1. 高级碰撞检测

不规则图形 比较非透明区 检查碰撞点的透明度

大量对象的互相碰撞检测，越多碰撞越复杂，遍历：

基于网格 把对象分配到格子中 使用二维数组 或者树等结构

每次循环移动时 可以以格为单位移动 也可以平缓移动，动态判断是否出格子边界/进入另一个格子，然后动态分配对象所在格子（最简判断标准是对象中心越过格子边缘）

可以认为进入同一个格子就发生了碰撞 也可以只判断格子中对象数（0，1，2），只对存在多个对象的格子里的对象进行碰撞检测，遍历的是格子

关于格子的索引，可以直接用对象的坐标来转换出来

扩展:使用多叉树 类似上面基于网格 动态分配到格子里 四叉树减少遍历总数，遍历单位是格子，从大格子到小格子 只对拥有多个对象的格子中的对象判断

网格检测不只是为了碰撞，还可以判断距离

1. 转向行为

使用转向力来进行转向显得比较自然

基本行为 ：寻找（固定或者移动） 避开 到达（到目标会减速） 追捕（接近目标会加速） 躲避 漫游 路径 分离（相邻保持距离） 凝聚（尽量不掉队） 队列（和相邻角色同一方向）

向量 方向大小（正负），分解成xy 速度、对象间距离，对象方向

寻找：

转向力

根据目标位置 **每桢**获得一个目标速度（大小方向，直接指向目标）， 和现有速度相减 ，获得**需要被加上的速度**总值，转向力方向和需要被加上的速度方向相同，这个力大小在**需要被加上的速度**达到时 归0

避开：发现目标后， 应该往回走，目标速度是和上面的寻找反向的

避开鼠标

到达：当距离接近的时候 减速

追捕：预测目标未来所在位置 需要一个预测时间 假设目标当前速度不变 预判对方时间后的位置 然后对该位置进行**寻找**

动态预测：目标远就调高预测时间 目标近就少预测一点 当前距离/目标速度

躲避就是追捕的反行为 预测目标未来位置使用避开 反向速度

漫游：布朗运动不平滑自然，太过随机 应该在目标前方设置一个圆，然后取圆内随机位置， 当成目标点来寻找

对象回避 : 预测障碍，绕开 比较复杂，如果太接近可能就撞上了 可能需要超急减速 超大回旋

障碍可以作为圆形， 用和障碍之间的连线方向，距离，两者半径 以及当前速度方向判断是否会相撞 根据距离大小，决定转向的力度 目标速度，速度方向的恢复

可以在接近时减速 在隔得比较远的时候不转向仍然直线运动，绕过对象再恢复原方向

多个需要回避的对象的话 障碍加入到数组遍历

路径跟随，沿路径运动 把路径每个节点当成目标点，进行依次 寻找 使用寻找算法运动会比较自然 （赛车游戏，过弯时如何转比较自然，路径点如何设计，是否要减速）

群落行为： 离队伍远了归队，凝聚 避免太接近，分离 飞行方向大概相同，队列

三个行为一般同时 群落位置，群落速度， 算附近的其他对象的平均值 平均速度，平均位置 以该点为目标速度，目标位置运动

凝聚就是距离太远需要加速，然后距离接近时减速到目标速度

队列就是调整运动方向到目标速度

分离就是发现附近有太近的对象时，转向，远离到一定距离再转回速度方向

1. 等角投影

3d X y z 固定 俯视视角 45 60 tile

z-index

运动和碰撞 此时坐标轴旋转过了 要用正常的xy，然后转过去

1. 寻路

把地图画成方形网格 ，加入不能通行的块 比如迷宫

节点（每个方块都是节点） 代价f g起点到当前点的代价（已走的路径） h当前点到目标点的估计代价 估价函数

待考察表 已经估价

已考察表 取最小代价 对周围节点估价

过程：起点节点加入待考察 找到最小代价的节点设为当前节点 对邻节点（8个。不包括不能通过的）估价，当前节点加入已考察 ，邻节点都加入待考察

估价最简，算直线距离 水平距离1 垂直 斜线距离1.4 g h f=g+h

最佳路径可能有多条

估价方法：曼哈顿 ：算 横竖 之和

直线距离估价

对角线估价：先对角线到目标同行 再走横竖线

父节点

可能会从斜方向穿过墙直角 （因为是八方向） 可以禁止斜向运动，或者做出检查，在墙的直角才禁止斜运动，其他时候可以

自定义的代价（权值）：比如土路，沼泽， 在赛车游戏，这些路径会减速 导致代价增加

1. 摄像头 麦克风

声控游戏 声音参数 级数 活跃级数 根据声音画线

摄像头识别 滤镜 混合 变形缩放 运动 把视频绘制在位图中

反转 像素分析 颜色分析 Bitmapdata 跟踪颜色从而跟随移动

跟踪一般的移动 像素比较

图像边缘检测 周围像素 卷积滤镜

1. 高等物理 ：数值积分

ax vx x是欧拉积分 每桢都可能变化，从而形成积分

但是现实改变不是每桢一次，而是持续进行的，所以结果是不精确的

更精确的是Runge\_Kutta RK2 RK4 中点法（取前后状态的平均值，速度，位置）

时间驱动而不是桢驱动

Verlet积分法 （创建布娃娃

不保存速度只保存位置 当前位置 之前位置 getvx setvx setposition

Verlet 点 线段

点约束

线段 反弹

结构体（三角等

七、3D

八、绘制路径 直线 曲线

缠绕 交叉路径

三角

三角组成3D体

九、shader 调整像素通道

十、补间引擎 缓动函数 tween

缓动方法 easein先慢后快 easeout先快后慢 easeoutin

合并tween 补间序列