南华大学

计算机学院

实 验 报 告

（ 2024 ~2025学年度 第 1 学期 ）

**课程名称 Web 全栈开发**

**姓 名 徐慧婷**

**学 号 20224490127**

**专 业 医学信息工程**

# 一、实验目的

本次实验的目的是：

* 学习并掌握Phaser框架的基本用法，用于开发2D网页游戏。
* 理解并实践游戏开发中的关键概念，包括游戏循环、渲染、物理和碰撞检测。
* 掌握如何使用JavaScript进行游戏逻辑编程，实现交互功能。
* 体验从零开始构建一个完整游戏的过程，包括游戏设计、开发、测试和优化。

# 二、实验环境

列出本次实验使用的工具和环境。

* 操作系统：Windows
* 开发工具：VS Code
* 浏览器：Chrome
* 编程语言和框架：HTML、CSS、JavaScript

# 三、实验步骤

1. 准备工作：

* 确保开发环境已安装Node.js和npm。
* 使用npm安装Phaser框架。

1. 项目结构搭建：

* 创建项目文件夹，并初始化项目结构。
* 创建HTML文件作为游戏的入口。
* 创建JavaScript文件来编写游戏逻辑。

1. 游戏初始化：

* 在HTML文件中引入Phaser库。
* 在JavaScript文件中创建Phaser游戏实例。

1. 加载资源：

* 编写代码加载游戏所需的图片、音频等资源。

1. 创建游戏场景：

* 定义游戏的不同状态，如预加载、主游戏、游戏结束等。
* 在预加载状态中，设置加载进度条。
* 在主游戏状态中，创建游戏世界，包括球、挡板和砖块。

1. 实现游戏逻辑：

* 编写代码控制球的运动和反弹。
* 实现挡板的移动，响应用户输入。
* 设置碰撞检测，处理球与挡板、球与砖块的碰撞。

1. 游戏交互：

* 添加得分系统和生命值。
* 实现游戏结束条件，如球触底或砖块全部消除。

1. 优化和调试：

* 调整游戏参数，如球的速度、挡板的灵敏度等。
* 测试游戏，修复可能出现的bug。

1. 游戏发布：

* 将游戏部署到Web服务器或GitHub Pages上。
* 进行最终测试，确保游戏在不同设备和浏览器上运行正常。

# 四、代码示例

<!DOCTYPE html><html>

<head>

<!-- 设置文档的字符编码为UTF-8，确保正确显示各种字符 -->

<meta charset="utf-8" />

<!-- 设置网页的标题为 "Gamedev Phaser Workshop - lesson 16: Randomizing gameplay" -->

<title>Gamedev Phaser Workshop - lesson 16: Randomizing gameplay</title>

<!-- 全局样式设置，将所有元素的内边距和外边距都设置为0 -->

<style>

\* {

padding: 0;

margin: 0;

}

</style>

<!-- 引入Phaser游戏引擎的最小化版本脚本文件 -->

<script src="phaser.min.js"></script></head>

<body>

<script>

// 创建一个新的Phaser游戏实例

// 参数依次为：游戏宽度（480像素）、游戏高度（320像素）、渲染模式（Phaser.AUTO会自动选择合适的渲染方式）、

// 父容器（这里为null，表示没有特定父容器）、游戏状态对象（包含预加载、创建、更新等函数）

var game = new Phaser.Game(480, 320, Phaser.AUTO, null, {

preload: preload,

create: create,

update: update

});

// 定义游戏中的各种变量

// 代表游戏中的球对象

var ball;

// 代表游戏中的挡板对象

var paddle;

// 代表游戏中的砖块组对象

var bricks;

// 用于临时存储新创建的砖块对象

var newBrick;

// 存储砖块的相关信息，如宽度、高度、行列数量等

var brickInfo;

// 用于显示得分的文本对象

var scoreText;

// 存储玩家的得分

var score = 0;

// 存储玩家剩余的生命数

var lives = 3;

// 用于显示剩余生命数的文本对象

var livesText;

// 当玩家失去一条生命时显示的文本对象

var lifeLostText;

// 表示游戏是否正在进行的布尔值

var playing = false;

// 游戏开始按钮对象

var startButton;

// 预加载函数，在游戏开始前加载所需的资源

function preload() {

// 设置游戏的缩放模式为SHOW\_ALL，会根据屏幕大小自动缩放游戏画面以显示全部内容

game.scale.scaleMode = Phaser.ScaleManager.SHOW\_ALL;

// 水平方向上自动对齐页面

game.scale.pageAlignHorizontally = true;

// 垂直方向上自动对齐页面

game.scale.pageAlignVertically = true;

// 设置游戏舞台的背景颜色为浅灰色（#eee）

game.stage.backgroundColor = '#eee';

// 加载挡板的图片资源，文件名为paddle.png

game.load.image('paddle', 'paddle.png');

// 加载砖块的图片资源，文件名为brick.png

game.load.image('brick', 'brick.png');

// 加载球的精灵图资源，文件名为wobble.png，每个帧的宽度为20像素，高度为20像素

game.load.spritesheet('ball', 'wobble.png', 20, 20);

// 加载按钮的精灵图资源，文件名为button.png，每个帧的宽度为120像素，高度为40像素

game.load.spritesheet('button', 'button.png', 120, 40);

}

// 创建函数，用于创建游戏中的各种对象和设置初始状态

function create() {

// 启动Phaser的物理引擎系统，这里使用的是ARCADE物理引擎

game.physics.startSystem(Phaser.Physics.ARCADE);

// 设置在物理碰撞检测中不检查向下的碰撞（可能根据游戏具体需求而定）

game.physics.arcade.checkCollision.down = false;

// 创建球对象并设置其位置在游戏世界宽度的一半、距离底部25像素处，使用的图片资源为'ball'

ball = game.add.sprite(game.world.width \* 0.5, game.world.height - 25, 'ball');

// 为球对象添加一个名为'wobble'的动画，帧序列为[0, 1, 0, 2, 0, 1, 0, 2, 0]，帧率为24帧/秒

ball.animations.add('wobble', [0, 1, 0, 2, 0, 1, 0, 2, 0], 24);

// 设置球对象的锚点为其中心（0.5表示中心位置）

ball.anchor.set(0.5);

// 启用球对象的物理属性，使用ARCADE物理引擎

game.physics.enable(ball, Phaser.Physics.ARCADE);

// 设置球碰到游戏世界边界时会反弹

ball.body.collideWorldBounds = true;

// 设置球的反弹系数为1，即完全弹性碰撞

ball.body.bounce.set(1);

// 设置检查球是否超出游戏世界边界

ball.checkWorldBounds = true;

// 当球超出游戏世界边界时，调用ballLeaveScreen函数

ball.events.onOutOfBounds.add(ballLeaveScreen, this);

// 创建挡板对象并设置其位置在游戏世界宽度的一半、距离底部5像素处，使用的图片资源为'paddle'

paddle = game.add.sprite(game.world.width \* 0.5, game.world.height - 5, 'paddle');

// 设置挡板对象的锚点为中心底部（0.5表示宽度方向中心，1表示高度方向底部）

paddle.anchor.set(0.5, 1);

// 启用挡板对象的物理属性，使用ARCADE物理引擎

game.physics.enable(paddle, Phaser.Physics.ARCADE);

// 设置挡板为不可移动的物体

paddle.body.immovable = true;

// 初始化砖块

initBricks();

// 设置文本的样式，字体为18px的Arial，颜色为蓝色（#0095DD）

textStyle = {

font: '18px Arial',

fill: '#0095DD'

};

// 创建用于显示得分的文本对象，初始文本为'Points: 0'，位置在左上角（5, 5）像素处

scoreText = game.add.text(5, 5, 'Points: 0', textStyle);

// 创建用于显示剩余生命数的文本对象，初始文本为'Lives: '加上当前生命数，位置在右上角（游戏世界宽度 - 5, 5）像素处

livesText = game.add.text(game.world.width - 5, 5, 'Lives: ' + lives, textStyle);

// 设置剩余生命数文本对象的锚点为右上角（1表示宽度方向最右边，0表示高度方向顶部）

livesText.anchor.set(1, 0);

// 创建当玩家失去一条生命时显示的文本对象，初始位置在游戏世界中心，文本为'Life lost, tap to continue'

lifeLostText = game.add.text(game.world.width \* 0.5, game.world.height \* 0.5, 'Life lost, tap to continue', textStyle);

// 设置失去生命提示文本对象的锚点为中心（0.5表示中心位置）

lifeLostText.anchor.set(0.5);

// 初始时隐藏失去生命提示文本对象

lifeLostText.visible = false;

// 创建游戏开始按钮对象，位置在游戏世界中心，使用的图片资源为'button'，点击按钮时调用startGame函数，

// 按钮的不同状态对应的帧索引分别为1、0、2（可能对应不同的按钮外观，如正常、按下、禁用等状态）

startButton = game.add.button(game.world.width \* 0.5, game.world.height \* 0.5, 'button', startGame, this, 1, 0, 2);

// 设置开始按钮的锚点为中心（0.5表示中心位置）

startButton.anchor.set(0.5);

}

// 更新函数，在每一帧都会被调用，用于处理游戏中的逻辑更新

function update() {

// 检测球和挡板之间的碰撞，当发生碰撞时调用ballHitPaddle函数

game.physics.arcade.collide(ball, paddle, ballHitPaddle);

// 检测球和砖块之间的碰撞，当发生碰撞时调用ballHitBrick函数

game.physics.arcade.collide(ball, bricks, ballHitBrick);

// 如果游戏正在进行

if (playing) {

// 设置挡板的x坐标为当前鼠标的x坐标，如果鼠标不在游戏区域内，则设置为游戏世界宽度的一半

paddle.x = game.input.x || game.world.width \* 0.5;

}

}

// 初始化砖块的函数

function initBricks() {

// 定义砖块的相关信息对象

brickInfo = {

// 砖块的宽度为50像素

width: 50,

// 砖块的高度为20像素

height: 20,

// 存储砖块的行列数量信息

count: {

row: 7,

col: 3

},

// 存储砖块在游戏世界中的偏移量信息

offset: {

top: 50,

left: 60

},

// 砖块之间的间距为10像素

padding: 10

};

// 创建一个砖块组对象

bricks = game.add.group();

// 遍历列数

for (c = 0; c < brickInfo.count.col; c++) {

// 遍历行数

for (r = 0; r < brickInfo.count.row; r++) {

// 计算当前砖块的x坐标，根据行列信息、砖块宽度、间距和偏移量来确定

var brickX = (r \* (brickInfo.width + brickInfo.padding)) + brickInfo.offset.left;

// 计算当前砖块的y坐标，根据行列信息、砖块高度、间距和偏移量来确定

var brickY = (c \* (brickInfo.height + brickInfo.padding)) + brickInfo.offset.top;

// 创建一个新的砖块对象，设置其位置和使用的图片资源为'brick'

newBrick = game.add.sprite(brickX, brickY, 'brick');

// 启用新创建砖块的物理属性，使用ARCADE物理引擎

game.physics.enable(newBrick, Phaser.Physics.ARCADE);

// 设置新创建的砖块为不可移动的物体

newBrick.body.immovable = true;

// 设置新创建砖块的锚点为中心（0.5表示中心位置）

newBrick.anchor.set(0.5);

// 将新创建的砖块添加到砖块组中

bricks.add(newBrick);

}

}

}

// 当球击中砖块时调用的函数

function ballHitBrick(ball, brick) {

// 创建一个用于缩放砖块的补间动画对象

var killTween = game.add.tween(brick.scale);

// 设置补间动画的目标缩放值为（0, 0），即缩放到消失，持续时间为200毫秒，使用线性缓动效果

killTween.to({

x: 0,

y: 0

}, 200, Phaser.Easing.Linear.None);

// 当补间动画完成时，调用一次内部的匿名函数，在匿名函数中销毁当前被击中的砖块

killTween.onComplete.addOnce(function () {

brick.kill();

}, this);

// 启动补间动画

killTween.start();

// 玩家得分增加10分

score += 10;

// 更新显示得分的文本内容

scoreText.setText('Points: ' + score);

// 如果玩家的得分达到了所有砖块的总分（行列数相乘再乘以每个砖块的分值10）

if (score === brickInfo.count.row \* brickInfo.count.col \* 10) {

// 弹出提示框告知玩家获胜，并重新加载页面

alert('You won the game, congratulations!');

location.reload();

}

}

// 当球离开游戏屏幕时调用的函数

function ballLeaveScreen() {

// 玩家剩余生命数减1

lives--;

// 如果玩家还有剩余生命

if (lives) {

// 更新显示剩余生命数的文本内容

livesText.setText('Lives: ' + lives);

// 显示失去生命提示文本对象

lifeLostText.visible = true;

// 将球重置到游戏世界宽度的一半、距离底部25像素处

ball.reset(game.world.width \* 0.5, game.world.height - 25);

// 将挡板重置到游戏世界宽度的一半、距离底部5像素处

paddle.reset(game.world.width \* 0.5, game.world.height - 5);

// 当玩家点击屏幕时，调用一次内部的匿名函数，在匿名函数中隐藏失去生命提示文本对象，并设置球的速度

game.input.onDown.addOnce(function () {

lifeLostText.visible = false;

ball.body.velocity.set(150, -150);

}, this);

}

// 如果玩家没有剩余生命了

else {

// 弹出提示框告知玩家游戏失败，并重新加载页面

alert('You lost, game over!');

location.reload();

}

}

// 当球击中挡板时调用的函数

function ballHitPaddle(ball, paddle) {

// 播放球的'wobble'动画

ball.animations.play('wobble');

// 根据挡板和球的相对位置设置球在x方向的速度，实现不同角度的反弹效果

ball.body.velocity.x = -1 \* 5 \* (paddle.x - ball.x);

}

// 游戏开始按钮点击时调用的函数

function startGame() {

// 销毁游戏开始按钮对象

startButton.destroy();

// 设置球的初始速度为（150, -150）

ball.body.velocity.set(150, -150);

// 将游戏正在进行的标志设置为true

playing = true;

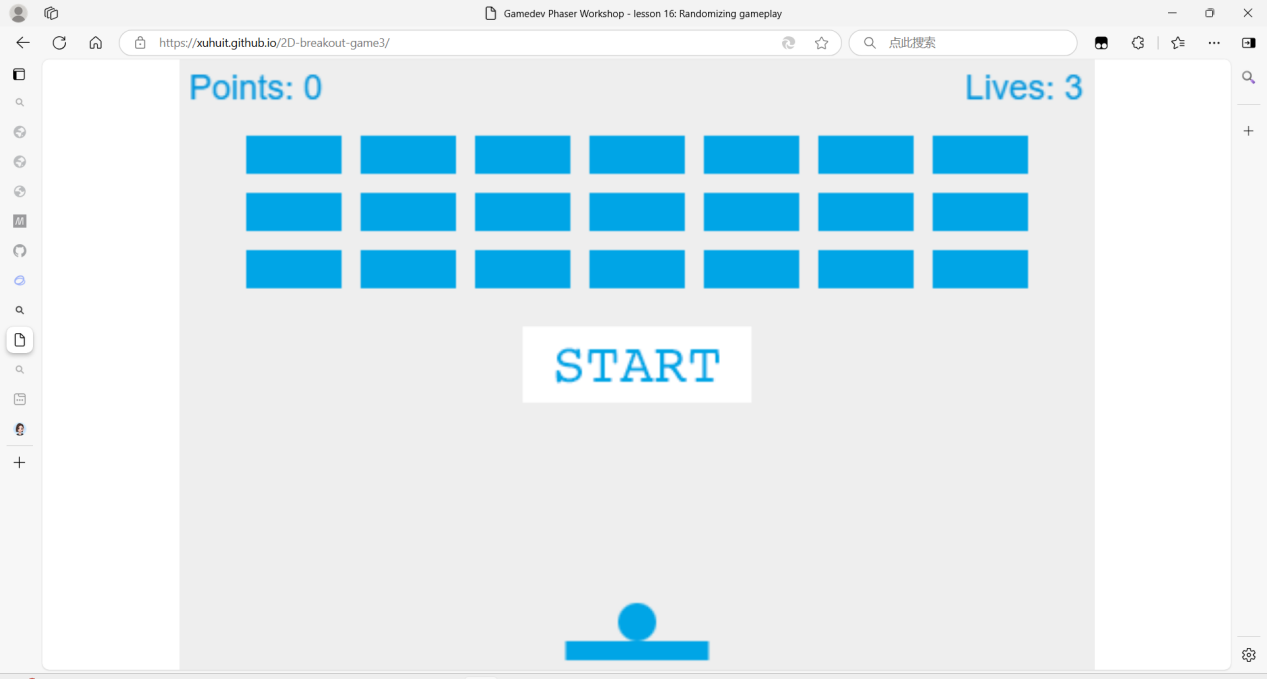
}

</script></body>

</html>

# 五、实验结果

结果截图：



效果描述：

1. 背景

* 背景颜色为浅灰色（#eee），整个界面简洁明了。

1. 游戏元素

* 界面中央偏上的位置有一个蓝色方块组成的矩形阵列，代表游戏中的砖块。这些砖块整齐排列，共有 3 列 7 行，总共 21 块。
* 界面左上角显示 “Points: 0”，表示玩家当前的得分是 0 分。
* 界面右上角显示 “Lives: 3”，表示玩家当前有 3 条生命。
* 界面中央有一个白色背景的矩形按钮，按钮上用蓝色大写字母写着 “START”，这是游戏的开始按钮。
* 界面底部中央有一个蓝色的小圆点，可能是游戏中的球，但目前它处于静止状态。
* 功能推测

1. 开始游戏

* 玩家点击 “START” 按钮后，游戏应该会开始，球会开始移动，玩家需要使用挡板（目前未显示）来反弹球，击打砖块。

1. 游戏目标

* 玩家的目标可能是通过控制挡板反弹球来击打所有的砖块，每击打一块砖块可以获得一定的分数。
* 当玩家失去所有生命（球掉出屏幕底部）或者成功击打所有砖块时，游戏会结束。

# 六、问题与解决

问题：遇到在本地服务器中Phaser.js在运行时没办法找到源映射map文件没办法正常运行的问题，找了很多种方法，官网也没有找到该源映射文件。

解决办法：上传到github上生成链接之后是可以直接使用的，也可以运行出来。

# 七、实验总结

### 技术收获

1. Phaser 游戏引擎的熟悉
2. 游戏初始化：学会了如何使用Phaser.Game函数来创建一个游戏实例，包括设置游戏的宽度、高度、渲染模式等参数。
3. 物理引擎：掌握了Phaser.Physics.ARCADE物理引擎的基本使用方法，例如设置物体的碰撞属性（如collideWorldBounds、body.immovable）和碰撞检测（如game.physics.arcade.collide）。
4. 动画创建：了解了如何为游戏对象添加动画，如通过animations.add为球对象添加了 “wobble” 动画。
5. 游戏资源管理
6. 预加载资源：在preload函数中，学会了加载不同类型的游戏资源，包括图片和精灵图（game.load.image和game.load.spritesheet），并且能够设置游戏的背景颜色和缩放模式等。
7. 游戏对象操作
8. 创建和设置对象：能够创建游戏中的各种对象，如球、挡板、砖块和按钮，并设置它们的属性，包括位置、锚点、大小等。
9. 对象分组：通过game.add.group创建了砖块组，方便对多个砖块进行统一管理。
10. 对象交互：实现了游戏对象之间的交互逻辑，例如球与挡板、球与砖块之间的碰撞处理函数（ballHitPaddle和ballHitBrick）。
11. 游戏逻辑处理
12. 得分与生命管理：实现了游戏得分和玩家生命的计算与显示，包括在ballHitBrick函数中更新得分，在ballLeaveScreen函数中处理玩家失去生命的情况。
13. 游戏状态控制：通过布尔变量playing来控制游戏的开始和暂停，例如在update函数中根据playing的值来决定挡板是否跟随鼠标移动。
14. 补间动画：学会了使用game.add.tween来创建补间动画，如在ballHitBrick函数中创建了砖块消失的动画。

### 心得

1. 问题解决能力

在开发过程中，遇到了诸如物理碰撞不符合预期、动画播放不正确等问题。通过查阅 Phaser 的官方文档、在线论坛和调试代码，逐渐找到了问题的解决方案，提升了自己的问题解决能力。

1. 逻辑思维的重要性

游戏开发需要严谨的逻辑思维，尤其是在处理游戏对象的交互和游戏状态的转换时。每一个函数的调用顺序和条件判断都需要仔细考虑，否则可能会导致游戏出现错误或不符合预期的行为。

1. 对游戏开发流程的理解

从预加载资源、创建游戏对象、设置初始状态到游戏的实时更新和交互处理，对整个游戏开发流程有了更清晰的理解。这有助于在未来的项目中更好地规划和组织代码。

1. 创意与实现的结合

虽然这是一个相对简单的小游戏，但在实现过程中可以思考如何在现有框架上增加更多的创意元素，如不同的关卡设计、特殊砖块效果等，体会到了将创意转化为实际代码的乐趣和挑战。