化，因为变动跟节点相关的逻辑难免牵扯到Mesh及其子孙类的行为，同时在设计层面上也不合理，仔细琢磨Mesh与子节点类的关系，难道就因为Mesh要在场景中放置就必须要继承子节点？其次子节点的方法与属性跟Mesh基本没有任何关系，何不将他们拆分，互不影响，用组合设计模式将他们关联起来呢？Ogre引擎就是这么干的，整个设计就是扁平化的，更加合理，也容易扩展。

3.19 2015-12-21 坐标变换

今天有同事问我一个坐标变化的问题，问题很基础，但对于不熟悉矩阵变换的朋友来说就难以下手。问题是这样的：世界坐标系下有个容器（容器可认为是一个全新的局部坐标系），该容器原点相对世界的坐标为(x, y)，容器里面有个随机放置的蓝色方块，方块左上角相对容器坐标为(x_1, y_1)，大小为(w_1, h_1)，容器里面还有个红点，红点坐标相对容器为(x_2, y_2)。当方块围绕其中心点旋转A角度后，请问红点应该如何变换才能使其保持旋转前后相对于方块位置不变？即假如原来红点在方块右上角，方块旋转后红点‘看上去’依然要在方块右上角。

3.19.1 建立全新坐标系与旋转矩阵

先建立以方块旋转中心为原点的全新局部坐标系，该坐标系位于容器空间内。现在方块要旋转，方块的旋转其实可以看作是新坐标系下的物体旋转而非新坐标系本身旋转，如果看作后者，那就要涉及到容器空间，不然没有父空间何谈旋转？所以新空间中物体的旋转就是典型的笛卡儿坐标系下的旋转非仿射空间的旋转，公式就是典型的：

$$\begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix} \tag{3.5}$$

3.19.2 坐标变换

第二步要做的就是将容器空间下的红点坐标映射到刚才建立的全新空间坐标系下，我们知道方块中心坐标旋转前后始终不变：$(x_1 + w_1/2, y_1 + h_1/2)$，那么从方块局部空间到容器空间的变换矩阵就应该是:

$$\begin{bmatrix} 1 & 0 & x_1 + w_1/2 \\ 0 & 1 & y_1 + h_1/2 \end{bmatrix} \tag{3.6}$$

反之从容器空间到方块局部空间的变换就是其逆矩阵：

$$\begin{bmatrix} 1 & 0 & -(x_1 + w_1/2) \\ 0 & 1 & -(y_1 + h_1/2) \end{bmatrix} \tag{3.7}$$

如此问题便解决了，红点坐标(x_2, y_2)通过矩阵3.7变化到新建的局部空间内$(x_2-(x_1+w_1/2), y_2-(y_1 + h_1/2))$,然后在经过局部空间下的旋转变化3.5计算出红点在旋转后的位置（相对于方块的局部空间）

$$\begin{pmatrix} (x_2 - (x_1 + w_1/2))cos\theta - (y_2 - (y_1 + h_1/2))sin\theta \\ (x_2 - (x_1 + w_1/2))sin\theta + (y_2 - (y_1 + h_1/2))cos\theta \end{pmatrix} \tag{3.8}$$

最后将该方块空间下的坐标点变换到容器空间下，也就是与矩阵3.6相乘；

$$\begin{pmatrix} (x_2 - (x_1 + w_1/2))cos\theta - (y_2 - (y_1 + h_1/2))sin\theta + (x_1 + w_1/2) \\ (x_2 - (x_1 + w_1/2))sin\theta + (y_2 - (y_1 + h_1/2))cos\theta + (y_1 + h_1/2) \end{pmatrix} \tag{3.9}$$

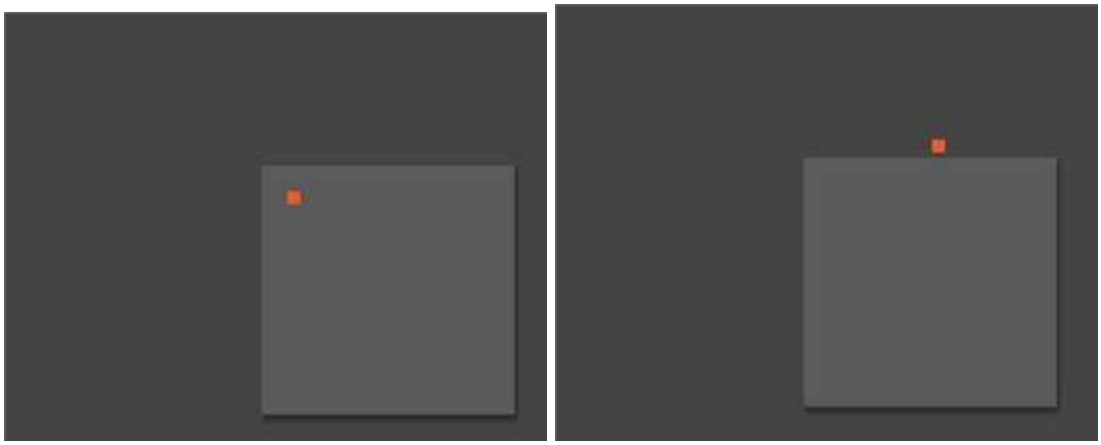


图 3.2: 以上变换全部是假设所有空间都没有缩放，除了方块旋转外，其他空间都没有旋转。

3.19.3 总结

以上变换其实就是仿射空间下一个最基本的子空间旋转问题，其基本思路就是现将子空间平移到父空间原点，然后旋转，最后再平移回原来的地方。

3.20 2015-12-29 Mirage Effector使用指南(0.8.3)

3.20.1 介绍

Mirage Effector是一款基于Mirage引擎，纯AS3语言开发的开源游戏2D粒子特效编辑器。市面上2D粒子编辑器比较少，而目前游戏开发特效的表现又越加被重视，Effect正是在这样的需求下立项并发布。Effect可以很快的设计出Mirage引擎支持的大部分粒子特效，可视化参数调节及时展现最新的表现效果，为你的游戏开发节约大量时间成本。

3.20.2 特点

- 快速启动：Effector轻量级的设计使得工具启动速度得到了很大提升;

- 及时呈现：Effector可以快速的呈现用户对粒子各项参数调整之后的表现，使得开发轻松愉快;
- 随时发布：一个特效生成一个文件，任何目录都可以保存，也可以随时读入编辑，灵活性很大;

图 3.3: 0.8.3版本Effector与粒子展示效果，z_Spark是我。

3.20.3 使用指南

布局

Effector界面上分布着5个区域，如下图所示，彩色文字与对应颜色区域明确表示，里面还有左上角的黑色区域，是菜单区（点击后出现）。

选择模板

Effector提供了39个2D粒子系统模板文件，全部位于程序根目录的template/目录下，该目录还有粒子系统需要用到的纹理png文件。用户点击菜单，选择“模板库”条目，就可以打开模板预览与选择窗口了，单击列表中的粒子系统条目，先从窗口右侧进行预览，方便用户找到与想要粒子系统效果相近的模板，等待确定后，双击该条目就可以在编辑器“效果展示区”出现了。打开模板文件的另一条路径就是传统的从“资源管理器”窗口找到想要的文件（.pex后缀），双击即可。具体操作参见“文件的保存、打开”栏目（见下）。

编辑粒子属性、设置底图

粒子系统属性的编辑很简单，绝大部分就是用鼠标拖动图四中红色区域所示的滑块改变属性值即可，只有三项属性设置与此不同，这里介绍下：第一个是为粒子系统设置纹理的条目。点击图中“纹理路径”按钮，打开“选择纹理”系统窗口，从这里可以找到计算机中任何地方的纹理资源（带透明通道的png资源）。选好后，点击“打开”按钮，就可以在编辑器查看最新纹理下粒子系统的表现了。同时刚才选择的纹理资源会被编辑器自动拷贝到根目录的“work”子目录下，从而方便后期处理；第二个是下拉菜单形式的“混合类型”与“发射器类型”条目，见图3.3.2。这与传统软件的下拉菜单无异，不多赘述。

第三个是设置“横坐标”与“纵坐标”的可输入条目，见图3.3.3。这种条目会在鼠标点击后高亮显示边框，意味着用户可以此时输入数字。这里需要注意的是：并不是每次输入都是可以即时呈现的，而是在输入完毕后回车才能呈现。

Effector的编辑区还有一个需要注意的地方就是：不同的“发射器类型”（一共2种），其“粒子发射器属性”面板中的内容是不同的，见下面两张图。这是因为不同类型的发射器有自己独特的属性（图中已列出对应类型的全部粒子发射器属性）。

效果控制

效果控制的所有交互组件都在“控制区”。这里的交互是从粒子系统数量、持续时间、间隔时间三个纬度去横向观察用户编辑的粒子系统。“实例数量”是指效果展示区中一共有多少个用户编辑的粒子系统（最少一个），见图3.4.1。“持续时间”是指效果展示区中的所有实例播放多少秒后停止播放；“发射间隔”是指这些粒子系统在停止播放后又经过多少秒再次全新播放。**注意：这三个参数并不会随着粒子系统的导出（保存）而存在于最后的.pex文件中，它们仅仅是一种效果横向上的展示。**

文件的保存、打开

当用户编辑完了自己的粒子系统后，可以通过单击主界面工具栏内的“保存”按钮（一个软盘类的图标），或者从打开的菜单中选择“保存”选项都会打开一个名叫“保存文件”的窗口，这里选择您想要保存的目录，然后输入需要保存的文件名（可不带pex扩展），最后点击对话框红的“保存”按钮，Effector会将最终的pex文件保存在这里。打开文件的时候不能从模板选择那里查找，而是应该点击工具栏内的“打开文件”按钮或者选择菜单中的“打开文件”选项，在出现的“打开文件”窗口内选择之前保存的pex文件，最后点击“打开按钮”就可以将选择的pex文件导入Effector进行再次编辑。

重点说下导入的pex粒子系统文件纹理获取的策略。目前所有非模板pex文件在导入Effector后，Effector会首先检测软件根目录下的work子目录内是否有需要的纹理文件（同名同类型），没有的话会使用template子目录下的default.png默认纹理；而所有模板pex文件在导入后会检测template子目录下是否有纹理文件。

文件使用

Effector存在的目的不是为了生成pex文件，而是在具体项目中使用pex文件，这就需要一个解析逻辑去处理pex文件，而这个文件就在土豆库中。2D粒子系统的使用非常简单，通过使用土豆库中的MirageEffector的静态方法::load()就可以获得ParticleSystem2D的实例。load()方法有两个参数：

1. pex的全局或者相对路径（相对于具体项目根目录）；
2. 该粒子系统需要用到的纹理文件所在的目录（全局或相对目录）；

当获得粒子系统实例后，就可以像使用Mirage中其他显示对象一样操作。

```
var pexURL:String="url/to/your/pex/file.pex";
var ps:ParticleSystem2D=MirageEffector.load(pexURL,"textureFolder/");
ps.x=500; ps.y=400;
addChild(ps);
```

3.20.4 关于软件

Effector在MIT开源软件协议下发布。被授权人有权利使用、复制、修改、合并、出版发行、散布、再授权及贩售软件及软件的副本。被授权人可根据程序的需要修改授权条款为适当的内容。

同时在授权人在其软件和软件的所有副本中都必须包含版权声明和许可声明。

3.21 2015-12-31 MirageAnimator使用指南(0.8.2)

3.21.1 介绍

MirageAnimator是一款基于Mirage引擎，纯AS3语言开发的开源序列帧动画编辑器。通过简单的拖拽、属性的设置、对帧的编辑来完成一个动画，解放出更多的程序开发人员参与游戏逻辑的开发中。Mirage Animator提供新建、预览、保存等功能，可以随时看到动画的样式，导出来的格式方便Mirage提供的解决方案使用。

3.21.2 特点

- 快速启动：MirageAnimator轻量级的设计使得工具启动速度得到了很大提升;
- 简单编辑：MirageAnimator通过简单的拖拽、属性的设置、对帧的编辑来完成一个动画，简单轻便。解放单纯无聊的一遍遍程序中参数的调整;
- 预览功能：设计动画的过程中可以及时预览，方便开发;

3.21.3 使用指南

3.1 布局

Animator界面上分布着7个区域，如下图所示，彩色文字与对应颜色区域明确表示，里面还有左上角的黑色区域，是菜单区（点击后出现）。

添加资源

在开始使用Animator之前，首先要将序列帧资源拷贝到asserts目录下，没有该目录可以新建，也可以运行Animator让程序为您创建。拷贝到asserts下的资源没有目录结构的约定。拷贝完毕后就可以打开Animator进行编辑了，点击资源库asserts扩展三角箭头，就能看到刚才拷贝的所有资源，左键单击可以在“预览资源区”查看资源内容（目前支持png与jpg），如图3.2.1。

导入资源到展示区

右键双击一个资源后会将该资源导入展示区，Animator会自动识别可能是序列帧的资源，并弹出提示告诉用户是否同意将一系列资源导入展示区，见图3.3.1。选择“确定”的话，Animator会将当前双击的资源连同后续的资源一起导入（**注意：这里是后续资源，之前的资源尽管可能是一些列的，也不会导入**），并且填充时间轴上从帧头开始的后续空白/关键帧。选择“取消”的话，只会导入双击的唯一资源。

编辑资源数据

用户可以点击展示区中的资源，然后在右侧编辑区进行资源偏移、高宽、缩放数据的编辑，图3.4.1。

时间轴的操作

时间轴就是位于软件正下方的有幕轨的一块区域，如图3.5.1。

时间轴区域里有针对时间轴的编辑按钮，依次是：**清除帧、复制帧、粘贴帧、剪切帧、插入帧、删除帧**。还有下方的幕轨区域，其中名称见下图

编辑按钮说明

- 清除帧：如果当前帧是个关键帧，则删除当前帧的内容，使其成为空白帧；如果是空白帧则不予理会；
- 复制帧：复制当前帧里的内容到剪贴板，等待粘贴帧时候使用；
- 粘贴帧：粘贴剪贴板中最近一次复制、剪切的关键帧内容到当前帧；
- 剪切帧：将当前帧中的内容剪切到剪贴板，当前帧执行清除操作；
- 插入帧：在当前帧后面插入一空白帧，其原本后面的所有帧（空白或者关键帧）依次后移；
- 删除帧：删除掉当前帧（包括其内容）；

幕轨操作

用户可以拖动帧头左右移动进行当前帧的改变，这种方式亦可用来简单预览序列帧播放的效果；也可以左键单击不同的帧（空白或者关键帧）直接选择想要的当前帧；

保存、打开编辑文件

当用户编辑动画期间需要将当前的内容作为一个外部文件存储的时候，就需要保存编辑文件。保存按钮有两处可供选择：一处位于主界面工具栏内，形状像个软盘的按钮，另一处位于菜单栏内，从文字中清晰可见。点击保存按钮后，Animator会弹出来“保存文件”窗口，用户选择好目录，并且输入想要存储的名字后（名字可以不带mvd扩展），点击“保存”按钮，即可将资源内容保存至选中的目录下。同样，想要打开一个已保存的mvd文件的话，点击“打开文件”按钮（与“保存”按钮所在位置一致），选择好文件点击“打开”按钮即可。**注意：mvd文件中的资源依然是从asserts目录下查找的，所以不要轻易删掉不确定使用与否的资源。**

发布动画

尽管mvd文件中有保存的数据，但是这些文件仅仅是用来再次编辑时候使用的，它们不是最终的文件。最终发布动画的时候需要依次点击：菜单-发布动画，在弹出的“发布动画”窗口中输入一个全新的动画配置文件名，就会将本次编辑的动画数据保存到指定的文件里面，如图3.7.1。

用户可以在“保存文件”窗口中选择之前创建的某个动画配置文件，从而将本次新编辑的动画内容追加在文件后面，如图3.7.2。一般的做法并不是每个动画文件在发布的时候都创建不同的文件，而是选择使用追加模式。保存的内容其实非常简单，如果是创建新文件，就给新文件写入图3.7.3中的全部内容；如果是追加文件，就向旧有文件追加写入图3.7.3中的第二行内容；

文件使用

首先用土豆库资源管理器导入所有的资源配置文件，然后执行下面这条，将动画配置文件导入项目内存；

```
MovieAsset.appendConfig(movieCfg.txt);
```

movieCfg.txt就是Animator发布的动画配置文件。执行完这条语句后表示文件中所有的动画配置数据已经导入内存，想要在舞台上显示的话，首先新建个Movie实例（来自土豆库）,然后指定movieName属性为想要显示的动画名即可（movieName就是图3.7.3中诸如default_这样的数据），代码见下。之后对Movie实例的操作就如同其他显示对象一样，具体接口参见土豆库Movie组件。

```
var m:Movie=new Movie();
m.movieName="default_";
addChild(m);
```

3.22 关于软件

Animator在MIT开源软件协议下发布。被授权人有权利使用、复制、修改、合并、出版发行、散布、再授权及贩售软件及软件的副本。被授权人可根据程序的需要修改授权条款为适当的内容。同时在授权人在其软件和软件的所有副本中都必须包含版权声明和许可声明。

图 3.4: superflash动画编辑工具

3.23 2016-06-10 Prototype.JS

3.23.1 类：Ajax.PeriodialUpdater

PrototypeJS针对Ajax的函数中有个让我新鲜的API参数，该接口叫Ajax.PeriodicalUpdater，其参数（可选）为frequency(Number,default=2)与decay(Number,default=1)，frequency表示间隔多少秒后全新发送一次请求，鉴于网络传输时间的异同，这里的间隔并不是发送到下一次发送，而是上一次发送返回到下一次发送开始；decay表示发送间隔延迟的倍数，默认为1表示永远只间隔2秒发送下一个，如果是2的话，第二次请求是2秒后，而第三次请求就是$2 \times 2 = 4$秒后了，第四次是$4 \times 2 = 8$秒后。这个间隔的增长并不以为这可以在越来越长的间隔内做些“坏事”，当上一次返回的数据不合法的话，请求间隔会重新从默认值开始。**PeriodicalUpdater与Updater不同，没有继承关系**

3.23.2 类：Ajax.Request

Request就是个对一次通信数据、回调函数的包装结构，它有着相对简短的生命周期：

- Created：创建
- Initialized：初始化
- Request sent：请求发送
- Response being received (can occur many times, as packets come in)：接收响应，可能触发多次；

- Response received, request compelte：全部接收，完成；

官方文档强调了一下onComplete事件。无论成功与否，都会在最后回调完成事件，加入还有onSuccess事件的话，那肯定是先onSuccess后onComplete。onSuccess会在下面二种情况下被调用：1. 收到的网络状态码不识别；2. 收到的网络状态码为2xy家族；**如果请求的数据符合同源条件，并且符合以下MIME-types的话，Prototype.JS会在收到数据后调用eval执行，所以你甚至都不需要在Request实例中写回调函数。**符合的MIME-types为：

- application/ecmascript
- applicqation/javascript
- application/x-ecmascript
- application/x-javascript
- text/ecmascript
- text/javascript
- text/x-ecmascript
- text/x-javascript

3.23.3 类：Ajax.Updater

Updater类是Request类的子类，用于在收到回应数据后刷新指定的DOM元素的内容，构造函数参数有evalScript，表示是否先执行回应数据内部的JS代码，后刷新元素内容。可以将第一个参数使用一个对象代替，这样可以加入success与failure两个键值对，表示针对不同的响应对不同的DOM元素进行刷新。

3.24 2016-06-20 格子引擎想法

首先说一下，我试图设计一个尽可能高效的格子碰撞系统，整个表现效果可以直观的参考《冒险岛》游戏。

下面记录下一个粒子与格子的碰撞，不考虑粒子之间的碰撞。此前一直将格子的种类限制在具体格子上，比如墙格子、软墙格子，梯子格子，斜面格子等。这样一直困扰着我，比如东西站在平面上往前走，前面是个向下的斜面，如何让其继续贴着斜面前进？粒子是不是要实时检测下面格子的类型，不然如何知晓下面的格子是个斜面格子？诸如此类。我现在的想法是对格子的分类不再局限于具体的东西，而是扩大了范围，出现了一些“抽象”的格子，比如委托格子，传感器格子。

3.24.1 基于格子的碰撞

格子引擎或者说基于格子的碰撞系统，它的核心必须是用当前对象所处的位置及其该位置下格子的类型，决定对象的位置与行为。基于格子的碰撞没有优秀物理引擎的宽检测与精细检测等不同的阶段，可以说它就只有宽检测阶段，这样势必效率高，

表达式	说明
\d	匹配所有数字
\D	匹配所有非数字（上面一项的反面）
\w	匹配数字、字母、下划线
\W	（上一项的反面）
\s	匹配所有的空白字符，比如空格、制表符、回车符等
\S	（上一项的反面）
\r	匹配回车符
\n	匹配换行符
\t	匹配水平制表符
\v	匹配垂直制表符
\u????	匹配以此指定的unicode字符，比如\u002B就是+号
\x??	匹配以此指定的ASCII码字符，比如\x61就是小写字母a

表 3.8: 正则表达式中部分转义单元

3.25 2016-05-31 正则表达式

有人说会正则的程序员比较神秘，也有人说使用*vim*的程序员更加牛逼，于是我觉得自己好普通，因为我既会正则又熟练使用*vim*。

正则表达式可能对大部分初学程序的朋友而言是陌生的，因为用不到，用到了也有其他方法变通的解决，其次就是比较少见，实在需要的话也是网上搜搜或者请教前辈。网上有很多优秀的正则表达式讲解，有的非常不错[11]。我主要说说以下几点，目的依然简单：我需要加强记忆与理解：

3.25.1 术语与说明

- 表达式：一串字符，用来表示某种含义。
- 匹配：指符合给定的正则表达式。
- 匹配内容：指通过正则表达式匹配后得到的被验证的字符串中的某一子串。
- 正则表达式在匹配的时候是逐字符去验证，只要符合就验证下一个字符，不符合重新匹配正则中的第一个字符。

3.25.2 转义

所谓转义，就是符号含义并非其字面意思，比如\w表示匹配所有的数字字母下划线，而\d表示匹配所有的数字，更多参见表3.8。

3.25.3 重复

重复就是再一次匹配，比如字符串"abc"，如果我想匹配这个字符的话，按照到目前为止的写法，只能写成这样：/\w\w\w/，重复3次，表示匹配验证3次，上面说了正则表达式引擎是逐字符扫描的，当扫描到字符a时，与正则表达式中的第一个进行验证，发现字母a就是\w的一种，然后继续

[11] http://www.regexlab.com/zh/regref.htm

向后检查第二个字符b，发现它也符合正则表达式中的第二个\w，于是继续扫描，直到正则表达式的结束，一路都是符合要求，所以字符串abc被选中。这样的话显然比较粗鲁，倘若我需要匹配一长串数字的话，那需要多少个\d了？更何况我岂能预先知道它会有多少个数字!

表达式	说明
+	前一项正则的内容必须至少出现一次
*	前一项正则的内容可有可无
?	前一项正则的内容可无，可有的话也只能出现一次
{n}	前一项正则的内容必须出现n次，不能多不能少
{n,}	前一项正则的内容必须至少出现n次
{n,m}	前一项正则的内容必须至少出现n次，至多出现m次

当我们需要匹配与前一项相同的字符时，可以使用表单中的正则符号，比如上面的一长串数字，可以这样写:/\d+/，就是在逐行扫描的过程只只要遇到数字，就算匹配成功，然后后面又是数字的话，依然匹配，直到没有出现数字为止（比如一个空格）；

3.25.4 括号

正则表达式中的括号可以有圆括号、方括号、花括号（上面已说明）三个：

方括号

方括号在正则中的作用是用来枚举字符，只要有一个命中，整个方括号就算匹配。它的使用形如这样：/[xxx]/，/xxx-yyy/，或者这样/[^xxx]/，/^xxx-yyy/，其实就两点，第一中间有连字符（-），表示这个区间；第二有帽子符号（^），表示不在枚举中的其他字符（反面）；举例来说，我想找到字符串"abc1234def5678g"中的所有数字，这些数字无非就是0到9，可以枚举，所以我们的正则可以这样写：/[0-9]+/，只要数字就匹配。

圆括号

圆括号在正则表达式中的作用比较多，至少有三点（参考《JS权威指南》）：

1. **定义子表达式**：子表达式就如同嵌入在其中的另一个正则表达式，既然是另一个表达式，那么就可以拥有自己的匹配空间；
2. **定义子匹配模式**：可以在所匹配的字符串中抽出匹配括号中子表达式的子字符串；比如字符串"abc1234def"，我希望找到夹杂在"abc"与"def"之间的数字，这里的正则表达式就可以写成这样：/abc(\d+)def/，这样当我整个正则匹配到"abc1234def"的时候，我会从中抽出括号中的东西，而括号中的内容就是我需要的数字。
3. **定义整个正则表达式中后面部分要匹配的模式**：后面部分要匹配的字符串可能不太确定，但能确定的是与前面匹配的相同，这样就可以使用括号；这个很好理解，比如你要在一篇文章中查找引号中的内容，你可以像这样写正则：/"[^"]+"/，现在问题来了，有的内容可能在单引号中，于是你可能再写一个相同样子的正则比如：/'[^']+'/从而再执行一边查找，这当然没有问题。但如果我们要求你只能写一个呢？这就用到了圆括号。正则就像这样：/(['"])[^'"]+\1/，其中\1就是再次匹配与前面第一个圆括号中内容相同的字符，如果前面找到了单引号，后面就匹配单引号，类同。

#	名称/符号	举例	说明
1	^	/^[1-9]\d/	匹配以非零数字开头的数字
2	$	/$\|/\d[1-9]$/	必须以非零数字结尾的数字，但开头可以有零
3	零宽正向先行断言	/\d(?=￥)/	数字后面必须要有￥符号， 否则匹配失败，匹配内容只有数字。（下同）
4	零宽反向先行断言	/\d(?!=￥)/	数字后面必须不能有￥符号
5	零宽正向后行断言	/(?<=￥)\d/	数字前面必须要有￥符号

表 3.9: 正则表达式中的几种定位语法，零宽正向后行断言在浏览器中不能识别（*JS*引擎），其他语言没有尝试过，不过微软的*MSDN*中也没有对后行断言的说明，感觉是要被废掉的样子。

3.25.5 引用

引用就是利用圆括号的第二、三个作用，就是获取完整正则表达式中的子表达式。这样做的目的要么是获取子表达式，在其他非正则的地方使用，要么是正则本身在后期使用子字符串。举例来说：我有一份10000多页的电子文档，里面有字母、数字、还有其他什么怪异的符号。单就数字而言，有的表示金额，有的表示电话号码等等。倘若有人告诉我一个这篇文档中的数字，我肯定不能确定这这个数字是金额数字还是电话号码。但我能确定的是所有的电话号码前面总有："TEL:"。我如何输出文档中全部的电话号码？

这个问题就需要利用正则中的引用，我们先写能匹配所有这种模式的正则表达式：/TEL:\d+/，然后需要从其中引用到一个子表达式，该子表达式表示电话号码，正则标称这样：/TEL:(\d+)/，这样经过匹配后，所有的电话号码就能得到了，这是使用引用内容的例子。正则后期使用的例子就比如上面的单双引号。

3.25.6 分组

我们用|来进行选择性的匹配，意思就是说：要么你能匹配上，要么我能匹配上，要么大家都能匹配上，无论谁匹配上了都表示命中。比如我希望统计一片文章中的出现的hello world数量，这个hello world可以有好多写法：Hello World、Hello world、hello world、hello World，其中hello与world之间没准是个空格，没准是个制表符，没准是个其他非可视的字符，也没准重复多少次。我就可以这样写正则：/(h|H)ello\s+(w|W)orld/，其中|就是分组，在圆括号中的字母（此时圆括号起到了第一种作用）h与H都行。**这里需要注意的是：分支优先匹配左边的，只要成功，右边不予考虑**。比如：/(sgg|sggg)/，他可以匹配sgg也可以匹配"sggg"，当出现"sgggggggg"的时候，只有其中前3个字符sggg被匹配，因为**分组中左边的有优先权**。

3.25.7 定位

定位算是正则中的高级话题，定位的作用就是在正则查找的时候需要，输出匹配内容的时候不需要，所以才叫定位嘛。最常见的定位就是开头定位与结尾定位，还有其他的比如零宽正向先行断言、零宽正向后行断言等，见表3.9，测试调试地址：

http://tool.oschina.net/regex/

https://regex101.com/

https://www.debuggex.com/

3.26 2014-01-05 使用ECS思想开发的一款格子游戏引擎

此前有过ECS写一段程序的想法，因为我总觉得传统的对象继承对内存浪费是比较大的（当然这有可能是程序设计上的问题），有时候我不得不继承，但继承后却有大部分基类字段是根本用不着的。对内存的这种浪费，就我个人而言是不能接受的。前不久接触到该架构思想，有种相见恨晚的感觉。Demo地址：https://github.com/zspark/tileengine

3.26.1 ECS思想

此前编程涉及的都是传统OOP理念下的实物对象继承游戏，随着游戏技术的不断发展，传统继承机制暴露出了部分缺陷，比如游戏里面有2个类，一个是不能行走、攻击但是生长的BaseTree，另一个是能走能攻击敌人的BaseMonster，现在我的游戏里面需要有这样一种对象：会长枝叶的树形怪物（它能够攻击玩家，同时具备一切树木的基本属性，还能走路），请问你打算继承哪个基类（这里不考虑多重继承）？这就是传统OOP编程至少在游戏编程领域慢慢暴露出的问题，这便导致ECS架构的慢慢出现。

> Entity-component-system (ECS) is an architectural pattern that is mostly used in game development. An ECS follows the Composition over inheritance principle that allows greater flexibility in defining entities —摘自wiki

ECS并不是面向对象的OOP，而是面向数据的DOP。这就需要一定的认知与理解上的转变,它是一种设计思想，并不是一种相对具体的框架，在其思想下，你可以写框架，亦可以写引擎，或者直接写游戏。

Component

ECS中的字母C是**Component**。说起组件可能我们更加倾向于想到按钮、单选框、文本等，这两种“组件”虽然在含义上有出入，但究其在各自框架结构中扮演的角色却是相同的：**他们都是一个更大系统的组成部分**。

设想一个按钮组件，我们可以想到它会有caption/label属性，正常情况下它也一定可以被点击，有一定的外形。但是我们绝不会想到正常情况下的按钮组件会有列表List的功能、播放动画的功能等等。联系到ECS中的组件也是一样的：**一个组件会有一项具体的的功能，但绝对不会有系统全部的功能。**

我们继续分析按钮组件。一个按钮会被点击，从而触发一些应用逻辑，这是说按钮有部分逻辑存在；同样当我们设置按钮的状态为不可点击时，它会一直被渲染成类似灰色的样子是因为其记录了一些数据。这样看来按钮组件就可以被定义为一个具有被点击功能的逻辑与数据的合体。这种对按钮组件的抽象并不适合ECS中的组件定义，结合之前所说ECS是面相数据的，于是更加准确的CES组件定义便是： **一个具体的、性质相对单一的一组数据的集合。**

> Component: the raw data for one aspect of the object, and how it interacts with the world. ”Labels the Entity as possessing this particular aspect”. Implementations typically use Structs, Classes, or Associative Arrays. —摘wiki

了解了**Component**的基本思想后，我们可以将上面树妖的简单数据分成三个组件，A包含了

成长相关的数据；B包含攻击相关的数据；C包含行走相关的数据。一般情况下我们会再创建一个新的结构或者类，用它去组合A与B，如此一来，ECS中的**Entity**便登场了。

Entity

ECS中的字母E‘Entity‘就是“实体”的意思。这个实体是数据实体，并不是对象实体（因为前面说过他是面相数据的，并且它是由各种Component组合而成的）。究竟是如何组合的呢？想想我们的树妖，能走能攻击还能生长，那我们就创建一个树妖实体，让他组合了上面A，B，C三种组件，这不就包含了一切需要的数据嘛，倘若是个普通怪物，仅仅组合B，C即可。普通树的话就只有A了。这就是Entity：**由各种必要Component组合而成的另一个对象。**

> Entity: The entity is a general purpose object. Usually, it only consists of a unique id. They ”tag every coarse gameobject as a separate item”. Implementations typically use a plain integer for this.
> Entity: A container into which components can be added, usually hierarchical (any Entity can have sub-Entities). —摘自wiki

System

System是逻辑的躯体。它们（游戏里面可能有好多System）驱动着添加进来的所有同一类型组件**Component**做出指定的逻辑运算。有的System需要的Component不止一种。比如系统要渲染Bitmap到屏幕，至少需要知道BitmapData的数据（绘制什么东西）、Bitmap的坐标、旋转、缩放，父容器（绘制到哪里）才行，如果我们用PositionComponent记录坐标、旋转与缩放的话，那肯定还需要另外一个记录着BitmapData数据的Compnent。再拿上面的树妖做比，树妖要攻击敌人，其ComponentB组件虽然保存有攻击力、暴击率、暴击伤害、攻击速度等一套数据，但也仅仅是一堆数据而已，他们如何计算？如何依赖?，我们需要将该组件以类似参数的形式传递给攻击系统（姑且叫做SystemAttack），攻击系统经过逻辑运算得出了本次攻击的具体伤害数值。而树妖实体E（Entity）的另外一个组件ComponentA则被传递给了成长系统（SystemGrow）得到了具体的成长值。其他树妖组件同此逻辑；**System是用来执行游戏逻辑，计算组件数据，驱动数据流的进行。**可以看出一个**System**并不是针对一个具体的**Entity**而执行逻辑，而是根据E中更小的**Component**来执行；

> System: ”Each System runs continuously (as though each System had its own private thread) and performs global actions on every Entity that possesses a Component of the same aspect as that System.” —摘自wiki

3.26.2 Extra

有的开发者将一个具体的System用到的所有Component封装到一个被称为Node的全新类里面，方便每次System驱动的时候只需要一种对象Node实例，而不是一堆组件实例；ECS架构还有一个称作SystemManager的类，这个其实就是集中管理所有System实体，整体驱动、停止等；

有了ECS这种将复杂对象的一组相关属性封装成组件存储，而将逻辑统一由System处理的架构思想，上面传统对象继承存在的问题迎刃而解，树怪实体（Entity）可以包含基类树与基类怪的所有Component，外带自己独特的Component，将他们各自推进针对不同Component处理

的System逻辑处理器里面执行。这就是ECS的核心：**尽可能抽象对象，分裂对象，分裂到具体要求的最低级别，然后用各自独立的系统去批处理相同定义的数据块，屏蔽实体对系统的多样化要求，进而简化程序开发复杂度。**

3.26.3 引擎目前不支持的常用功能

斜坡格子与电梯；

3.26.4 经验总结

必须对自己将要设计的系统全面熟悉，组件一定要严格存储自己的数据，不同组件之间不可有交集，且字段定义必须非常明确；系统是所有一类Node的批处理系统，这些Node中的组件可能来自完全不同的实体，这就需要系统尽量抽象，不可简单针对一类实体下的组件属性，更不可能只针对一个;习惯了写传统类结构关系，从而在开发期间，总想着各种对象继承，对象属性什么的，切记一定要强迫自己转变（是吸收、使用新的技术，绝不是抛弃传统的对象继承）；实体中各种组件具体值的初始化，可交由不同的config类完成，你可以根据实体的不同类型使用一个很长的if...else指向具体的config装配方法，这都不是重点。

3.26.5 相关参考资料