Relatório

Projeto ”RAMRaiders”

2º Semestre 24/25

G3PL4 Brandon Correia Gabriel Pedroso Miguel Correia

Descrição

O projeto desenvolvido é uma versão moderna inspirada no clássico **Galaga**, um jogo de naves espaciais lançado originalmente em 1981. O objetivo central do jogador é **sobreviver a ondas sucessivas de inimigos** que descem em formação e atacam com projéteis, enquanto o jogador move-se horizontalmente na parte inferior do ecrã e dispara para eliminar os inimigos.

Padrões de Projeto

1) **Strategy Pattern:** Este padrão é fundamental para definir a família de algoritmos de comportamento dos inimigos. Permite encapsular diferentes estratégias de movimento (como movimento em círculo, de cima para baixo ou em ziguezague) e de ataque (como ataque teleguiado, kamikaze ou disparo linear) em classes separadas. A classe `EnemyBehavior` utilizara estas estratégias, podendo alternar entre elas dinamicamente, o que facilita a criação de inimigos com comportamentos variados e a introdução de novas estratégias sem modificar o código central do inimigo.

2) **Observer Pattern**: Utilizado para criar uma relação de dependência entre os objetos do jogo, especificamente entre os inimigos e o jogador. Os inimigos (observadores) registam-se para serem notificados sobre mudanças de estado ou posição do jogador (sujeito). Isto permite que os inimigos reajam dinamicamente às ações do jogador, como ajustar a mira de um ataque teleguiado (`HomingShootAttack`) ou iniciar um movimento evasivo.

Características Principais

No que toca a controlos, o jogador controla uma **nave espacial** que se move **horizontalmente** ao longo da base da tela. A nave pode **disparar projéteis** verticais para destruir inimigos. Existe **limitação no número de projéteis ativos**, simulando o comportamento clássico de Galaga.

Em relação aos inimigos, os mesmos surgem em **formações organizadas**, movendo-se lateralmente e, por vezes, em ziguezague. Alguns inimigos **descem em voo kamikaze**, tentando colidir com o jogador ou atirar projéteis diretamente. Cada inimigo possui um **comportamento definido** pela sua classe **EnemyBehavior**, que pode incluir movimentos simples, ataques individuais ou ataques coordenados em grupo. Os Inimigos têm **estratégias de ataque trocáveis**, como:

- Ataque direto (disparo para a posição atual do jogador).
- Ataque em curva ou em padrão (ziguezague, forma de “V”, etc).
- Ataque em grupo (vários inimigos disparam ao mesmo tempo).

Em termos de colisões e danos, deu-se a utilização de colisores (circulares ou Poligonais) para deteção precisa de colisões. Quando um inimigo ou projétil colide com o jogador, o jogador perde uma vida e quando o projétil do jogador atinge um inimigo, o inimigo é destruído e pontos são atribuídos.

Os inimigos “observam” o jogador para reagir à sua posição:

- Alguns alinham os disparos à posição do jogador.
- Outros planeiam movimentos ou descidas em direção ao jogador.

A dificuldade aumenta com o tempo como por exemplo: Novos padrões de ataque são introduzidos, A frequência dos ataques aumenta e O número de inimigos por onda cresce.

Diagrama de Classes

Testes Unitários

```java
package test;

import game.*;
import game.behaviorItems.LinearShootAttack;

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import geometry.Ponto;
import geometry.Retangulo;
import java.io.ByteArrayOutputStream;
import java.io.PrintStream;

import static org.junit.jupiter.api.Assertions.*;

public class TestLinearShootAttack {
    private GameObject attacker;
    private LinearShootAttack shootAttack;
    private InputEvent inputEvent;

    @BeforeEach
    void setUp() {
        Ponto[] points = { new Ponto(2.0, 4), new Ponto(2.0, 0), new Ponto(0.0,
0.0), new Ponto(0.0, 4.0) };
        Transform transform = new Transform(new Ponto(1.0, 2), 0, 0, 1);
        Retangulo rectangle = new Retangulo(points, transform);

        attacker = new GameObject(
                "TEST_ATTACKER",
                transform,
                rectangle,
                new Behavior(),
                new Shape());

        shootAttack = new LinearShootAttack();
        inputEvent = new InputEvent();
        inputEvent.setPlayerMove("ATTACK", true);
    }

    @Test
    void testShootBullet() {
        GameObject bullet = (GameObject) shootAttack.execute(attacker, null);
        GameObject bullet2 = (GameObject) shootAttack.execute(attacker, null);

        assertNotNull(bullet, "Bullet should not be null after attack execution");
        assertEquals("LINEAR_BULLET 0", bullet.name());
        assertEquals("LINEAR_BULLET 1", bullet2.name());
        assertEquals(1.0, bullet.transform().position().x(), "Bullet x position
should be 1.0");

        // Test bullet movement
        Ponto initialPos = bullet.transform().position();
        bullet.onUpdate();
        assertNotEquals(initialPos, bullet.transform().position(), "Bullet should
move after update");
    }

    @Test
    void testInvalidAttacker() {
        ByteArrayOutputStream outContent = new ByteArrayOutputStream();
        System.setOut(new PrintStream(outContent));

        assertThrows(IllegalArgumentException.class, () ->
shootAttack.execute(null, null));
        assertEquals("LinearShootAttack:vi", outContent.toString().trim());

        assertThrows(IllegalArgumentException.class, () ->
shootAttack.execute(null, attacker));

    }
}
```

```java
package test;

import game.*;
import game.behaviorItems.FlyCircleMovement;
import game.behaviorItems.FlyTopDownMovement;
import geometry.Poligono;
import geometry.Ponto;
import geometry.Retangulo;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

public class TestEnemyBehavior
{
    private EnemyBehavior enemyBehavior;
    private GameObject enemy;

    @BeforeEach
    void setUp()
    {
        Ponto[] pts = {new Ponto(2.0, 4), new Ponto(2.0, 0), new Ponto(0.0, 0.0), new Ponto(0.0, 4.0)};
        Transform transform = new Transform(new Ponto(1.0, 2), 1, 90, 1);
        Poligono polygon = new Retangulo(pts, transform);
        enemyBehavior = new EnemyBehavior();


        enemy = new GameObject(
            "TestEnemy",
            transform,
            polygon,
            enemyBehavior,
            new Shape());
        this.enemy.onInit();
        this.enemyBehavior.activateMovement(true);
    }

    @Test
    void testZigZagMovement()
    {
        // Test multiple frames to verify zigzag pattern
        double[] expectedX = new double[4];

        // Store initial position
        Ponto initialPos = enemy.transform().position();


        // Run movement for 4 frames and store x positions
        for (int i = 0; i < 4; i++)
        {
            enemyBehavior.move();

            expectedX[i] = enemy.transform().position().x();
        }

        // Verify zigzag pattern (alternating left and right movement)
        assertTrue(expectedX[1] != expectedX[0], "Position should change after first move");
        assertTrue(expectedX[2] != expectedX[1], "Position should change after second move");

        assertTrue(Math.abs(expectedX[3] - initialPos.x()) <= 0.5, "Movement should stay within amplitude
bounds");
    }

    @Test
    void testYPositionMaintained()
    {
        double initialY = enemy.transform().position().y();
        // Move multiple times
        for (int i = 0; i < 10; i++)
        {
            enemyBehavior.move();
            assertEquals(initialY, enemy.transform().position().y(), "Y position should remain constant");
        }
    }

    @Test
    void testFlyCircleMovementBehaviorLeft()
    {

        FlyCircleMovement flyCircleMovement = new FlyCircleMovement();
        enemyBehavior.setMovement(flyCircleMovement);

        assertFalse(flyCircleMovement.isActive(), "Movement should not be active");

        flyCircleMovement.setActive(true);
        flyCircleMovement.setIsLeft(true);
        Ponto initialPosition = enemy.transform().position();

        for (int i = 0; i < 80; i++)
        {
            enemy.onUpdate();

            if(!flyCircleMovement.isActive())
                break;

            assertTrue(flyCircleMovement.isActive());
        }

        assertEquals(initialPosition.x(),enemy.transform().position().x(),0.1);

            assertTrue(fly.isActive());
        }
    }
}
```

```java
@Test
void testFlyCircleMovementBehaviorLeft()
{

    FlyCircleMovement flyCircleMovement = new FlyCircleMovement();
    enemyBehavior.setMovement(flyCircleMovement);

    assertFalse(flyCircleMovement.isActive(), "Movement should not be active");

    flyCircleMovement.setActive(true);
    flyCircleMovement.setIsLeft(true);
    Ponto initialPosition = enemy.transform().position();

    for (int i = 0; i < 80; i++)
    {
        enemy.onUpdate();

        if(!flyCircleMovement.isActive())
            break;

        assertTrue(flyCircleMovement.isActive());
    }

    assertEquals(initialPosition.x(),enemy.transform().position().x(),0.1);
    assertEquals(initialPosition.y(),enemy.transform().position().y(),0.1);

}

@Test
void testFlyCircleMovementBehaviorRight()
{

    FlyCircleMovement flyCircleMovement = new FlyCircleMovement();
    enemyBehavior.setMovement(flyCircleMovement);
    assertFalse(flyCircleMovement.isActive(), "Movement should not be active");
    flyCircleMovement.setActive(true);
    Ponto initialPosition = enemy.transform().position();


    for (int i = 0; i < 80; i++)
    {
        enemy.onUpdate();

        if(!flyCircleMovement.isActive())
            break;

        assertTrue(flyCircleMovement.isActive());
    }

    assertEquals(initialPosition.x(),enemy.transform().position().x(),0.1);
    assertEquals(initialPosition.y(),enemy.transform().position().y(),0.1);

}



@Test
void testFlyTopDownMovementBehaviorRight()
{

    FlyTopDownMovement fly = new FlyTopDownMovement();
    enemyBehavior.setMovement(fly);
    assertFalse(fly.isActive(), "Movement should not be active");
    fly.setActive(true);
    fly.setDirection(true);

    for (int i = 0; i < 100; i++)
    {
        System.out.println(enemy);
        enemy.onUpdate();

        if(!fly.isActive())
            break;

        assertTrue(fly.isActive());
    }


}
```

```java
package test;

import game.*;
import game.behaviorItems.KamikazeAttack;
import game.objectsInterface.IGameObject;
import geometry.Ponto;
import geometry.Retangulo;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

import java.io.ByteArrayOutputStream;
import java.io.PrintStream;

import static org.junit.jupiter.api.Assertions.*;

public class TestKamikazeAttack
{
    private GameObject attacker;
    private GameObject target;
    private KamikazeAttack kamikazeAttack;
    private Ponto[] points = {
            new Ponto(2.0, 4.0),
            new Ponto(2.0, 0.0),
            new Ponto(0.0, 0.0),
            new Ponto(0.0, 4.0)
    };

    @BeforeEach
    void setUp()
    {
        // Setup attacker
        Transform attackerTransform = new Transform(new Ponto(10.0, 10.0), 0, 0, 1);
        Retangulo attackerRect = new Retangulo(points, attackerTransform);
        attacker = new GameObject(
                "TEST_ATTACKER",
                attackerTransform,
                attackerRect,
                new EnemyBehavior(),
                new Shape()
        );

        // Setup target
        Transform targetTransform = new Transform(new Ponto(0.0, 0.0), 0, 0, 1);
        Retangulo targetRect = new Retangulo(points, targetTransform);
        target = new GameObject(
                "TEST_TARGET",
                targetTransform,
                targetRect,
                new PlayerBehavior(),
                new Shape()
        );

        kamikazeAttack = new KamikazeAttack();
        target.onInit();
        attacker.onInit();
    }

    @Test
    void testValidKamikazeAttack()
    {
        IGameObject result = kamikazeAttack.execute(attacker, target);
        assertNotNull(result, "Attack result should not be null");
        assertEquals(attacker, result, "Should return the modified attacker");

        Ponto velocity = ((GameObject)result).velocity();
        assertNotNull(velocity, "Velocity should be set");
        assertEquals(1.0, Math.sqrt(velocity.x() * velocity.x() + velocity.y() * velocity.y()), 0.001,
                "Velocity magnitude should be 1.0");

        assertEquals(225,attacker.transform().angle());
    }

    @Test
    void testInvalidAttackerAndTarget()
    {
        ByteArrayOutputStream outContent = new ByteArrayOutputStream();
        System.setOut(new PrintStream(outContent));

        // Test null attacker
        assertThrows(IllegalArgumentException.class,
                () -> kamikazeAttack.execute(null, target));
        assertEquals("KamikazeAttack:vi", outContent.toString().trim());

        // Test null target
        assertThrows(IllegalArgumentException.class,
                () -> kamikazeAttack.execute(attacker, null));

        // Test same object for attacker and target
        assertThrows(IllegalArgumentException.class,
                () -> kamikazeAttack.execute(attacker, attacker));
    }
```

```
@Test
    void testMovementDirection()
    {
        IGameObject result = kamikazeAttack.execute(attacker, target);
        Ponto velocity = ((GameObject)result).velocity();

        // Calculate expected angle to target
        double dx = target.transform().position().x() - attacker.transform().position().x();
        double dy = target.transform().position().y() - attacker.transform().position().y();
        double expectedAngle = Math.atan2(dy, dx);

        // Calculate actual angle from velocity
        double actualAngle = Math.atan2(velocity.y(), velocity.x());

        assertEquals(expectedAngle, actualAngle, 0.001,
                "Velocity direction should match angle to target");
    }


    @Test
    void testAttackAngleAndCollisionFromDifferentPosition()
    {
        Transform attackerTransform = new Transform(new Ponto(4.0, 1.0), 0,180 , 1);
        Retangulo attackerRect = new Retangulo(new Ponto[] {
                new Ponto(4.0, 2.0),
                new Ponto(4.0, 0.0),
                new Ponto(0.0, 0.0),
                new Ponto(0.0, 2.0)
        }, attackerTransform);
            attacker = new GameObject("TEST_ATTACKER", attackerTransform, attackerRect, new
EnemyBehavior(), new Shape());
            attacker.onInit();

        Transform targetTransform = new Transform(new Ponto(20.0, 2.0), 0, 0, 1);
        Retangulo targetRect = new Retangulo(points, targetTransform);
            target = new GameObject("TEST_TARGET", targetTransform, targetRect, new PlayerBehavior(), new
Shape());
            target.onInit();

            double angle = attacker.transform().angle();
            kamikazeAttack.execute(attacker, target);

        assertNotEquals(angle, target.transform().angle(), 0.0, "Angle should be different");
        for (int i = 0; i < 14; i++)
            attacker.onUpdate();
        assertTrue(attacker.collider().colision(target.collider()));
    }


    @Test
    void testAttackFromSameAngle()
    {
        Transform attackerTransform = new Transform(new Ponto(0.0, 6.0), 0,270 , 1);
        Retangulo attackerRect = new Retangulo(new Ponto[] {
                new Ponto(4.0, 2.0),
                new Ponto(4.0, 0.0),
                new Ponto(0.0, 0.0),
                new Ponto(0.0, 2.0)
        }, attackerTransform);
        attacker = new GameObject("TEST_ATTACKER", attackerTransform, attackerRect, new EnemyBehavior(),
new Shape());
        attacker.onInit();

        Transform targetTransform = new Transform(new Ponto(0.0, 0.0), 0, 90, 1);
        Retangulo targetRect = new Retangulo(points, targetTransform);
        target = new GameObject("TEST_TARGET", targetTransform, targetRect, new PlayerBehavior(), new
Shape());
        target.onInit();

        kamikazeAttack.execute(attacker, target);
        for (int i = 0; i < 3; i++)
            attacker.onUpdate();

        assertTrue(attacker.collider().colision(target.collider()));
        assertEquals(270, attacker.transform().angle());
    }

}
```

```java
package test;
import game.*;
import game.objectsInterface.IGameObject;
import geometry.Ponto;
import geometry.Poligono;
import geometry.Retangulo;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

import java.util.ArrayList;

import static org.junit.jupiter.api.Assertions.*;

public class TestPlayerBehavior
{
    private PlayerBehavior playerBehavior0;
    private GameObject gameObject0;

    private PlayerBehavior playerBehavior1;
    private GameObject gameObject1;

    private InputEvent inputEvent;

    @BeforeEach
    void setUp()
    {
        playerBehavior0 = new PlayerBehavior();

        Ponto[] pts = {new Ponto(2.0, 4), new Ponto(2.0, 0), new Ponto(0.0, 0.0), new Ponto(0.0, 4.0)};
        Transform transform = new Transform(new Ponto(1.0, 2), 0, 0, 1);
        Poligono polygon = new Retangulo(pts, transform);

        gameObject0 = new GameObject("PLAYER 0", transform, polygon, playerBehavior0, new Shape());
        gameObject0.onInit();


        playerBehavior1 = new PlayerBehavior();
        gameObject1 = new GameObject("PLAYER 1", transform, polygon, playerBehavior1, new Shape());
        gameObject1.onInit();

        inputEvent = new InputEvent();
    }

    @Test
    void attackNotCreatesProjectileWhenNotAttacking()
    {
        IGameObject result = playerBehavior0.attack(inputEvent);
        assertNull(result);
    }

    @Test
    void attackReturnsNullWhenAlreadyAttacking()
    {
        inputEvent.setPlayerMove("MOUSE_LEFT", true);
        IGameObject firstAttack = playerBehavior0.attack(inputEvent);
        assertNotNull(firstAttack);

        IGameObject secondAttack = playerBehavior0.attack(inputEvent);
        assertNull(secondAttack);
    }

    @Test
    void onCollisionReducesLifeByOneWhenHit()
    {
        ArrayList<IGameObject> collisions = new ArrayList<>();
        collisions.add(gameObject1);

        playerBehavior0.onCollision(collisions);
        assertEquals(2, playerBehavior0.life());
    }

    @Test
    void onCollisionMakesPlayerInvincibleTemporarily() throws InterruptedException
    {
        ArrayList<IGameObject> collisions = new ArrayList<>();
        collisions.add(gameObject1);

        playerBehavior0.onCollision(collisions);
        assertTrue(playerBehavior0.isInvincible());

        Thread.sleep(1500);
        assertFalse(playerBehavior0.isInvincible());
    }
```

```java
@Test
    void playerRemainsAliveWhenHitWithInvincibility() throws InterruptedException
    {
        ArrayList<IGameObject> collisions = new ArrayList<>();
        collisions.add(gameObject1);


        playerBehavior0.onCollision(collisions); // hit
        playerBehavior0.onCollision(collisions);

        Thread.sleep(1300);
        assertEquals(2, playerBehavior0.life());
        playerBehavior0.onCollision(collisions); // hit
        assertEquals(1, playerBehavior0.life());
        assertTrue(playerBehavior0.isEnabled());
    }

    @Test
    void playerIsDestroyedWhenLifeReachesZero() throws InterruptedException
    {
        ArrayList<IGameObject> collisions = new ArrayList<>();
        collisions.add(gameObject1);


        assertEquals(3, playerBehavior0.life());
        playerBehavior0.onCollision(collisions); // hit
        Thread.sleep(1300);
        assertEquals(2, playerBehavior0.life());
        playerBehavior0.onCollision(collisions);// hit
        Thread.sleep(1300);
        assertEquals(1, playerBehavior0.life());
        playerBehavior0.onCollision(collisions); // hit

        assertEquals(0, playerBehavior0.life());
        assertFalse(playerBehavior0.isEnabled());


    }

    @Test
    void evasiveManeuverMovesPlayerRight()
    {
        inputEvent.setPlayerMove("RIGHT", true);
        Ponto initialPosition = gameObject0.transform().position();

        playerBehavior0.evasiveManeuver(inputEvent);

        Ponto newPosition = gameObject0.transform().position();
        assertTrue(newPosition.x() > initialPosition.x());
        assertEquals(initialPosition.y(), newPosition.y());
    }

    @Test
    void evasiveManeuverMovesPlayerLeft() throws InterruptedException
    {
        inputEvent.setPlayerMove("LEFT", true);
        Ponto initialPosition = gameObject0.transform().position();

        playerBehavior0.evasiveManeuver(inputEvent);
        assertTrue(playerBehavior0.isInvincible());

        Ponto newPosition = gameObject0.transform().position();
        assertTrue(newPosition.x() < initialPosition.x());
        assertEquals(initialPosition.y(), newPosition.y());

        Thread.sleep(1200);
        assertFalse(playerBehavior0.isInvincible());
    }

}
```

```java
package test;
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import geometry.Ponto;
import game.*;
import geometry.Circulo;

public class GameObjectTest {
    private Transform transform;
    private Collider collider;
    private GameObjects gameObject;
    private String name;
    private Ponto velocity;
    private int velocityLayer;
    private double rotateSpeed;
    private double scaleDiff;

    @BeforeEach
    void setUp() {
        transform = new Transform(new Ponto(0.0, 0.0), 1, 0.0, 1.0);
        collider = new Circulo(new Ponto(0.0, 0.0), 1.0, transform);
        name = "TestObject";
        velocity = new Ponto(1.0, 0.0);
        velocityLayer = 1;
        rotateSpeed = 10.0;
        scaleDiff = 0.1;

        gameObject = new GameObjects(name, transform, collider, velocity, velocityLayer, rotateSpeed,
scaleDiff);
    }

    @Test
    void testConstructor() {
        assertNotNull(gameObject);
        assertEquals(name, gameObject.name());
        assertEquals(transform, gameObject.transform());
        assertEquals(collider, gameObject.collider());

        assertThrows(IllegalArgumentException.class, () -> new GameObjects(null, transform, collider,
velocity, velocityLayer, rotateSpeed, scaleDiff));
        assertThrows(IllegalArgumentException.class, () -> new GameObjects(name, null, collider,
velocity, velocityLayer, rotateSpeed, scaleDiff));
        assertThrows(IllegalArgumentException.class, () -> new GameObjects(name, transform, null,
velocity, velocityLayer, rotateSpeed, scaleDiff));
    }

    @Test
    void testRotate() {
        gameObject.rotate();
        assertEquals(10.0, transform.angle());

        gameObject.rotate();
        assertEquals(20.0, transform.angle());
    }

    @Test
    void testRotateNegativeSpeed() {
        GameObjects negativeRotationObject = new GameObjects(name, transform, collider, velocity,
velocityLayer, -15.0, scaleDiff);
        negativeRotationObject.rotate();
        assertEquals(345.0, transform.angle());
    }

    @Test
    void testScale() {
        double initialScale = transform.scale();
        gameObject.scale();
        assertEquals(initialScale + scaleDiff, transform.scale());
    }
```

```java
@Test
    void testScaleNegative() {
        GameObjects negativeScaleObject = new GameObjects(name, transform, collider, velocity,
velocityLayer, rotateSpeed, -0.2);
        negativeScaleObject.scale();
        assertEquals(0.8, transform.scale());
    }

    @Test
    void testMove() {
        Ponto initialPosition = transform.position();
        gameObject.move();
        Ponto expectedPosition = new Ponto(initialPosition.x() + velocity.x(), initialPosition.y() +
velocity.y());
        assertEquals(expectedPosition.x(), transform.position().x());
        assertEquals(expectedPosition.y(), transform.position().y());
    }

    @Test
    void testMoveZeroVelocity() {
        GameObjects stationaryObject = new GameObjects(name, transform, collider, new Ponto(0.0, 0.0),
velocityLayer, rotateSpeed, scaleDiff);
        stationaryObject.move();
        assertEquals(transform.position().x(), 0.0);
        assertEquals(transform.position().y(), 0.0);
    }

    @Test
    void testUpdate() {
        Ponto initialPosition = transform.position();
        double initialScale = transform.scale();
        double initialAngle = transform.angle();

        gameObject.update();

        assertEquals(initialAngle + rotateSpeed, transform.angle());
        assertEquals(initialScale + scaleDiff, transform.scale());
        Ponto expectedPosition = new Ponto(initialPosition.x() + velocity.x(), initialPosition.y() +
velocity.y());
        assertEquals(expectedPosition.x(), transform.position().x());
        assertEquals(expectedPosition.y(), transform.position().y());
    }

    @Test
    void testMultipleSequentialUpdates() {
        gameObject.update();
        gameObject.update();
        gameObject.update();

        assertEquals(30.0, transform.angle());
        assertEquals(1.3, transform.scale());
        assertEquals(3.0, transform.position().x());
    }


    @Test
    void testLayerMovement() {
        int initialLayer = transform.layer();
        gameObject.move();
        assertEquals(initialLayer + velocityLayer, transform.layer());
    }


    @Test
    void testMultipleUpdates() {
        Ponto initialPosition = gameObject.transform().position();
        gameObject.update();
        gameObject.update();
        Ponto finalPosition = gameObject.transform().position();
        assertNotEquals(initialPosition, finalPosition);
    }

    @Test
    void testToString() {
        String expected = name + "\n" + transform.toString() + "\n" + collider.toString();
        assertEquals(expected, gameObject.toString());
    }
}
```

```java
package test;
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import geometry.Ponto;
import game.Transform;

class TransformTest {

    private Transform transform;

    @BeforeEach
    void setUp() {
        transform = new Transform(new Ponto(1.0, 2.0), 1, 45.0, 1.0);
    }

    @Test
    void constructsTransformWithParameters() {
        assertNotNull(transform);
        assertEquals(new Ponto(1.0, 2.0), transform.position());
        assertEquals(1, transform.layer());
        assertEquals(45.0, transform.angle());
        assertEquals(1.0, transform.scale());
    }

    @Test
    void constructsTransformByCopying() {
        Transform copiedTransform = new Transform(transform);
        assertNotNull(copiedTransform);
        assertEquals(transform.position(), copiedTransform.position());
        assertEquals(transform.layer(), copiedTransform.layer());
        assertEquals(transform.angle(), copiedTransform.angle());
        assertEquals(transform.scale(), copiedTransform.scale());
    }

    @Test
    void movesTransform() {
        transform.move(new Ponto(2.0, 3.0), 1);
        assertEquals(new Ponto(3.0, 5.0), transform.position());
        assertEquals(2, transform.layer());
    }

    @Test
    void rotatesTransform() {
        transform.rotate(15.0);
        assertEquals(60.0, transform.angle());
    }

    @Test
    void scalesTransform() {
        transform.scale(2.0);
        assertEquals(3.0, transform.scale());
    }

}
```

```java
package test;
import static org.junit.jupiter.api.Assertions.*;

import geometry.Circulo;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import geometry.Ponto;
import game.GameObjects;
import game.GameEngine;
import game.Transform;
import game.Collider;


class GameEngineTest {
    private GameEngine gameEngine;
    private GameObjects gameObject1;
    private GameObjects gameObject2;

    @BeforeEach
    void setUp() {
        gameEngine = new GameEngine();

        Transform transform1 = new Transform(new Ponto(1, 1), 0, 0, 1);
        Collider collider1 = new Circulo(transform1.position(), 1.0, transform1);
        gameObject1 = new GameObjects("Object1", transform1, collider1, new Ponto(1, 0), 0, 0, 0);

        Transform transform2 = new Transform(new Ponto(3, 3), 0, 0, 1);
        Collider collider2 = new Circulo(transform2.position(), 1.5, transform2);
        gameObject2 = new GameObjects("Object2", transform2, collider2, new Ponto(-1, 0), 0, 0, 0);
    }

    @Test
    void testRemoveGameObject() {
        gameEngine.add(gameObject1);
        gameEngine.destroy(gameObject1);
        assertEquals(0, gameEngine.size());
    }

    @Test
    void testUpdateGameObjects() {
        gameEngine.add(gameObject1);
        gameEngine.add(gameObject2);

        gameEngine.update();

        assertEquals(new Ponto(2, 1), gameObject1.transform().position());
        assertEquals(new Ponto(2, 3), gameObject2.transform().position());
    }

    @Test
    void testCollisionDetection() {
        gameEngine.add(gameObject1);
        gameEngine.add(gameObject2);

        gameObject2.transform().move(new Ponto(1, 1), 0); // Moving Object2 closer to Object1
        gameEngine.update();

        assertTrue(gameObject1.collider().colision(gameObject2.collider()));
    }

    @Test
    void testMultipleObjectCollisions() {
        Transform transform3 = new Transform(new Ponto(1.5, 1.5), 0, 0, 1);
        Collider collider3 = new Circulo(transform3.position(), 1.0, transform3);
        GameObjects gameObject3 = new GameObjects("Object3", transform3, collider3, new Ponto(0, 0), 0,
0, 0);

        gameEngine.add(gameObject1);
        gameEngine.add(gameObject2);
        gameEngine.add(gameObject3);

        gameEngine.update();
        assertTrue(gameObject1.collider().colision(gameObject3.collider()));
    }

    @Test
    void testLayerBasedUpdates() {
        Transform t1 = new Transform(new Ponto(0, 0), 1, 0, 1);
        Transform t2 = new Transform(new Ponto(0, 0), 2, 0, 1);

        GameObjects obj1 = new GameObjects("Layer1", t1, new Circulo(t1.position(), 1.0, t1),
                                           new Ponto(1, 0), 1, 0, 0);
        GameObjects obj2 = new GameObjects("Layer2", t2, new Circulo(t2.position(), 1.0, t2),
                                           new Ponto(1, 0), 2, 0, 0);

        gameEngine.add(obj1);
        gameEngine.add(obj2);
        gameEngine.update();

        assertNotEquals(obj1.transform().position(), obj2.transform().position());
    }
}
```

```java
package test;
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;
import geometry.*;

    public class PontoTest {

        @Test
        void constructorWithPolarCoordinates() {
            Ponto p1 = new Ponto(8, 5);
            Ponto p2 = new Ponto(4, 7);
            Ponto p3 = new Ponto(3, 7);
            assertEquals(8, p1.x());
            assertEquals(4, p2.x());
            assertEquals(3, p3.x());
            assertEquals(5, p1.y());
            assertEquals(7, p2.y());
            assertEquals(7, p3.y());
        }

        @Test
        void constructorWithCartesianCoordinates() {
            Ponto p = new Ponto(3, 4);
            assertEquals(5, p.r(), 0.001);
            assertEquals(53.1301, p.theta(), 0.001);
        }

        @Test
        void copyConstructor() {
            Ponto p1 = new Ponto(5.0, 45);
            Ponto p2 = new Ponto(p1);
            assertEquals(p1.r(), p2.r());
            assertEquals(p1.theta(), p2.theta());
        }




        @Test
        void distanciaRadial() {
            Ponto p1 = new Ponto(5, 45);
            Ponto p2 = new Ponto(5, 135);
            assertEquals(7.071, p1.distanciaRadial(p2), 0.001);
        }

        @Test
        void distancia() {
            Ponto p1 = new Ponto(3, 4);
            Ponto p2 = new Ponto(6, 8);
            assertEquals(5, p1.distancia(p2), 0.001);
        }

        @Test
        void equals() {
            Ponto p1 = new Ponto(5, 45);
            Ponto p2 = new Ponto(5, 45);
            assertTrue(p1.equals(p2));
        }

        @Test
        void notEquals() {
            Ponto p1 = new Ponto(5, 45);
            Ponto p2 = new Ponto(5, 60);
            assertFalse(p1.equals(p2));
        }

        @Test
        void toStringTest() {
            Ponto p = new Ponto(3, 4);
            assertEquals("(3,4)", p.toString());
        }


        @Test
        void xCoordinate() {
            Ponto p = new Ponto(5, 45);
            assertEquals(4, p.x());
        }

        @Test
        void yCoordinate() {
            Ponto p = new Ponto(5, 45.0);
            assertEquals(5, p.r());
        }

        @Test
        void invalidPolarCoordinates() {
            assertThrows(IllegalArgumentException.class, () -> new Ponto(-1.0, 45));
            assertThrows(IllegalArgumentException.class, () -> new Ponto(5.0, 100));
        }

        @Test
        void invalidCartesianCoordinates() {
            assertThrows(IllegalArgumentException.class, () -> new Ponto(-1, 4));
            assertThrows(IllegalArgumentException.class, () -> new Ponto(3, -4));
        }
    }
```

```java
package test;
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.BeforeEach;
import geometry.*;
import game.Transform;

class ColliderTest
{
    private Transform transform;
    private Circulo circulo;
    private Poligono poligono;
    private Ponto[] vertices;

    @BeforeEach
    void setUp() {
        transform = new Transform(new Ponto(0.0, 0.0),1, 0.0, 1.0);
        circulo = new Circulo(0.0, 0.0, 5.0, transform);

        // Create a square polygon
        vertices = new Ponto[] {
            new Ponto(0.0, 0.0),
            new Ponto(10.0, 0.0),
            new Ponto(10.0, 10.0),
            new Ponto(0.0, 10.0)
        };
        poligono = new Poligono(vertices, transform);
    }

    // Test Circulo constructors
    @Test
    void testCirculoConstructors() {
        // Test constructor with x, y, r
        Circulo c1 = new Circulo(1.0, 2.0, 3.0, transform);
        assertEquals(1.0, c1.centro().x());
        assertEquals(2.0, c1.centro().y());
        assertEquals(3.0, c1.r());

        // Test constructor with Ponto and r
        Ponto p = new Ponto(4.0, 5.0);
        Circulo c2 = new Circulo(p, 6.0, transform);
        assertEquals(4.0, c2.centro().x());
        assertEquals(5.0, c2.centro().y());
        assertEquals(6.0, c2.r());

        // Test constructor with transform position
        Circulo c3 = new Circulo(7.0, transform);
        assertEquals(0.0, c3.centro().x());
        assertEquals(0.0, c3.centro().y());
        assertEquals(7.0, c3.r());
    }

    // Test Poligono constructors
    @Test
    void testPoligonoConstructors() {
        // Test constructor with Ponto array
        Ponto[] pts = new Ponto[] {
            new Ponto(0.0, 0.0),
            new Ponto(5.0, 0.0),
            new Ponto(5.0, 5.0)
        };
        Poligono p1 = new Poligono(pts, transform);
        assertEquals(3, p1.vertices().length);
        assertEquals(3, p1.lados().length);

        // Test constructor with string
        String pontosStr = "0.0 0.0 5.0 0.0 5.0 5.0";
        Poligono p2 = new Poligono(pontosStr, transform);
        assertEquals(3, p2.vertices().length);
        assertEquals(3, p2.lados().length);

        // Test with decimal points
        String pontosStr2 = "1.5 2.5 3.5 2.5 3.5 4.5 1.5 4.5";
        Poligono p3 = new Poligono(pontosStr2, transform);
        assertEquals(4, p3.vertices().length);
        assertEquals(4, p3.lados().length);
    }

    // Test perimetro() method
    @Test
    void testPerimetro() {
        // Test circle perimeter
        assertEquals(2.0 * Math.PI * 5.0, circulo.perimetro());

        // Test polygon perimeter (square with side 10)
        assertEquals(40.0, poligono.perimetro());

        // Test triangle perimeter
        Ponto[] pts = new Ponto[] {
            new Ponto(0.0, 0.0),
            new Ponto(3.0, 0.0),
            new Ponto(0.0, 4.0)
        };
        Poligono p = new Poligono(pts, transform);
        assertEquals(12.0, p.perimetro());
    }
```

```java
// Test intersection methods
    @Test
    void testIntersecta() {
        // Test circle-circle intersection
        Circulo c1 = new Circulo(0.0, 0.0, 5.0, transform);
        Circulo c2 = new Circulo(3.0, 3.0, 3.0, transform);
        assertTrue(c1.intersecta(c2));

        // Test circle-polygon intersection
        assertTrue(circulo.intersecta(poligono));

        // Test polygon-polygon intersection
        Ponto[] pts2 = new Ponto[] {
            new Ponto(5.0, 5.0),
            new Ponto(15.0, 5.0),
            new Ponto(15.0, 15.0),
            new Ponto(5.0, 15.0)
        };
        Poligono p2 = new Poligono(pts2, transform);
        assertTrue(poligono.intersecta(p2));

        // Test non-intersecting shapes
        Circulo c3 = new Circulo(20.0, 20.0, 2.0, transform);
        assertFalse(circulo.intersecta(c3));
        assertFalse(poligono.intersecta(c3));
    }

    // Test isInside methods
    @Test
    void testIsInside() {
        // Test circle inside polygon
        Circulo c = new Circulo(5.0, 5.0, 2.0, transform);
        assertTrue(poligono.isInside(c));

        // Test polygon inside circle
        Ponto[] smallPts = new Ponto[] {
            new Ponto(2.0, 2.0),
            new Ponto(3.0, 2.0),
            new Ponto(3.0, 3.0),
            new Ponto(2.0, 3.0)
        };
        Poligono smallPoly = new Poligono(smallPts, transform);
        assertTrue(circulo.isInside(smallPoly));

        // Test circle with center at polygon center
        Circulo c2 = new Circulo(5.0, 5.0, 4.0, transform);
        assertTrue(poligono.isInside(c2));
    }

    // Test update methods
    @Test
    void testUpdate() {
        // Test circle update
        transform.move(new Ponto(10.0, 10.0), 1);
        transform.scale(2.0);
        circulo.update();
        assertEquals(10.0, circulo.centro().x());
        assertEquals(10.0, circulo.centro().y());
        assertEquals(10.0, circulo.r());

        // Test polygon update
        transform.move(new Ponto(20.0, 20.0), 1);
        transform.scale(1.5);
        transform.rotate(45.0);
        poligono.update();
        assertEquals(20.0, poligono.centro().x());
        assertEquals(20.0, poligono.centro().y());
    }

    // Test translation methods
    @Test
    void testTranslacao() {
        // Test circle translation
        Ponto newCenter = new Ponto(10.0, 10.0);
        Circulo c = circulo.translacao(newCenter);
        assertEquals(10.0, c.centro().x());
        assertEquals(10.0, c.centro().y());

        // Test polygon translation
        Poligono p = poligono.translacao(newCenter);
        assertEquals(10.0, p.centro().x());
        assertEquals(10.0, p.centro().y());

        // Test translation with negative coordinates
        Ponto negCenter = new Ponto(-5.0, -5.0);
        Circulo c2 = circulo.translacao(negCenter);
        assertEquals(-5.0, c2.centro().x());
        assertEquals(-5.0, c2.centro().y());
    }
```

```java
// Test rotation and scaling
    @Test
    void testRotacaoAndEscalar() {
        // Test polygon rotation
        Poligono p = poligono.rotacao(45.0);
        assertNotNull(p);

        // Test polygon scaling
        Poligono scaled = poligono.escalar(2.0);
        assertEquals(10.0, scaled.centro().x());
        assertEquals(10.0, scaled.centro().y());

        // Test rotation with negative angle
        Poligono p2 = poligono.rotacao(-45.0);
        assertNotNull(p2);

        // Test scaling with small factor
        Poligono scaled2 = poligono.escalar(0.5);
        assertEquals(5.0, scaled2.centro().x());
        assertEquals(5.0, scaled2.centro().y());
    }

    // Test invalid cases
    @Test
    void testInvalidCases() {
        // Test invalid circle radius
        assertThrows(IllegalArgumentException.class, () -> {
            new Circulo(0.0, 0.0, -5.0, transform);
        });

        // Test invalid polygon (collinear points)
        Ponto[] invalidPts = new Ponto[] {
            new Ponto(0.0, 0.0),
            new Ponto(5.0, 0.0),
            new Ponto(10.0, 0.0)  // Collinear with previous points
        };
        assertThrows(IllegalArgumentException.class, () -> {
            new Poligono(invalidPts, transform);
        });

        // Test invalid polygon (self-intersecting)
        Ponto[] invalidPts2 = new Ponto[] {
            new Ponto(0.0, 0.0),
            new Ponto(5.0, 5.0),
            new Ponto(0.0, 5.0),
            new Ponto(5.0, 0.0)
        };
        assertThrows(IllegalArgumentException.class, () -> {
            new Poligono(invalidPts2, transform);
        });
    }

    // Test edge cases for intersection
    @Test
    void testEdgeCasesIntersection() {
        // Test circles just touching
        Circulo c1 = new Circulo(0.0, 0.0, 5.0, transform);
        Circulo c2 = new Circulo(10.0, 0.0, 5.0, transform);
        assertTrue(c1.intersecta(c2));

        // Test circles not touching
        Circulo c3 = new Circulo(11.0, 0.0, 5.0, transform);
        assertFalse(c1.intersecta(c3));

        // Test circle exactly on polygon edge
        Circulo c4 = new Circulo(5.0, 0.0, 1.0, transform);
        assertTrue(poligono.intersecta(c4));

        // Test circle outside polygon
        Circulo c5 = new Circulo(15.0, 15.0, 2.0, transform);
        assertFalse(poligono.intersecta(c5));

        // Test circle with center at polygon vertex
        Circulo c6 = new Circulo(0.0, 0.0, 1.0, transform);
        assertTrue(poligono.intersecta(c6));
    }
```

```java
// Test edge cases for isInside
    @Test
    void testEdgeCasesIsInside() {
        // Test circle exactly on polygon boundary
        Circulo c1 = new Circulo(5.0, 0.0, 1.0, transform);
        assertFalse(poligono.isInside(c1));

        // Test circle with center on polygon boundary
        Circulo c2 = new Circulo(5.0, 0.0, 0.5, transform);
        assertFalse(poligono.isInside(c2));

        // Test polygon exactly on circle boundary
        Ponto[] boundaryPts = new Ponto[] {
            new Ponto(5.0, 0.0),
            new Ponto(5.0, 5.0),
            new Ponto(0.0, 5.0)
        };
        Poligono boundaryPoly = new Poligono(boundaryPts, transform);
        assertFalse(circulo.isInside(boundaryPoly));

        // Test circle with center at polygon center
        Circulo c3 = new Circulo(5.0, 5.0, 4.0, transform);
        assertTrue(poligono.isInside(c3));
    }

    // Test multiple transformations
    @Test
    void testMultipleTransformations() {
        // Test multiple translations
        Ponto newCenter1 = new Ponto(10.0, 10.0);
        Ponto newCenter2 = new Ponto(20.0, 20.0);
        Circulo c = circulo.translacao(newCenter1).translacao(newCenter2);
        assertEquals(20.0, c.centro().x());
        assertEquals(20.0, c.centro().y());

        // Test multiple rotations
        Poligono p = poligono.rotacao(45.0).rotacao(45.0);
        assertNotNull(p);

        // Test multiple scaling
        Poligono scaled = poligono.escalar(2.0).escalar(1.5);
        assertEquals(15.0, scaled.centro().x());
        assertEquals(15.0, scaled.centro().y());

        // Test combined transformations
        Poligono p2 = poligono.translacao(newCenter1).rotacao(45.0).escalar(2.0);
        assertNotNull(p2);
    }

    // Test toString method
    @Test
    void testToString() {
        // Test circle toString
        Circulo c = new Circulo(1.5, 2.5, 3.5, transform);
        String expectedCircle = "1.50 2.50 3.50";
        assertEquals(expectedCircle, c.toString());

        // Test polygon toString
        Ponto[] pts = new Ponto[] {
            new Ponto(1.5, 2.5),
            new Ponto(3.5, 4.5)
        };
        Poligono p = new Poligono(pts, transform);
        String expectedPolygon = "1.50 2.50 3.50 4.50";
        assertEquals(expectedPolygon, p.toString());
    }
```

```java
// Test distance calculations
    @Test
    void testDistanceCalculations() {
        // Test distance from circle center to point
        Ponto p = new Ponto(3.0, 4.0);
        assertEquals(5.0, circulo.distanciaAoCentro(p));

        // Test distance with negative coordinates
        Ponto p2 = new Ponto(-3.0, -4.0);
        assertEquals(5.0, circulo.distanciaAoCentro(p2));

        // Test distance to circle center
        Ponto p3 = new Ponto(0.0, 0.0);
        assertEquals(0.0, circulo.distanciaAoCentro(p3));
    }

    // Test polygon edge cases
    @Test
    void testPolygonEdgeCases() {
        // Test minimum valid polygon (triangle)
        Ponto[] minPts = new Ponto[] {
            new Ponto(0.0, 0.0),
            new Ponto(1.0, 0.0),
            new Ponto(0.0, 1.0)
        };
        Poligono minPoly = new Poligono(minPts, transform);
        assertNotNull(minPoly);

        // Test polygon with very small sides
        Ponto[] smallPts = new Ponto[] {
            new Ponto(0.0, 0.0),
            new Ponto(0.0001, 0.0),
            new Ponto(0.0001, 0.0001),
            new Ponto(0.0, 0.0001)
        };
        Poligono smallPoly = new Poligono(smallPts, transform);
        assertNotNull(smallPoly);

        // Test polygon with very large coordinates
        Ponto[] largePts = new Ponto[] {
            new Ponto(0.0, 0.0),
            new Ponto(1000000.0, 0.0),
            new Ponto(1000000.0, 1000000.0),
            new Ponto(0.0, 1000000.0)
        };
        Poligono largePoly = new Poligono(largePts, transform);
        assertNotNull(largePoly);

        // Test polygon with decimal coordinates
        Ponto[] decimalPts = new Ponto[] {
            new Ponto(0.5, 0.5),
            new Ponto(1.5, 0.5),
            new Ponto(1.5, 1.5),
            new Ponto(0.5, 1.5)
        };
        Poligono decimalPoly = new Poligono(decimalPts, transform);
        assertNotNull(decimalPoly);
    }
```

```java
// Test circle edge cases
    @Test
    void testCircleEdgeCases() {
        // Test circle with very small radius
        Circulo c1 = new Circulo(0.0, 0.0, 0.0001, transform);
        assertNotNull(c1);

        // Test circle with very large radius
        Circulo c2 = new Circulo(0.0, 0.0, 1000000.0, transform);
        assertNotNull(c2);

        // Test circle with decimal coordinates
        Circulo c3 = new Circulo(1.5, 2.5, 3.5, transform);
        assertNotNull(c3);

        // Test circle with negative center coordinates
        Circulo c4 = new Circulo(-1.0, -1.0, 1.0, transform);
        assertNotNull(c4);
    }

    // Test complex transformations
    @Test
    void testComplexTransformations() {
        // Test circle with multiple transformations
        transform.move(new Ponto(10.0, 10.0), 1);
        transform.scale(2.0);
        transform.rotate(45.0);
        circulo.update();
        assertEquals(10.0, circulo.centro().x());
        assertEquals(10.0, circulo.centro().y());
        assertEquals(10.0, circulo.r());

        // Test polygon with multiple transformations
        transform.move(new Ponto(20.0, 20.0), 1);
        transform.scale(1.5);
        transform.rotate(90.0);
        poligono.update();
        assertEquals(20.0, poligono.centro().x());
        assertEquals(20.0, poligono.centro().y());
    }

    @Test
    void testEdgeCaseCollisions() {
        // Test collision at exact edge
        Circulo c1 = new Circulo(new Ponto(0, 0), 1.0, transform);
        Circulo c2 = new Circulo(new Ponto(2, 0), 1.0, transform);
        assertTrue(c1.colision(c2));

        // Test barely touching circles
        Circulo c3 = new Circulo(new Ponto(2.001, 0), 1.0, transform);
        assertFalse(c1.colision(c3));
    }

    @Test
    void testRotatedCollisions() {
        Transform t1 = new Transform(new Ponto(0, 0), 1, 45.0, 1.0);
        Transform t2 = new Transform(new Ponto(1, 1), 1, -45.0, 1.0);

        Poligono p1 = new Poligono(vertices, t1);
        Poligono p2 = new Poligono(vertices, t2);

        assertTrue(p1.colision(p2));
    }
}
```

```java
package test;
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;
import geometry.*;

public class SegmentoTest {

    @Test
    void constructorWithValidPoints() {
        Ponto p1 = new Ponto(1, 2);
        Ponto p2 = new Ponto(3, 4);
        Segmento seg = new Segmento(p1, p2);
        assertEquals(p1, seg.a());
        assertEquals(p2, seg.b());
    }

    @Test
    void constructorWithEqualPoints() {
        Ponto p1 = new Ponto(1, 2);
        assertThrows(IllegalArgumentException.class, () -> new Segmento(p1, p1));
    }

    @Test
    void lengthCalculation() {
        Ponto p1 = new Ponto(0, 0);
        Ponto p2 = new Ponto(3, 4);
        Segmento seg = new Segmento(p1, p2);
        assertEquals(5, seg.length(), 0.001);
    }

    @Test
    void perpendicularSegments() {
        Ponto p1 = new Ponto(0, 0);
        Ponto p2 = new Ponto(1, 0);
        Ponto p3 = new Ponto(0, 0);
        Ponto p4 = new Ponto(0, 1);
        Segmento seg1 = new Segmento(p1, p2);
        Segmento seg2 = new Segmento(p3, p4);
        assertTrue(seg1.isPerpendicular(seg2));
    }

    @Test
    void nonPerpendicularSegments() {
        Ponto p1 = new Ponto(0, 0);
        Ponto p2 = new Ponto(1, 1);
        Ponto p3 = new Ponto(0, 0);
        Ponto p4 = new Ponto(1, 0);
        Segmento seg1 = new Segmento(p1, p2);
        Segmento seg2 = new Segmento(p3, p4);
        assertFalse(seg1.isPerpendicular(seg2));
    }

    @Test
    void intersectingSegments() {
        Ponto p1 = new Ponto(0, 0);
        Ponto p2 = new Ponto(2, 2);
        Ponto p3 = new Ponto(0, 2);
        Ponto p4 = new Ponto(2, 0);
        Segmento seg1 = new Segmento(p1, p2);
        Segmento seg2 = new Segmento(p3, p4);
        assertTrue(seg1.intersecta(seg2));
    }

    @Test
    void nonIntersectingSegments() {
        Ponto p1 = new Ponto(0, 0);
        Ponto p2 = new Ponto(1, 1);
        Ponto p3 = new Ponto(2, 2);
        Ponto p4 = new Ponto(3, 3);
        Segmento seg1 = new Segmento(p1, p2);
        Segmento seg2 = new Segmento(p3, p4);
        assertFalse(seg1.intersecta(seg2));
    }
```

```java
@Test
    void pointOnSegment() {
        Ponto p1 = new Ponto(0, 0);
        Ponto p2 = new Ponto(2, 2);
        Ponto p3 = new Ponto(1, 1);
        Segmento seg = new Segmento(p1, p2);
        assertTrue(seg.isPointOnSegment(p3));
    }

    @Test
    void pointNotOnSegment() {
        Ponto p1 = new Ponto(0, 0);
        Ponto p2 = new Ponto(2, 2);
        Ponto p3 = new Ponto(3, 3);
        Segmento seg = new Segmento(p1, p2);
        assertFalse(seg.isPointOnSegment(p3));
    }

    @Test
    void pointOnSegmentEdgeCase() {
        Ponto p1 = new Ponto(0, 0);
        Ponto p2 = new Ponto(2, 2);
        Ponto p3 = new Ponto(2, 2);
        Segmento seg = new Segmento(p1, p2);
        assertTrue(seg.isPointOnSegment(p3));
    }

    @Test
    void pointNotOnSegmentEdgeCase() {
        Ponto p1 = new Ponto(0, 0);
        Ponto p2 = new Ponto(2, 2);
        Ponto p3 = new Ponto(2, 3);
        Segmento seg = new Segmento(p1, p2);
        assertFalse(seg.isPointOnSegment(p3));
    }

    @Test
    void intersectingSegmentsEdgeCase() {
        Ponto p1 = new Ponto(0, 0);
        Ponto p2 = new Ponto(2, 2);
        Ponto p3 = new Ponto(2, 2);
        Ponto p4 = new Ponto(4, 4);
        Segmento seg1 = new Segmento(p1, p2);
        Segmento seg2 = new Segmento(p3, p4);
        assertFalse(seg1.intersecta(seg2));
    }

    @Test
    void nonIntersectingSegmentsEdgeCase() {
        Ponto p1 = new Ponto(0, 0);
        Ponto p2 = new Ponto(2, 2);
        Ponto p3 = new Ponto(2, 2);
        Ponto p4 = new Ponto(4, 4);
        Segmento seg1 = new Segmento(p1, p2);
        Segmento seg2 = new Segmento(p3, p4);
        assertFalse(seg1.intersecta(seg2));
    }
}
```

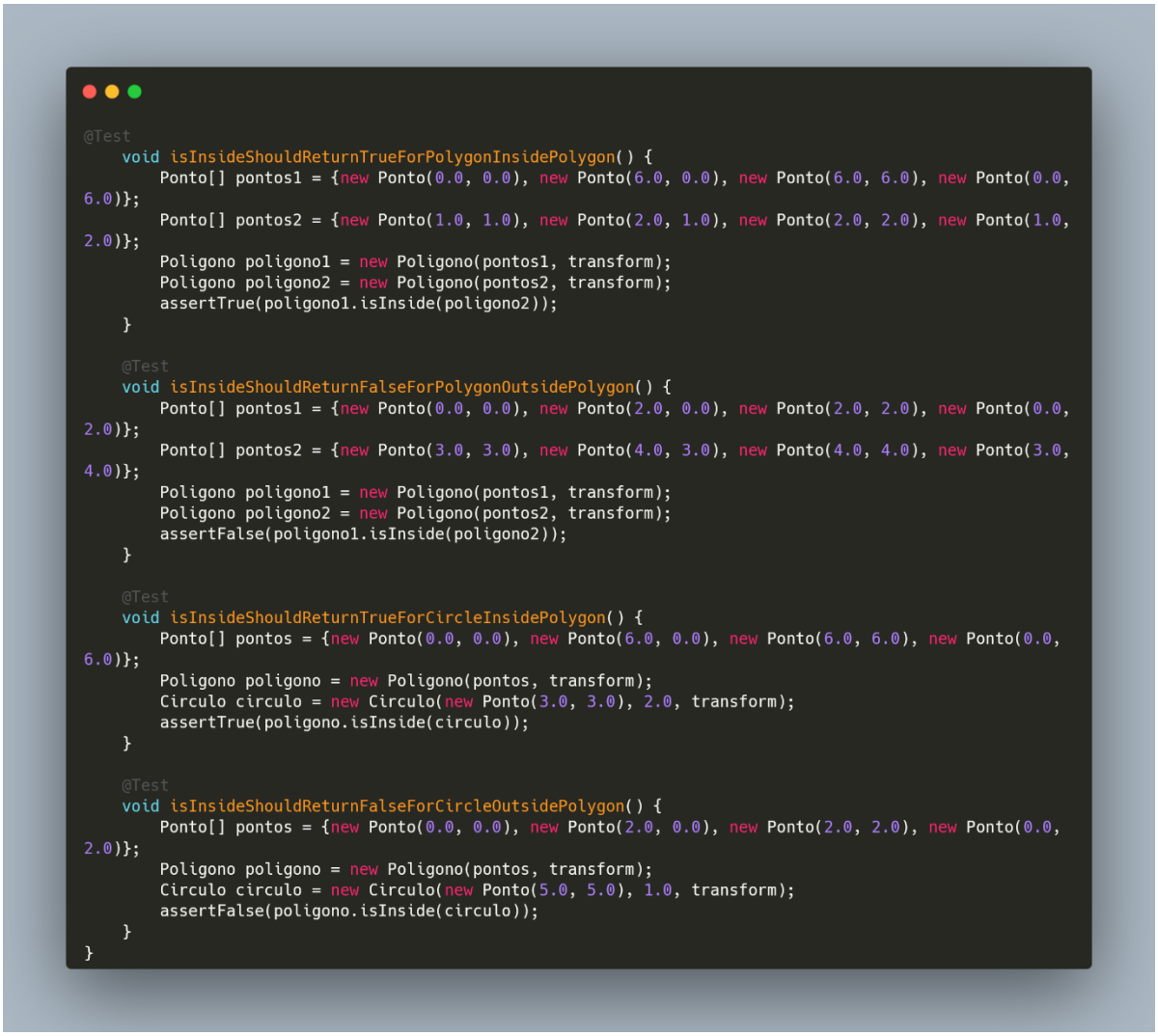
```java
package test;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.BeforeEach;
import geometry.*;
import game.Transform;

class PoligonoTest {
    private Transform transform;
    private Poligono square;
    private Poligono triangle;
    private Poligono pentagon;
    private Ponto[] squareVertices;
    private Ponto[] triangleVertices;
    private Ponto[] pentagonVertices;

    @BeforeEach
    void setUp() {
        transform = new Transform(new Ponto(0.0, 0.0), 1, 0.0, 1.0);

        // Square vertices
        squareVertices = new Ponto[] {
            new Ponto(0.0, 0.0),
            new Ponto(4.0, 0.0),
            new Ponto(4.0, 4.0),
            new Ponto(0.0, 4.0)
        };
        square = new Poligono(squareVertices, transform);

        // Triangle vertices
        triangleVertices = new Ponto[] {
            new Ponto(0.0, 0.0),
            new Ponto(3.0, 0.0),
            new Ponto(0.0, 4.0)
        };
        triangle = new Poligono(triangleVertices, transform);

        // Pentagon vertices
        pentagonVertices = new Ponto[] {
            new Ponto(0.0, 0.0),
            new Ponto(4.0, 0.0),
            new Ponto(5.0, 3.0),
            new Ponto(2.0, 5.0),
            new Ponto(-1.0, 3.0)
        };
        pentagon = new Poligono(pentagonVertices, transform);
    }

    // Constructor Tests
    @Test
    void testConstructors() {
        // Test constructor with Ponto array
        assertNotNull(square);
        assertEquals(4, square.vertices().length);
        assertEquals(4, square.lados().length);

        // Test constructor with string
        String pontosStr = "0.0 0.0 4.0 0.0 4.0 4.0 0.0 4.0";
        Poligono p2 = new Poligono(pontosStr, transform);
        assertEquals(4, p2.vertices().length);
        assertEquals(4, p2.lados().length);

        // Test invalid constructors
        assertThrows(IllegalArgumentException.class, () -> {
            new Poligono((Ponto[]) null, transform);
        });

        assertThrows(IllegalArgumentException.class, () -> {
            new Poligono(new Ponto[]{}, transform);
        });

        assertThrows(IllegalArgumentException.class, () -> {
            new Poligono(new Ponto[]{new Ponto(0.0, 0.0)}, transform);
        });
```

```
assertThrows(IllegalArgumentException.class, () -> {
            new Poligono(new Ponto[]{new Ponto(0.0, 0.0), new Ponto(1.0, 1.0)}, transform);
        });

        // Test invalid string constructor
        assertThrows(IllegalArgumentException.class, () -> {
            new Poligono("", transform);
        });

        assertThrows(IllegalArgumentException.class, () -> {
            new Poligono("0.0 0.0", transform);
        });

        assertThrows(IllegalArgumentException.class, () -> {
            new Poligono("0.0 0.0 1.0 1.0", transform);
        });

        assertThrows(NumberFormatException.class, () -> {
            new Poligono("invalid string", transform);
        });
    }

    // Access Method Tests
    @Test
    void testLados() {
        Segmento[] lados = square.lados();
        assertEquals(4, lados.length);
        assertEquals(4.0, lados[0].length());
        assertEquals(4.0, lados[1].length());
        assertEquals(4.0, lados[2].length());
        assertEquals(4.0, lados[3].length());
    }

    @Test
    void testVertices() {
        Ponto[] vertices = square.vertices();
        assertEquals(4, vertices.length);
        assertEquals(0.0, vertices[0].x());
        assertEquals(0.0, vertices[0].y());
        assertEquals(4.0, vertices[1].x());
        assertEquals(0.0, vertices[1].y());
    }

    @Test
    void testToString() {
        String expected = "0.0 0.0 4.0 0.0 4.0 4.0 0.0 4.0";
        assertEquals(expected, square.toString());
    }

    @Test
    void testPerimetro() {
        assertEquals(16.0, square.perimetro());
        assertEquals(12.0, triangle.perimetro());
        assertTrue(pentagon.perimetro() > 0);
    }

    @Test
    void testCentro() {
        Ponto squareCentro = square.centro();
        assertEquals(2.0, squareCentro.x());
        assertEquals(2.0, squareCentro.y());

        Ponto triangleCentro = triangle.centro();
        assertEquals(1.0, triangleCentro.x());
        assertEquals(1.33, triangleCentro.y(), 0.01);

        Ponto pentagonCentro = pentagon.centro();
        assertEquals(2.0, pentagonCentro.x());
        assertEquals(2.2, pentagonCentro.y(), 0.1);
    }
```

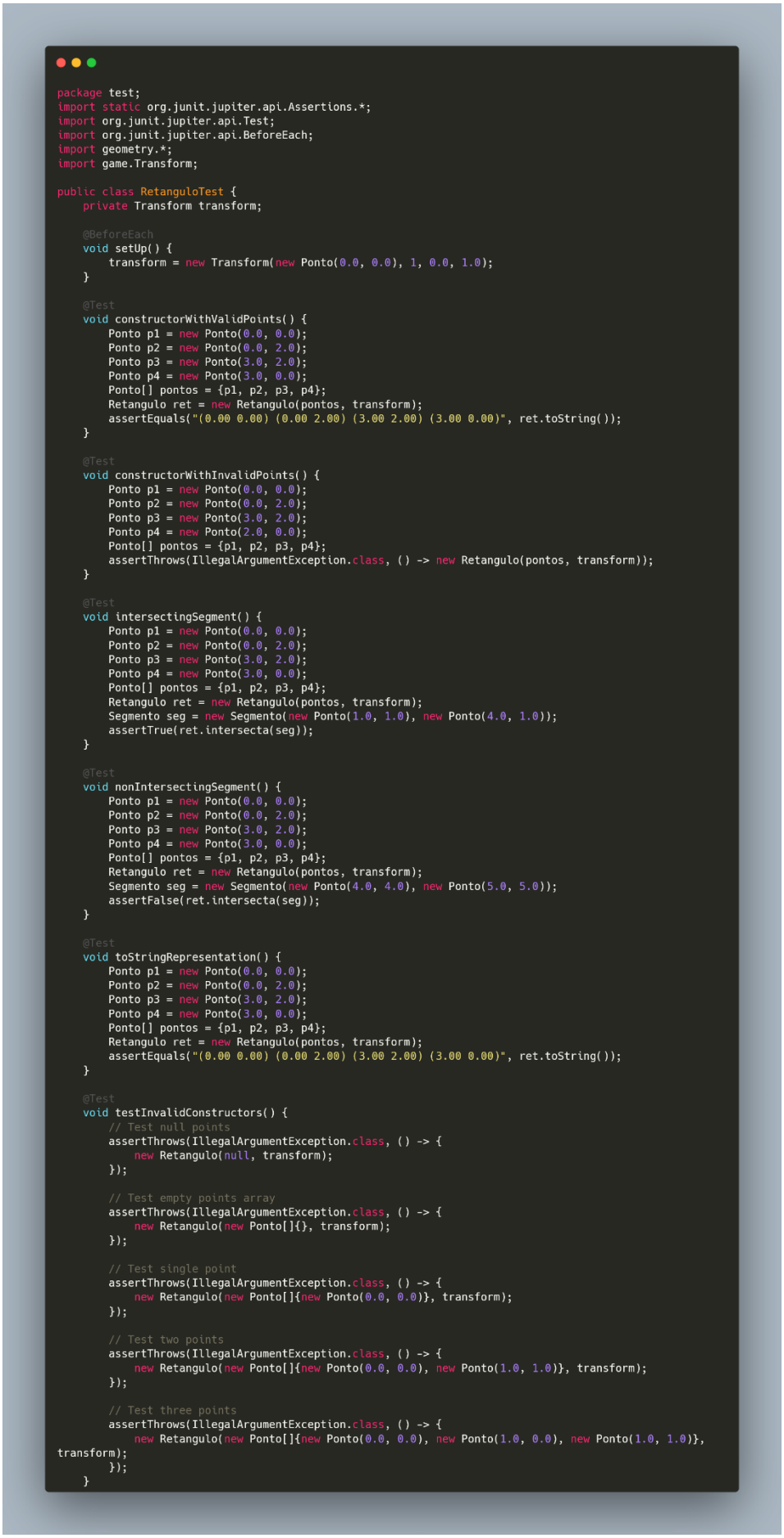
```java
// Transformation Tests
    @Test
    void testTranslacao() {
        // Test translacao with dx, dy
        Poligono translated = (Poligono) square.translacao(2, 3);
        assertEquals(2.0, translated.vertices()[0].x());
        assertEquals(3.0, translated.vertices()[0].y());

        // Test translacao with new center
        Ponto newCenter = new Ponto(5.0, 5.0);
        Poligono translatedToCenter = square.translacao(newCenter);
        assertEquals(5.0, translatedToCenter.centro().x());
        assertEquals(5.0, translatedToCenter.centro().y());

        // Test zero translation
        assertSame(square, square.translacao(0, 0));
    }

    @Test
    void testRotacao() {
        // Test 90-degree rotation
        Poligono rotated90 = square.rotacao(90.0);
        assertEquals(2.0, rotated90.centro().x());
        assertEquals(2.0, rotated90.centro().y());

        // Test 180-degree rotation
        Poligono rotated180 = square.rotacao(180.0);
        assertEquals(2.0, rotated180.centro().x());
        assertEquals(2.0, rotated180.centro().y());

        // Test 360-degree rotation
        Poligono rotated360 = square.rotacao(360.0);
        assertEquals(0.0, rotated360.vertices()[0].x());
        assertEquals(0.0, rotated360.vertices()[0].y());
    }

    @Test
    void testEscalar() {
        // Test scale up
        Poligono scaledUp = square.escalar(2.0);
        assertEquals(8.0, scaledUp.perimetro());

        // Test scale down
        Poligono scaledDown = square.escalar(0.5);
        assertEquals(4.0, scaledDown.perimetro());

        // Test scale with 1.0
        assertSame(square, square.escalar(1.0));
    }

    // Intersection and Containment Tests
    @Test
    void testIsInside() {
        // Test polygon inside polygon
        Ponto[] smallSquare = new Ponto[] {
            new Ponto(1.0, 1.0),
            new Ponto(2.0, 1.0),
            new Ponto(2.0, 2.0),
            new Ponto(1.0, 2.0)
        };
        Poligono smallSquarePoligono = new Poligono(smallSquare, transform);
        assertTrue(square.isInside(smallSquarePoligono));
    }
```

```java
@Test
    void intersectaShouldReturnFalseForNonIntersectingSegments() {
        Ponto[] pontos = {new Ponto(0.0, 0.0), new Ponto(2.0, 0.0), new Ponto(2.0, 2.0), new Ponto(0.0,
2.0)};
        Poligono poligono = new Poligono(pontos, transform);
        Segmento segmento = new Segmento(new Ponto(3.0, 3.0), new Ponto(4.0, 4.0));
        assertFalse(poligono.intersecta(segmento));
    }

    @Test
    void intersectaShouldReturnTrueForIntersectingPolygons() {
        Ponto[] pontos1 = {new Ponto(0.0, 0.0), new Ponto(2.0, 0.0), new Ponto(2.0, 2.0), new Ponto(0.0,
2.0)};
        Ponto[] pontos2 = {new Ponto(1.0, 1.0), new Ponto(3.0, 1.0), new Ponto(3.0, 3.0), new Ponto(1.0,
3.0)};
        Poligono poligono1 = new Poligono(pontos1, transform);
        Poligono poligono2 = new Poligono(pontos2, transform);
        assertTrue(poligono1.intersecta(poligono2));
    }

    @Test
    void intersectaShouldReturnFalseForNonIntersectingPolygons() {
        Ponto[] pontos1 = {new Ponto(0.0, 0.0), new Ponto(2.0, 0.0), new Ponto(2.0, 2.0), new Ponto(0.0,
2.0)};
        Ponto[] pontos2 = {new Ponto(3.0, 3.0), new Ponto(5.0, 3.0), new Ponto(5.0, 5.0), new Ponto(3.0,
5.0)};
        Poligono poligono1 = new Poligono(pontos1, transform);
        Poligono poligono2 = new Poligono(pontos2, transform);
        assertFalse(poligono1.intersecta(poligono2));
    }

    @Test
    void intersectaShouldReturnTrueForIntersectingCircle() {
        Ponto[] pontos = {new Ponto(0.0, 0.0), new Ponto(2.0, 0.0), new Ponto(2.0, 2.0), new Ponto(0.0,
2.0)};
        Poligono poligono = new Poligono(pontos, transform);
        Circulo circulo = new Circulo(new Ponto(1.0, 1.0), 1.0, transform);
        assertTrue(poligono.intersecta(circulo));
    }

    @Test
    void intersectaShouldReturnFalseForNonIntersectingCircle() {
        Ponto[] pontos = {new Ponto(0.0, 0.0), new Ponto(2.0, 0.0), new Ponto(2.0, 2.0), new Ponto(0.0,
2.0)};
        Poligono poligono = new Poligono(pontos, transform);
        Circulo circulo = new Circulo(new Ponto(5.0, 5.0), 1.0, transform);
        assertFalse(poligono.intersecta(circulo));
    }

    @Test
    void centroShouldCalculateCorrectCentroidForSquare() {
        Ponto[] pontos = {new Ponto(0.0, 0.0), new Ponto(2.0, 0.0), new Ponto(2.0, 2.0), new Ponto(0.0,
2.0)};
        Poligono poligono = new Poligono(pontos, transform);
        Ponto centro = poligono.centro();
        assertEquals(1.0, centro.x());
        assertEquals(1.0, centro.y());
    }

    @Test
    void centroShouldCalculateCorrectCentroidForTriangle() {
        Ponto[] pontos = {new Ponto(0.0, 0.0), new Ponto(4.0, 0.0), new Ponto(2.0, 3.0)};
        Poligono poligono = new Poligono(pontos, transform);
        Ponto centro = poligono.centro();
        assertEquals(2.0, centro.x());
        assertEquals(1.0, centro.y());
    }
```

```java
@Test
    void rotacaoShouldRotatePolygonCorrectly() {
        Ponto[] pontos = {new Ponto(1.0, 1.0), new Ponto(2.0, 1.0), new Ponto(2.0, 2.0), new Ponto(1.0,
2.0)};
        Poligono poligono = new Poligono(pontos, transform);
        Poligono rotated = poligono.rotacao(90.0);
        assertEquals("(1.00 1.00) (1.00 2.00) (0.00 2.00) (0.00 1.00)", rotated.toString());
    }

    @Test
    void escalarShouldScalePolygonCorrectly() {
        Ponto[] pontos = {new Ponto(1.0, 1.0), new Ponto(2.0, 1.0), new Ponto(2.0, 2.0), new Ponto(1.0,
2.0)};
        Poligono poligono = new Poligono(pontos, transform);
        Poligono scaled = poligono.escalar(2.0);
        assertEquals("(0.00 0.00) (2.00 0.00) (2.00 2.00) (0.00 2.00)", scaled.toString());
    }

    @Test
    void centroShouldReturnCorrectCentroidForComplexPolygon() {
        Ponto[] pontos = {new Ponto(0.0, 0.0), new Ponto(4.0, 0.0), new Ponto(4.0, 3.0), new Ponto(0.0,
3.0)};
        Poligono poligono = new Poligono(pontos, transform);
        Ponto centro = poligono.centro();
        assertEquals(2.0, centro.x());
        assertEquals(1.5, centro.y());
    }

    @Test
    void centroShouldCalculateCorrectCentroidForPentagon() {
        Ponto[] pontos = {new Ponto(0.0, 0.0), new Ponto(4.0, 0.0), new Ponto(5.0, 3.0), new Ponto(2.0,
5.0), new Ponto(-1.0, 3.0)};
        Poligono poligono = new Poligono(pontos, transform);
        Ponto centro = poligono.centro();
        assertEquals(2.0, centro.x());
        assertEquals(2.0, centro.y());
    }

    @Test
    void rotacaoShouldRotatePolygonBy180Degrees() {
        Ponto[] pontos = {new Ponto(1.0, 1.0), new Ponto(2.0, 1.0), new Ponto(2.0, 2.0), new Ponto(1.0,
2.0)};
        Poligono poligono = new Poligono(pontos, transform);
        Poligono rotated = poligono.rotacao(180.0);
        assertEquals("(1.00 1.00) (0.00 1.00) (0.00 0.00) (1.00 0.00)", rotated.toString());
    }

    @Test
    void escalarShouldScalePolygonByHalf() {
        Ponto[] pontos = {new Ponto(2.0, 2.0), new Ponto(4.0, 2.0), new Ponto(4.0, 4.0), new Ponto(2.0,
4.0)};
        Poligono poligono = new Poligono(pontos, transform);
        Poligono scaled = poligono.escalar(0.5);
        assertEquals("(2.00 2.00) (3.00 2.00) (3.00 3.00)  (2.00 3.00)", scaled.toString());
    }

    @Test
    void centroShouldReturnCorrectCentroidForConcavePolygon() {
        Ponto[] pontos = {new Ponto(0.0, 0.0), new Ponto(4.0, 0.0), new Ponto(4.0, 4.0), new Ponto(2.0,
2.0), new Ponto(0.0, 4.0)};
        Poligono poligono = new Poligono(pontos, transform);
        Ponto centro = poligono.centro();
        assertEquals(2.0, centro.x());
        assertEquals(2.0, centro.y());
    }

    @Test
    void rotacaoShouldRotatePolygonBy45Degrees() {
        Ponto[] pontos = {new Ponto(1.0, 1.0), new Ponto(2.0, 1.0), new Ponto(2.0, 2.0), new Ponto(1.0,
2.0)};
        Poligono poligono = new Poligono(pontos, transform);
        Poligono rotated = poligono.rotacao(45.0);
        assertEquals("(1.00 1.00) (1.70 0.29) (2.41 1.00) (1.70 1.70)", rotated.toString());
    }

    @Test
    void translacaoShouldTranslatePolygonByNegativeValues() {
        Ponto[] pontos = {new Ponto(2.0, 2.0), new Ponto(4.0, 2.0), new Ponto(4.0, 4.0), new Ponto(2.0,
4.0)};
        Poligono poligono = new Poligono(pontos, transform);
        Poligono translated = (Poligono) poligono.translacao(-1, -1);
        assertEquals("(1.00 1.00) (3.00 1.00) (3.00 3.00) (1.00 3.00)", translated.toString());
    }
```



```java
@Test
    void isInsideShouldReturnTrueForPolygonInsidePolygon() {
        Ponto[] pontos1 = {new Ponto(0.0, 0.0), new Ponto(6.0, 0.0), new Ponto(6.0, 6.0), new Ponto(0.0,
6.0)};
        Ponto[] pontos2 = {new Ponto(1.0, 1.0), new Ponto(2.0, 1.0), new Ponto(2.0, 2.0), new Ponto(1.0,
2.0)};
        Poligono poligono1 = new Poligono(pontos1, transform);
        Poligono poligono2 = new Poligono(pontos2, transform);
        assertTrue(poligono1.isInside(poligono2));
    }

    @Test
    void isInsideShouldReturnFalseForPolygonOutsidePolygon() {
        Ponto[] pontos1 = {new Ponto(0.0, 0.0), new Ponto(2.0, 0.0), new Ponto(2.0, 2.0), new Ponto(0.0,
2.0)};
        Ponto[] pontos2 = {new Ponto(3.0, 3.0), new Ponto(4.0, 3.0), new Ponto(4.0, 4.0), new Ponto(3.0,
4.0)};
        Poligono poligono1 = new Poligono(pontos1, transform);
        Poligono poligono2 = new Poligono(pontos2, transform);
        assertFalse(poligono1.isInside(poligono2));
    }

    @Test
    void isInsideShouldReturnTrueForCircleInsidePolygon() {
        Ponto[] pontos = {new Ponto(0.0, 0.0), new Ponto(6.0, 0.0), new Ponto(6.0, 6.0), new Ponto(0.0,
6.0)};
        Poligono poligono = new Poligono(pontos, transform);
        Circulo circulo = new Circulo(new Ponto(3.0, 3.0), 2.0, transform);
        assertTrue(poligono.isInside(circulo));
    }

    @Test
    void isInsideShouldReturnFalseForCircleOutsidePolygon() {
        Ponto[] pontos = {new Ponto(0.0, 0.0), new Ponto(2.0, 0.0), new Ponto(2.0, 2.0), new Ponto(0.0,
2.0)};
        Poligono poligono = new Poligono(pontos, transform);
        Circulo circulo = new Circulo(new Ponto(5.0, 5.0), 1.0, transform);
        assertFalse(poligono.isInside(circulo));
    }
}
```



```java
package test;
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.BeforeEach;
import geometry.*;
import game.Transform;

public class RetanguloTest {
    private Transform transform;

    @BeforeEach
    void setUp() {
        transform = new Transform(new Ponto(0.0, 0.0), 1, 0.0, 1.0);
    }

    @Test
    void constructorWithValidPoints() {
        Ponto p1 = new Ponto(0.0, 0.0);
        Ponto p2 = new Ponto(0.0, 2.0);
        Ponto p3 = new Ponto(3.0, 2.0);
        Ponto p4 = new Ponto(3.0, 0.0);
        Ponto[] pontos = {p1, p2, p3, p4};
        Retangulo ret = new Retangulo(pontos, transform);
        assertEquals("(0.00 0.00) (0.00 2.00) (3.00 2.00) (3.00 0.00)", ret.toString());
    }

    @Test
    void constructorWithInvalidPoints() {
        Ponto p1 = new Ponto(0.0, 0.0);
        Ponto p2 = new Ponto(0.0, 2.0);
        Ponto p3 = new Ponto(3.0, 2.0);
        Ponto p4 = new Ponto(2.0, 0.0);
        Ponto[] pontos = {p1, p2, p3, p4};
        assertThrows(IllegalArgumentException.class, () -> new Retangulo(pontos, transform));
    }

    @Test
    void intersectingSegment() {
        Ponto p1 = new Ponto(0.0, 0.0);
        Ponto p2 = new Ponto(0.0, 2.0);
        Ponto p3 = new Ponto(3.0, 2.0);
        Ponto p4 = new Ponto(3.0, 0.0);
        Ponto[] pontos = {p1, p2, p3, p4};
        Retangulo ret = new Retangulo(pontos, transform);
        Segmento seg = new Segmento(new Ponto(1.0, 1.0), new Ponto(4.0, 1.0));
        assertTrue(ret.intersecta(seg));
    }

    @Test
    void nonIntersectingSegment() {
        Ponto p1 = new Ponto(0.0, 0.0);
        Ponto p2 = new Ponto(0.0, 2.0);
        Ponto p3 = new Ponto(3.0, 2.0);
        Ponto p4 = new Ponto(3.0, 0.0);
        Ponto[] pontos = {p1, p2, p3, p4};
        Retangulo ret = new Retangulo(pontos, transform);
        Segmento seg = new Segmento(new Ponto(4.0, 4.0), new Ponto(5.0, 5.0));
        assertFalse(ret.intersecta(seg));
    }

    @Test
    void toStringRepresentation() {
        Ponto p1 = new Ponto(0.0, 0.0);
        Ponto p2 = new Ponto(0.0, 2.0);
        Ponto p3 = new Ponto(3.0, 2.0);
        Ponto p4 = new Ponto(3.0, 0.0);
        Ponto[] pontos = {p1, p2, p3, p4};
        Retangulo ret = new Retangulo(pontos, transform);
        assertEquals("(0.00 0.00) (0.00 2.00) (3.00 2.00) (3.00 0.00)", ret.toString());
    }

    @Test
    void testInvalidConstructors() {
        // Test null points
        assertThrows(IllegalArgumentException.class, () -> {
            new Retangulo(null, transform);
        });

        // Test empty points array
        assertThrows(IllegalArgumentException.class, () -> {
            new Retangulo(new Ponto[]{}, transform);
        });

        // Test single point
        assertThrows(IllegalArgumentException.class, () -> {
            new Retangulo(new Ponto[]{new Ponto(0.0, 0.0)}, transform);
        });

        // Test two points
        assertThrows(IllegalArgumentException.class, () -> {
            new Retangulo(new Ponto[]{new Ponto(0.0, 0.0), new Ponto(1.0, 1.0)}, transform);
        });

        // Test three points
        assertThrows(IllegalArgumentException.class, () -> {
            new Retangulo(new Ponto[]{new Ponto(0.0, 0.0), new Ponto(1.0, 0.0), new Ponto(1.0, 1.0)},
transform);
        });
    }
```

```java
@Test
    void testPerimetro() {
        Ponto p1 = new Ponto(0.0, 0.0);
        Ponto p2 = new Ponto(0.0, 2.0);
        Ponto p3 = new Ponto(3.0, 2.0);
        Ponto p4 = new Ponto(3.0, 0.0);
        Ponto[] pontos = {p1, p2, p3, p4};
        Retangulo ret = new Retangulo(pontos, transform);
        assertEquals(10.0, ret.perimetro());
    }

    @Test
    void testCentro() {
        Ponto p1 = new Ponto(0.0, 0.0);
        Ponto p2 = new Ponto(0.0, 2.0);
        Ponto p3 = new Ponto(3.0, 2.0);
        Ponto p4 = new Ponto(3.0, 0.0);
        Ponto[] pontos = {p1, p2, p3, p4};
        Retangulo ret = new Retangulo(pontos, transform);
        Ponto centro = ret.centro();
        assertEquals(1.5, centro.x());
        assertEquals(1.0, centro.y());
    }
}
```

```java
package test;
import static org.junit.Assert.*;

    import game.Transform;
    import org.junit.jupiter.api.Test;
    import geometry.*;

    public class CirculoTest {

        @Test
        void constructorWithValidParameters() {
            Transform transform = new Transform(new Ponto(0.0, 0), 1, 0, 1);
            Circulo circulo = new Circulo(0, 0, 5, transform);
            assertNotNull(circulo);
        }

        @Test
        void constructorWithInvalidRadius() {
            Transform transform = new Transform(new Ponto(0.0, 0), 1, 0, 1);
            assertThrows(IllegalArgumentException.class, () -> new Circulo(0, 0, -5, transform));
        }

        @Test
        void updatePosicao() {
            Transform transform = new Transform(new Ponto(1.0, 1), 1, 0, 1);
            Circulo circulo = new Circulo(0, 0, 5, transform);
            circulo.updatePosicao();
            assertEquals(new Ponto(1, 1), circulo.centro());
        }

        @Test
        void updateEscalar() {
            Transform transform = new Transform(new Ponto(0.0, 0), 2, 0, 2);
            Circulo circulo = new Circulo(0, 0, 5, transform);
            circulo.updateEscalar();
            assertEquals(10, circulo.r());
        }

        @Test
        void perimetro() {
            Transform transform = new Transform(new Ponto(0.0, 0), 1, 0, 1);
            Circulo circulo = new Circulo(0, 0, 5, transform);
            assertEquals(2 * Math.PI * 5, circulo.perimetro());
        }

        @Test
        void translacao() {
            Transform transform = new Transform(new Ponto(0.0, 0), 1, 0, 1);
            Circulo circulo = new Circulo(0, 0, 5, transform);
            Circulo novoCirculo = circulo.translacao(new Ponto(2, 2));
            assertEquals(new Ponto(2, 2), novoCirculo.centro());
        }

        @Test
        void distanciaAoCentro() {
            Transform transform = new Transform(new Ponto(0.0, 0), 1, 0, 1);
            Circulo circulo = new Circulo(0, 0, 5, transform);
            assertEquals(5, circulo.distanciaAoCentro(new Ponto(5, 0)));
        }

        @Test
        void intersectaCirculo() {
            Transform transform = new Transform(new Ponto(0.0, 0), 1, 0, 1);
            Circulo circulo1 = new Circulo(0, 0, 5, transform);
            Circulo circulo2 = new Circulo(7, 0, 5, transform);
            assertTrue(circulo1.intersecta(circulo2));
        }

        @Test
        void naoIntersectaCirculo() {
            Transform transform = new Transform(new Ponto(0.0, 0), 1, 0, 1);
            Circulo circulo1 = new Circulo(0, 0, 5, transform);
            Circulo circulo2 = new Circulo(11, 0, 5, transform);
            assertFalse(circulo1.intersecta(circulo2));
        }
```

```java
@Test
        void setRWithValidRadius() {
            Transform transform = new Transform(new Ponto(0.0, 0), 1, 0, 1);
            Circulo circulo = new Circulo(0, 0, 5, transform);
            circulo.setR(10);
            assertEquals(10, circulo.r());
        }

        @Test
        void setRWithInvalidRadius() {
            Transform transform = new Transform(new Ponto(0.0, 0), 1, 0, 1);
            Circulo circulo = new Circulo(0, 0, 5, transform);
            assertThrows(IllegalArgumentException.class, () -> circulo.setR(-10));
        }

        @Test
        void isInsidePolygon() {
            Transform transform = new Transform(new Ponto(0.0, 0), 1, 0, 1);
            Circulo circulo = new Circulo(0, 0, 5, transform);
            Poligono poligono = new Poligono(new Ponto[]{new Ponto(1.0, 1), new Ponto(2.0, 2), new
Ponto(1.0, 3)},transform);
            assertFalse(circulo.isInside(poligono));
        }

        @Test
        void intersectaPolygon()
        {
            Transform transform = new Transform(new Ponto(0.0, 0), 1, 0, 1);
            Circulo circulo = new Circulo(0, 0, 5, transform);
            Poligono poligono = new Poligono(new Ponto[]{new Ponto(1.0, 1), new Ponto(2.0, 2), new
Ponto(1.0, 3)},transform);
            assertTrue(circulo.intersecta(poligono));
        }

        @Test
        void toStringTest() {
            Transform transform = new Transform(new Ponto(0, 0), 1, 0, 1);
            Circulo circulo = new Circulo(0, 0, 5, transform);
            assertEquals("0.00 0.00) 5.00", circulo.toString());
        }
    }
```