# 球

| | |
|---|---|
| **原文** | **Ball (https://learnopengl.com/#!In-Practice/2D-Game/Collisions/Ball)** |
| 作者 | JoeyDeVries |
| 翻译 | aillieo (https://github.com/aillieo) |
| 校对 | 暂未校对 |

> **Note**
>
> 本节暂未进行完全的重写，错误可能会很多。如果可能的话，请对照原文进行阅读。如果有报告本节的错误，将会延迟至重写之后进行处理。

此时我们已经有了一个包含有很多砖块和玩家的一个挡板的关卡。与经典的Breakout内容相比还差一个球。游戏的目的是让球撞击所有的砖块，直到所有的可销毁砖块都被销毁，但同时也要满足条件：球不能碰触屏幕的下边缘。

除了通用的游戏对象组件，球还需要有半径和一个布尔值，该布尔值用于指示球被固定(stuck)在玩家挡板上还是被允许自由运动的状态。当游戏开始时，球被初始固定在玩家挡板上，直到玩家按下任意键开始游戏。

由于球只是一个附带了一些额外属性的`GameObject`，所以按照常规需要创建`BallObject`类作为`GameObject`的子类。

```
class BallObject : public GameObject
{
    public:
        // 球的状态
        GLfloat   Radius;
        GLboolean Stuck;


        BallObject();
        BallObject(glm::vec2 pos, GLfloat radius, glm::vec2 velocity, Texture2D sprit
e);

        glm::vec2 Move(GLfloat dt, GLuint window_width);
        void      Reset(glm::vec2 position, glm::vec2 velocity);
};
```

`BallObject`的构造函数不但初始化了其自身的值，而且实际上也潜在地初始化了`GameObject`。`BallObject`类拥有一个`Move`函数，该函数用于根据球的速度来移动球，并检查它是否碰到了场景的任何边界，如果碰到的话就会反转球的速度：

```
glm::vec2 BallObject::Move(GLfloat dt, GLuint window_width)
{
    // 如果没有被固定在挡板上
    if (!this->Stuck)
    {
        // 移动球
        this->Position += this->Velocity * dt;
        // 检查是否在窗口边界以外，如果是的话反转速度并恢复到正确的位置
        if (this->Position.x <= 0.0f)
        {
            this->Velocity.x = -this->Velocity.x;
            this->Position.x = 0.0f;
        }
        else if (this->Position.x + this->Size.x >= window_width)
        {
            this->Velocity.x = -this->Velocity.x;
            this->Position.x = window_width - this->Size.x;
        }
        if (this->Position.y <= 0.0f)
        {
            this->Velocity.y = -this->Velocity.y;
            this->Position.y = 0.0f;
        }

    }
    return this->Position;
}
```

除了反转球的速度之外，我们还需要把球沿着边界重新放置回来。只有在没有被固定时球才能够移动。

> 因为如果球碰触到底部边界时玩家会结束游戏（或失去一条命），所以在底部边界没有代码来控制球反弹。我们稍后需要在代码中某些地方实现这一逻辑。

你可以在下边看到`BallObject`的代码：

- BallObject: header (https://learnopengl.com/code_viewer.php?code=in-practice/breakout/ball_object_collisions.h), code (https://learnopengl.com/code_viewer.php?code=in-practice/breakout/ball_object_collisions)

首先我们在游戏中添加球。与玩家挡板相似，我们创建一个球对象并且定义两个用来初始化球的常量。对于球的纹理，我们会使用在LearnOpenGL Breakout游戏中完美适用的一张图片：球纹理(../../../../../img/06/Breakout/05/01/awesomeface.png)。

```
// 初始化球的速度
const glm::vec2 INITIAL_BALL_VELOCITY(100.0f, -350.0f);
// 球的半径
const GLfloat BALL_RADIUS = 12.5f;

BallObject     *Ball;

void Game::Init()
{
    [...]
    glm::vec2 ballPos = playerPos + glm::vec2(PLAYER_SIZE.x / 2 - BALL_RADIUS, -BALL_
RADIUS * 2);
    Ball = new BallObject(ballPos, BALL_RADIUS, INITIAL_BALL_VELOCITY,
        ResourceManager::GetTexture("face"));
}
```

然后我们在每帧中调用游戏代码中`Update`函数里的`Move`函数来更新球的位置：

```
void Game::Update(GLfloat dt)
{
    Ball->Move(dt, this->Width);
}
```

除此之外，由于球初始是固定在挡板上的，我们必须让玩家能够从固定的位置重新移动它。我们选择使用空格键来从挡板释放球。这意味着我们必须稍微修改`ProcessInput`函数：

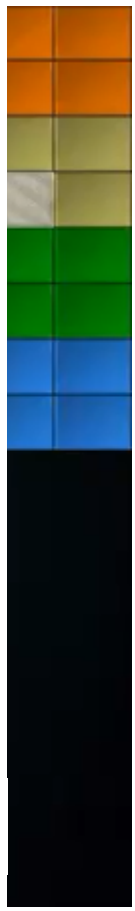
```
void Game::ProcessInput(GLfloat dt)
{
    if (this->State == GAME_ACTIVE)
    {
        GLfloat velocity = PLAYER_VELOCITY * dt;
        // 移动玩家挡板
        if (this->Keys[GLFW_KEY_A])
        {
            if (Player->Position.x >= 0)
            {
                Player->Position.x -= velocity;
                if (Ball->Stuck)
                    Ball->Position.x -= velocity;
            }
        }
        if (this->Keys[GLFW_KEY_D])
        {
            if (Player->Position.x <= this->Width - Player->Size.x)
            {
                Player->Position.x += velocity;
                if (Ball->Stuck)
                    Ball->Position.x += velocity;
            }
        }
        if (this->Keys[GLFW_KEY_SPACE])
            Ball->Stuck = false;
    }
}
```

现在如果玩家按下了空格键，球的`Stuck`值会设置为`false`。我们还需要更新`ProcessInput`函数，当球被固定的时候，会跟随挡板的位置来移动球。

最后我们需要渲染球，此时这应该很显而易见了：

```
void Game::Render()
{
    if (this->State == GAME_ACTIVE)
    {
        [...]
        Ball->Draw(*Renderer);
    }
}
```

结果就是球会跟随着挡板，并且当我们按下空格键时球开始自由运动。球会在左侧、右侧和顶部边界合理地反弹，但看起来不会撞击任何的砖块，就像我们可以在下边的视频中看到的那样：



0:00 / 0:05

我们要做的就是创建一个或多个函数用于检查球对象是否撞击关卡中的任何砖块，如果撞击的话就销毁砖块。这些所谓的碰撞检测(collision detection)功能将是我们下一个 (../02 Collision detection/)教程的重点。