平面凸多边形碰撞文档

简介

该文档用于解释本项目中碰撞判断与碰撞响应的实现。该文档需搭配 Github 仓库或 Gitee 仓库使用。

代码位置

该代码分为两部分，即入口和判断响应。其中入口在 Ground 类（声明的位置）里的 CollisionStart 方法，判断响应在 Collision 命名空间（声明的位置）中。同时所有图形的存储在 Ground 类中，所有图形的声明在 framework.h 的 BasicGraphics 命名空间中。

处理过程

每一帧的处理遵循以下过程：

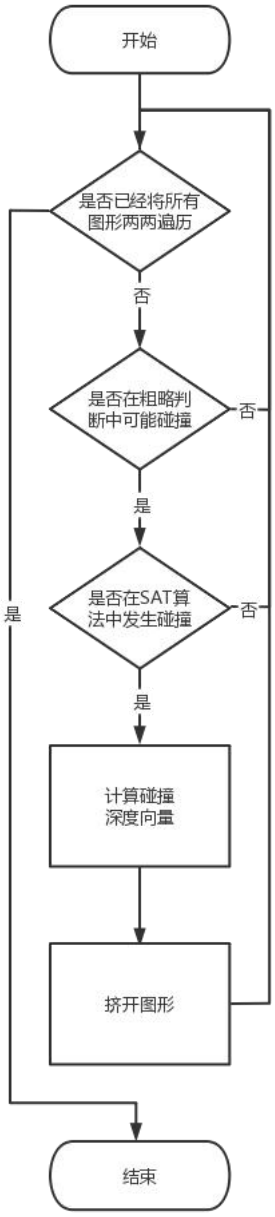
1. 在入口方法中，将所有图形两两组合进入步骤 2；

2. 首先进行粗略判断。若两者之间坐标的 X 或 Y 方向的距离大于最大正方形画布的边长（这里的正方形画布为最小的能覆盖任意形状的正方形，下同），则返回为不可能发生碰撞；否则继续步骤 3；

3. 将两个图形进行对应的 SAT 算法判断是否发生碰撞，如果发生碰撞，则返回碰撞深度向量，否则返回没有发生碰撞；

4. 若以上步骤返回为发生碰撞，则根据碰撞向量和两个图形各自是否固定进行移动，并修改与物理相关的变量；

以下为流程图：



图形格式

图形格式位于 BasicGraphics 命名空间中。以下为各种图形的格式：

矩形（Rectangle）：

```
struct Rectangle
{
    PointF address;
    float left;
    float top;
    float right;
    float bottom;
    ColorF color;
    Physics phy;

    operator D2D_RECT_F();
};
```

变量含义

address

图形的坐标。位于图形的左上角。

left, top, right, bottom

各方向的边界距离图形坐标的距离，正负代表方向。

color

图形的颜色。

phy

图形的物理值。

方法含义

operator D2D_RECT_F()

将保存的矩形转换成用于绘制的矩形。

凸多边形（ConvexPolygon）

```
struct ConvexPolygon
{
    PointF address;
    std::vector<VectorF> vertex;
    ColorF color;
    Physics phy;

    CComPtr<ID2D1PathGeometry> pGeometry = nullptr;
};
```

变量含义

address

图形的坐标，位于左上角。

vertex

图形坐标相对于图形的对应顶点的向量。

color

图形的颜色。

phy

图形的物理。

pGeometry

在使用 Direct2D 进行绘制时所使用的多边形。

圆形（Circle）

```
struct Circle
{
    PointF address;
    float radius;
    ColorF color;
    Physics phy;

    operator D2D1_ELLIPSE();
};
```

变量含义

address

图形的坐标。位于左上角。注意，这不是是圆心，向该坐标加上向量(radius,radius)才是圆心。

radius

圆的半径。

color

图形的颜色。

phy

图形的物理。

方法含义

operator D2D1_ELLIPSE();

将保存的圆形转换为绘制时使用的圆形。

碰撞检测

以下为该项目中用于碰撞判断的函数解释注意，以下函数位说明返回值的函数均返回是否发生碰撞）：

RoughCollision

粗略碰撞判断。注意，该函数声明位于 Ground.cpp。

函数原型

```
bool RoughCollision(const BasicGraphics::PointF& p1, const BasicGraphics::PointF& p2)
```

参数含义

p1
: 第一个图形的坐标。

p2
: 第二个图形的坐标。

返回值

是否可能发生碰撞。

注意

函数内的 MAXBLOCK 常量为任意图形在最大正方形画布边长。

Collision::CircleCircle

圆形与圆形的精确碰撞。

函数原型

```
bool CircleCircle(
    const BasicGraphics::Circle& c1, const BasicGraphics::Circle& c2,
    BasicGraphics::VectorF& depth);
```

参数含义

c1

第一个圆形。

c2

第二个圆形。

depth

碰撞深度向量，输出值。

执行过程

使用圆心距离判断，以下为执行过程：

1. 通过圆形的坐标与半径获得从第一个圆形的圆心指向第二个圆形的圆心的向量；
2. 若该向量的长度大于两个圆形半径之和，则未发生碰撞，返回假；若长度等于 0 则碰撞深度向量指向下、大小为两个圆形的半径之和，返回假；
3. 以上都不成立，则根据相似原理将第一步获得的向量长度缩短到半径之和与该向量长度之差，并通过参数 depth 返回，函数返回真。

Collision::CirclePolygon

圆形与凸多边形的精确碰撞。

函数原型

```
bool CirclePolygon(
    const BasicGraphics::Circle& c1, const BasicGraphics::ConvexPolygon& p2,
    BasicGraphics::VectorF& depth);
```

参数含义

c1

圆形。

p2

凸多边形。

depth

碰撞深度向量，输出值。

执行过程

使用 SAT 算法，以下为执行过程：

1. 先将由多边形中的第一个顶点与最后一个顶点连线的法向量方向上的碰撞深度向量求出，若没有该向量则函数返回假；
2. 再遍历除去该边外的所有多边形的边上的碰撞深度向量，若存在一个边没有该向量则函数返回假，否则将碰撞深度向量更新为最短的碰撞深度向量；
3. 最后找出与圆心距离最近的多边形的顶点，将该顶点与圆心连线的法向量方向上的碰撞深度向量求出，若没有该向量则函数返回假，否则将碰撞深度向量更新为最短的碰撞深度向量；
4. 将最短的碰撞深度向量通过参数 depth 返回并且函数返回真。

Collision::CircleRect

圆形与矩形的精确碰撞。

函数原型

```
bool CircleRect(
    const BasicGraphics::Circle& c1, const BasicGraphics::Rectangle& r2,
    BasicGraphics::VectorF& depth);
```

参数含义

c1

圆形

r2

矩形

depth

碰撞深度向量，输出值。

执行过程

由于当前项目没有旋转，使用 SAT 算法，以下为执行过程：

1. 先分别向平行于矩形的底边与侧边的方向投影，由于存在投影重合的点，所以可以将矩形的顶边与底边进行投影到平行于侧边的坐标轴，同理可以将两条侧边进行投影到平行于底边的坐标轴；
2. 将圆形投影到上步的两条坐标轴上，分别计算碰撞深度向量，保留最短的碰撞深度向量，若存在一个碰撞深度向量不存在，则函数返回假；
3. 将矩形周围的区域划分为下图区域：

TL 区域	T 区域	TR 区域
L 区域	矩形内部	R 区域
BL 区域	B 区域	BR 区域

若圆心在 T、R、B、L 区域中的任意一个区域，则直接进行步骤 4，否则计算圆心与距离矩形最近的顶点连线的法向量上的碰撞深度向量，若没有该向量则函数返回假，否则保留最短的碰撞深度向量；

4. 将最短的碰撞深度向量通过参数 depth 返回并且函数返回真。

Collision::PolygonPolygon

凸多边形与凸多边形的精确碰撞。

函数原型

```
bool PolygonPolygon(const BasicGraphics::ConvexPolygon& p1,
    const BasicGraphics::ConvexPolygon& p2,
    BasicGraphics::VectorF& depth);
```

参数含义

p1

第一个凸多边形。

p2

第二个凸多边形。

depth

碰撞深度向量，输出值。

执行过程

使用 SAT 算法，以下为执行过程：

1. 先分别计算两个多边形中的第一个顶点与最后一个顶点连线的法向量方向上的碰撞深度向量，若有一个不存在则函数返回假，否则保留最短碰撞深度向量；
2. 分别计算两个多边形中除去步骤 1 中的边外的所有边上的碰撞深度向量，若存在一个边没有该向量则函数返回假，否则将碰撞深度向量更新为最短的碰撞深度向量；
3. 将最短的碰撞深度向量通过参数 depth 返回并且函数返回真。

Collision::PolygonRect

凸多边形与矩形的精确碰撞。

函数原型

```
bool PolygonRect(
    const BasicGraphics::ConvexPolygon& p1, const BasicGraphics::Rectangle& r2,
    BasicGraphics::VectorF& depth);
```

参数含义

p1

凸多边形。

r2

矩形

depth

碰撞深度向量，输出值。

执行过程

使用 SAT 算法，以下为执行过程：

1. 先分别向平行于矩形的底边与侧边的方向投影，由于存在投影重合的点，所以可以将矩形的顶边与底边进行投影到平行于侧边的坐标轴，同理可以将两条侧边进行投影到平行于底边的坐标轴；
2. 将凸多边形投影到上步的两条坐标轴上，分别计算碰撞深度向量，保留最短的碰撞深度向量，若存在一个碰撞深度向量不存在，则函数返回假；
3. 再遍历多边形的所有边上的碰撞深度向量，若存在一个边没有该向量则函数返回假，否则将碰撞深度向量更新为最短的碰撞深度向量；
4. 将最短的碰撞深度向量通过参数 depth 返回并且函数返回真。

Collision::RectRect

圆形与圆形的精确碰撞。

函数原型

```
bool RectRect(
    const BasicGraphics::Rectangle& r1, const BasicGraphics::Rectangle& r2,
    BasicGraphics::VectorF& depth);
```

参数含义

r1

第一个矩形。

r2

第二个矩形。

depth

碰撞深度向量，输出值。

执行过程

由于目前没有旋转，使用 AABB 算法，以下为执行过程：

1. 由于没有旋转，所以两个矩形所有边上的法向量均平行于 X 方向或 Y 方向，所以只需要计算 X 正方向与 Y 正方向的碰撞深度向量即可，若存在一个碰撞深度向量不存在，则函数返回假；
2. 将最短的碰撞深度向量通过参数 depth 返回并且函数返回真。

碰撞响应

以下方法用于处理碰撞响应：

Collision::CollisionResponse

将发生碰撞的图形移开并提供速度。

函数原型

```
void CollisionResponse(
    BasicGraphics::PointF& g1Addr, BasicGraphics::Physics& g1Phy,
    BasicGraphics::PointF& g2Addr, BasicGraphics::Physics& g2Phy,
    const BasicGraphics::VectorF depth);
```

参数含义

g1Addr, g1Phy
　　第一个图形的坐标与物理信息。
g1Addr, g1Phy
　　第二个图形的坐标与物理信息。
depth
　　从第一个图形指向第二个图形的碰撞深度向量。

函数解释

1. 对于两个图形均为固定状态时，则直接返回；对于有且只有一个图形为固定状态时，则推开未固定的图形，对于两个图形均未固定时，则各推开碰撞深度的一半。
2. 对于其余代码部分为根据动量定理叠加速度。

注意事项

1. 该项目的世界坐标为窗口坐标，即 X 正方向为右，Y 正方向为下。
2. 所有的碰撞深度向量的方向均为由碰撞检测方法的第一个图形指向第二个图形。
3. 所有精确碰撞判断中均以第一个图形的坐标为原点进行计算，以减小浮点数计算时的精度问题。
4. 在 SAT 算法中，并不需要确定两个点的连线的法向量是否指向另一个图形，只需要在求该法向量上的碰撞深度向量的时候，将所有的点投影到与该法向量同向的一维坐标轴上，把所有点投影到坐标轴的点的坐标按照同一原点的位置赋值，将所求的重叠区域的两个端点按照当前提供法向量的图形所属投影的端点的坐标值减去另一个图形的，将该差值与法向量相乘，即可得到有该图形指向另一个图形的碰撞深度向量。

附录 1：计算碰撞深度向量

以凸多边形与凸多边形碰撞过程中其中一次计算碰撞深度向量为例，并假设存在碰撞深度向量，以下为执行过程：

1. 将两个顶点分为起始点与终止点（可以随意指定）；
2. 计算从起始点到终止点的向量的垂直向量，并缩放为法向量；
3. 将所有的点投影到该法向量上，并以某一点的投影点为原点获得所有投影的坐标；
4. 将两个图形的各自的最大投影坐标与最小投影坐标进行判断，获得重叠部分的向量，正方向为从第一个凸多边形的投影点指向第二个凸多边形的投影点；
5. 将该向量的值与法向量相乘得到碰撞深度向量。

参考

GitHub 仓库：twobitcoder101/FlatPhysics。
知乎：2D 凸多边形碰撞检测算法（一） - SAT