道具

原文	**Powerups (https://learnopengl.com/#!In-Practice/2D-Game/Powerups)**
作者	JoeydeVries
翻译	包纸 (https://github.com/ShirokoSama)
校对	暂无

> **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。

Breakout已经接近完成了，但我们可以至少再增加一种游戏机制让它变得更酷。“充电”（译注：Powerups，很多游戏中都会用这个单词指代可以提升能力的道具，本文之后也会用道具一词作为其翻译）怎么样？

这个想法的含义是，无论一个砖块何时被摧毁，它都有一定几率产生一个道具块。这样的道具快会缓慢降落，而且当它与玩家挡板发生接触时，会发生基于道具类型的有趣效果。例如，某一种道具可以让玩家挡板变长，另一种道具则可以让小球穿过物体。我们还可以添加一些可以给玩家造成负面影响的负面道具。

我们可以将道具建模为具有一些额外属性的`GameObject`，这也是为什么我们定义一个继承自`GameObject`的`PowerUp`类并在其中增添了一些额外的成员属性。

```
const glm::vec2 SIZE(60, 20);
const glm::vec2 VELOCITY(0.0f, 150.0f);

class PowerUp : public GameObject
{
public:
    // 道具类型
    std::string Type;
    GLfloat     Duration;
    GLboolean   Activated;
    // 构造函数
    PowerUp(std::string type, glm::vec3 color, GLfloat duration,
            glm::vec2 position, Texture2D texture)
        : GameObject(position, SIZE, texture, color, VELOCITY),
          Type(type), Duration(duration), Activated()
    { }
};
```

`PowerUp`类仅仅是一个有一些额外状态的`GameObject`， 所以我们简单地将它定义为一个头文件，你可以在这里 (https://learnopengl.com/code_viewer.php?code=in-practice/breakout/power_up.h)找到它。

每个道具以字符串的形式定义它的类型，持有表示它有效时长的持续时间与表示当前是否被激活的属性。在Breakout中，我们将添加4种增益道具与2种负面道具：

- **Speed**: 增加小球20%的速度
- **Sticky**: 当小球与玩家挡板接触时，小球会保持粘在挡板上的状态直到再次按下空格键，这可以让玩家在释放小球前找到更合适的位置
- **Pass-Through**: 非实心砖块的碰撞处理被禁用，使小球可以穿过并摧毁多个砖块
- **Pad-Size-Increase**: 增加玩家挡板50像素的宽度
- **Confuse**: 短时间内激活confuse后期特效，迷惑玩家
- **Chaos**: 短时间内激活chaos后期特效，使玩家迷失方向

你可以在下面找到道具的高质量纹理：

- **Texture**： Speed (https://learnopengl.com/img/in-practice/breakout/textures/powerup_speed.png),Sticky (https://learnopengl.com/img/in-practice/breakout/textures/powerup_sticky.png),Pass-Through

(https://learnopengl.com/img/in-practice/breakout/textures/powerup_passthrough.png),Pad-Size-Increase (https://learnopengl.com/img/in-practice/breakout/textures/powerup_increase.png),Confuse (https://learnopengl.com/img/in-practice/breakout/textures/powerup_confuse.png),Chaos (https://learnopengl.com/img/in-practice/breakout/textures/powerup_chaos.png).

与关卡中的砖块纹理类似，每个道具纹理都是完全灰度的，这使得我们在将其与颜色向量相乘时可以保持色彩的平衡。

因为我们需要跟踪游戏中被激活的道具的类型、持续时间、相关效果等状态，所以我们将它们存储在一个容器内：

```
class Game {
    public:
        [...]
        std::vector<PowerUp>  PowerUps;
        [...]
        void SpawnPowerUps(GameObject &block);
        void UpdatePowerUps(GLfloat dt);
};
```

我们还定义了两个管理道具的函数，`SpawnPowerUps`在给定的砖块位置生成一个道具，`UpdatePowerUps`管理所有当前被激活的道具。

SpawnPowerUps

每次砖块被摧毁时我们希望以一定几率生成一个道具，这个功能可以在`Game`的`SpawnPowerUps`函数中找到：

```
GLboolean ShouldSpawn(GLuint chance)
{
    GLuint random = rand() % chance;
    return random == 0;
}
void Game::SpawnPowerUps(GameObject &block)
{
    if (ShouldSpawn(75)) // 1/75的几率
        this->PowerUps.push_back(
             PowerUp("speed", glm::vec3(0.5f, 0.5f, 1.0f), 0.0f, block.Position, tex_
speed
         ));
    if (ShouldSpawn(75))
        this->PowerUps.push_back(
            PowerUp("sticky", glm::vec3(1.0f, 0.5f, 1.0f), 20.0f, block.Position, tex
_sticky
        );
    if (ShouldSpawn(75))
        this->PowerUps.push_back(
            PowerUp("pass-through", glm::vec3(0.5f, 1.0f, 0.5f), 10.0f, block.Positio
n, tex_pass
        ));
    if (ShouldSpawn(75))
        this->PowerUps.push_back(
            PowerUp("pad-size-increase", glm::vec3(1.0f, 0.6f, 0.4), 0.0f, block.Posi
tion, tex_size
        ));
    if (ShouldSpawn(15)) // 负面道具被更频繁地生成
        this->PowerUps.push_back(
            PowerUp("confuse", glm::vec3(1.0f, 0.3f, 0.3f), 15.0f, block.Position, te
x_confuse
        ));
    if (ShouldSpawn(15))
        this->PowerUps.push_back(
            PowerUp("chaos", glm::vec3(0.9f, 0.25f, 0.25f), 15.0f, block.Position, te
x_chaos
        ));
}
```

这样的`SpawnPowerUps`函数以一定几率（1/75普通道具，1/15负面道具）生成一个新的`PowerUp`对象，并设置其属性。每种道具有特殊的颜色使它们更具有辨识度，同时根据类型决定其持续时间的秒数，若值为0.0f则表示它持续无限长的时间。除此之外，每个道具初始化时传入被摧毁砖块的位置与上一小节给出的对应纹理。

激活道具

接下来我们更新游戏的`DoCollisions`函数使它不只检查小球与砖块和挡板的碰撞，还检查挡板与所有未被销毁的道具的碰撞。注意我们在砖块被摧毁的同时调用`SpawnPowerUps`函数。

```
void Game::DoCollisions()
{
    for (GameObject &box : this->Levels[this->Level].Bricks)
    {
        if (!box.Destroyed)
        {
            Collision collision = CheckCollision(*Ball, box);
            if (std::get<0>(collision))
            {
                if (!box.IsSolid)
                {
                    box.Destroyed = GL_TRUE;
                    this->SpawnPowerUps(box);
                }
                [...]
            }
        }
    }
    [...]
    for (PowerUp &powerUp : this->PowerUps)
    {
        if (!powerUp.Destroyed)
        {
            if (powerUp.Position.y >= this->Height)
                powerUp.Destroyed = GL_TRUE;
            if (CheckCollision(*Player, powerUp))
            {   // 道具与挡板接触，激活它！
                ActivatePowerUp(powerUp);
                powerUp.Destroyed = GL_TRUE;
                powerUp.Activated = GL_TRUE;
            }
        }
    }
}
```

对所有未被销毁的道具，我们检查它是否接触到了屏幕底部或玩家挡板，无论哪种情况我们都销毁它，但当道具与玩家挡板接触时，激活这个道具。

激活道具的操作可以通过将其`Activated`属性设为true来完成，实现其效果则需要将它传给`ActivatePowerUp`函数：

```
void ActivatePowerUp(PowerUp &powerUp)
{
    // 根据道具类型发动道具
    if (powerUp.Type == "speed")
    {
        Ball->Velocity *= 1.2;
    }
    else if (powerUp.Type == "sticky")
    {
        Ball->Sticky = GL_TRUE;
        Player->Color = glm::vec3(1.0f, 0.5f, 1.0f);
    }
    else if (powerUp.Type == "pass-through")
    {
        Ball->PassThrough = GL_TRUE;
        Ball->Color = glm::vec3(1.0f, 0.5f, 0.5f);
    }
    else if (powerUp.Type == "pad-size-increase")
    {
        Player->Size.x += 50;
    }
    else if (powerUp.Type == "confuse")
    {
        if (!Effects->Chaos)
            Effects->Confuse = GL_TRUE; // 只在chaos未激活时生效，chaos同理
    }
    else if (powerUp.Type == "chaos")
    {
        if (!Effects->Confuse)
            Effects->Chaos = GL_TRUE;
    }
}
```

`ActivatePowerUp`的目的正如其名称，它按本章教程之前所预设的那样激活了一个道具的效果。我们检查道具的类型并相应地改变游戏状态。对于**Sticky**和**Pass-through**效果，我们也相应地改变了挡板和小球的颜色来给玩家一些当前被激活了哪种效果的反馈。

因为**Sticky**和**Pass-through**效果稍微改变了一些原有的游戏逻辑，所以我们将这些效果作为属性存储在小球对象中，这样我们可以根据小球当前激活了什么效果而改变游戏逻辑。我们只在`BallObject`的头文件中增加了两个属性，但为了完整性下面给出了更新后的代码：

- **GameObject**: 头文件 (https://learnopengl.com/code_viewer.php?code=in-practice/breakout/ball_object.h)，代码 (https://learnopengl.com/code_viewer.php?code=in-practice/breakout/ball_object)

这样我们可以通过改动`DoCollisions`函数中小球与挡板碰撞的代码便捷地实现**Sticky**效果。

```
if (!Ball->Stuck && std::get<0>(result))
{
    [...]
    Ball->Stuck = Ball->Sticky;
}
```

在这里我们将小球的Stuck属性设置为它自己的Sticky属性，若**Stikcy**效果被激活，那么小球则会在与挡板接触时粘在上面，玩家不得不再次按下空格键才能释放它。

在同样的`DoCollisions`函数中还有个为了实现**Pass-through**效果的类似小改动。当小球的`PassThrough`属性被设置为true时，我们不对非实习砖块做碰撞处理操作。

```
Direction dir = std::get<1>(collision);
glm::vec2 diff_vector = std::get<2>(collision);
if (!(Ball->PassThrough && !box.IsSolid))
{
    if (dir == LEFT || dir == RIGHT) // 水平碰撞
    {
        [...]
    }
    else
    {
        [...]
    }
}
```

其他效果可以通过简单的更改游戏的状态来实现，如小球的速度、挡板的尺寸、`PostProcessor`对象的效果。

更新道具

现在剩下要做的就是保证道具生成后可以移动，并且在它们的持续时间用尽后失效，否则道具将永远保持激活状态。

在游戏的`UpdatePowerUps`函数中，我们根据道具的速度移动它，并减少已激活道具的持续时间，每当时间减少至小于0时，我们令其失效，并恢复相关变量的状态。

```
void Game::UpdatePowerUps(GLfloat dt)
{
    for (PowerUp &powerUp : this->PowerUps)
    {
        powerUp.Position += powerUp.Velocity * dt;
        if (powerUp.Activated)
        {
            powerUp.Duration -= dt;

            if (powerUp.Duration <= 0.0f)
            {
                // 之后会将这个道具移除
                powerUp.Activated = GL_FALSE;
                // 停用效果
                if (powerUp.Type == "sticky")
                {
                    if (!isOtherPowerUpActive(this->PowerUps, "sticky"))
                    {   // 仅当没有其他sticky效果处于激活状态时重置，以下同理
                        Ball->Sticky = GL_FALSE;
                        Player->Color = glm::vec3(1.0f);
                    }
                }
                else if (powerUp.Type == "pass-through")
                {
                    if (!isOtherPowerUpActive(this->PowerUps, "pass-through"))
                    {
                        Ball->PassThrough = GL_FALSE;
                        Ball->Color = glm::vec3(1.0f);
                    }
                }
                else if (powerUp.Type == "confuse")
                {
                    if (!isOtherPowerUpActive(this->PowerUps, "confuse"))
                    {
                        Effects->Confuse = GL_FALSE;
                    }
                }
                else if (powerUp.Type == "chaos")
                {
                    if (!isOtherPowerUpActive(this->PowerUps, "chaos"))
                    {
                        Effects->Chaos = GL_FALSE;
                    }
                }
            }
        }
    }
```

```
    this->PowerUps.erase(std::remove_if(this->PowerUps.begin(), this->PowerUps.end(),
        [](const PowerUp &powerUp) { return powerUp.Destroyed && !powerUp.Activated;
 }
    ), this->PowerUps.end());
}
```

你可以看到对于每个效果，我们通过将相关元素重置来停用它。我们还将`PowerUp`的`Activated`属性设为false，在`UpdatePowerUps`结束时，我们通过循环PowerUps容器，若一个道具被销毁切被停用，则移除它。我们在算法开头使用`remove_if`函数，通过给定的lamda表达式消除这些对象。

> `remove_if`函数将lamda表达式为true的元素移动至容器的末尾并返回一个迭代器指向应被移除的元素范围的开始部分。容器的`erase`函数接着擦除这个迭代器指向的元素与容器末尾元素之间的所有元素。

可能会发生这样的情况：当一个道具在激活状态时，另一个道具与挡板发生了接触。在这种情况下我们有超过1个在当前`PowerUps`容器中处于激活状态的道具。然后，当这些道具中的一个被停用时，我们不应使其效果失效因为另一个相同类型的道具仍处于激活状态。出于这个原因，我们使用`isOtherPowerUpActive`检查是否有同类道具处于激活状态。只有当它返回false时，我们才停用这个道具的效果。这样，给定类型的道具的持续时间就可以延长至最近一次被激活后的持续时间。

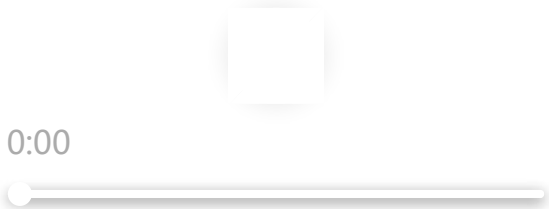
```
GLboolean IsOtherPowerUpActive(std::vector<PowerUp> &powerUps, std::string type)
{
    for (const PowerUp &powerUp : powerUps)
    {
        if (powerUp.Activated)
            if (powerUp.Type == type)
                return GL_TRUE;
    }
    return GL_FALSE;
}
```

这个函数简单地检查是否有同类道具处于激活状态，如果有则返回`GL_TRUE`。

最后剩下的一件事便是渲染道具：

```
void Game::Render()
{
    if (this->State == GAME_ACTIVE)
    {
        [...]
        for (PowerUp &powerUp : this->PowerUps)
            if (!powerUp.Destroyed)
                powerUp.Draw(*Renderer);
        [...]
    }
}
```

结合所有的这些功能，我们有了一个可以运作的道具系统，它不仅使游戏更有趣，还使游戏更具有挑战性。它看上去会像这样：



你可以在下面找到所有更新后的代码（当关卡重置时我们同时重置所有道具效果）：

- **Game**: 头文件 (https://learnopengl.com/code_viewer.php?code=in-practice/breakout/game_powerups.h)，代码 (https://learnopengl.com/code_viewer.php?code=in-practice/breakout/game_powerups)