Plan de Pruebas - Simulador de Física

Juan Carlos Gómez Hernández[1]

[1] Universidad Linda Vista, Ex-Finca Santa Cruz No. 1, 29750, México correo 1@ulv.edu.mx
[2] Springer Heidelberg, Tiergartenstr. 17, 69121 Heidelberg, Germany lncs@springer.com
[3] ABC Institute, Rupert-Karls-University Heidelberg, Heidelberg, Germany {abc,lncs}@uni-heidelberg.de

1. Introducción.

1.1. Propósito.

El propósito de este documento es proporcionar un plan de pruebas detallado para el simulador interactivo de física desarrollado en Pygame. Este plan establece una guía sistemática para validar la funcionalidad, precisión y estabilidad del sistema, incluyendo la interacción de los usuarios con la interfaz y el comportamiento del simulador en diferentes escenarios.

1.2. Descripción General del Proyecto.

Simulador interactivo desarrollado en Pygame que modela el equilibrio de fuerzas en un sistema de poleas, permitiendo la manipulación de peso, ángulos y visualización de tensiones.

2. Alcance.

2.1. Dentro del Alcance

- Cálculos matemáticos de tensiones
- Conversión de unidades (kg a N)
- Renderizado de escena
- Interacción del usuario con los elementos de la interfaz
- Manejo de diferentes escenarios de ángulos y pesos

2.2. Fuera del Alcance.

- Pruebas de rendimiento exhaustivas
- Pruebas de compatibilidad con múltiples sistemas operativos
- Pruebas de seguridad avanzadas

3. Estrategia de Prueba.

3.1. Objetivos de la Prueba.

- Validar la precisión de los cálculos matemáticos
- Verificar la funcionalidad de la interfaz de usuario
- Garantizar la estabilidad del renderizado
- Comprobar la conversión correcta de unidades

3.2. Supuestos de Prueba.

- Entorno de desarrollo con Python 3.8+ instalado
- Bibliotecas Pygame, Matplotlib y Pytest
- Acceso al código fuente completo

3.3. Enfoque de Datos.

- Utilizar conjuntos de datos predefinidos para pruebas
- Incluir casos extremos (ángulos 0°, 90°, pesos diferentes)
- Generar datos aleatorios para pruebas de robustez

3.4. Nivel de Prueba.

- Pruebas unitarias
- Pruebas funcionales
- Pruebas de integración básicas

3.5. Pruebas Unitarias.

- Probar funciones individuales de cálculo
- Verificar conversión de unidades
- Comprobar cálculo de tensiones
- Validar posicionamiento de objetos

Script de las pruebas unitarias.

```python
import pytest
import math
from Static_Balance import solution, calculate_tensions, calculate_body_position, conversor

# 3.5.1 Probar funciones individuales de c lculo
@pytest.mark.parametrize("weight, theta1, theta2, expected", [
    (100, 45, 45, (70.71, 70.71)),  # Correcto
    (50, 30, 60, (25, 43.3)),       # Corrige el orden
])
def test_solution(weight, theta1, theta2, expected):
    T1, T2 = solution(weight, theta1, theta2)
    assert math.isclose(T1, expected[0], rel_tol=1e-2)
    assert math.isclose(T2, expected[1], rel_tol=1e-2)

# 3.5.2 Verificar conversi n de unidades
@pytest.mark.parametrize("Kg, expected", [
    (10, 98.1), (0, 0), (1, 9.81),
])
def test_conversor(Kg, expected):
    assert math.isclose(conversor(Kg), expected, rel_tol=1e-2)

# 3.5.3 Comprobar c lculo de tensiones
@pytest.mark.parametrize("weight, theta1, theta2, expected", [
    (100, 45, 45, (70.71, 70.71)),  # Resultado esperado basado en T1 y T2
    (50, 30, 60, (25, 43.3)),       # Resultado esperado de acuerdo a la l gica de 'calculate_tensions'
])
def test_calculate_tensions(weight, theta1, theta2, expected):
    T1, T2 = calculate_tensions(weight, theta1, theta2)
    assert math.isclose(T1, expected[1], rel_tol=1e-2)
    assert math.isclose(T2, expected[0], rel_tol=1e-2)


# 3.5.4 Validar posicionamiento de objetos
def test_calculate_body_position():
    anchor1_x, anchor2_x, anchor_y = 0, 10, 10
    T1, T2 = 70.71, 70.71
    theta1, theta2 = 45, 45

    body_x, body_y = calculate_body_position(anchor1_x, anchor2_x, anchor_y, T1, T2, theta1, theta2)
    assert body_x > 0
    assert body_y > 0
```

Figura 1. Resultados de las pruebas unitarias.

3.6. Pruebas Funcionales.

- Probar interacción con botones
- Verificar renderizado de escena
- Comprobar generación de gráficos
- Validar cambios de estado de la interfaz

Script de las pruebas de funciones.

```
import pygame
import pytest
from Static_Balance import draw_scene, conversor,
    crear_grafico

# Prueba de inicializaci n de Pygame
def test_pygame_initialization():
    pygame.init()
    screen = pygame.display.set_mode((1350, 840))
    assert screen is not None, "No se pudo inicializar la
        ventana de Pygame"
    pygame.quit()

# Prueba de renderizado de escena
def test_scene_render():
    pygame.init()
    screen = pygame.Surface((1350, 840))  # Superficie para
        pruebas
    draw_scene(screen, weight=100, theta1=45, theta2=45,
        result_conversion=0, conversor_visible=False)
    # Actualizar el color esperado
    assert screen.get_at((10, 10)) == (0, 71, 125, 255), "El
        fondo no se renderiz  correctamente"
    pygame.quit()


# Prueba del conversor
```

```python
def test_conversion_kg_to_n():
    result = conversor(10)  # Convertir 10 kg a N
    assert result == pytest.approx(98.1, rel=1e-3), f"Conversi n incorrecta, esperado 98.1, obtenido {result}"


def test_graph_generation():
    peso = 100
    theta1 = 45
    theta2 = 45
    T11, T22 = 70.71, 70.71  # Valores esperados para estas condiciones
    grafico = crear_grafico(peso, theta1, theta2, T11, T22)

    assert grafico.get_width() > 0 and grafico.get_height() > 0, "El gr fico no se gener  correctamente"
```

```
pages/prb.py::test_pygame_initialization PASSED                          [ 25%]
pages/prb.py::test_scene_render PASSED                                   [ 50%]
pages/prb.py::test_conversion_kg_to_n PASSED                             [ 75%]
pages/prb.py::test_graph_generation PASSED                               [100%]
```

Figura 2. Resultados de las pruebas de funcionalidad.

4. Estrategia de Ejecución.

4.1. Criterios de Entrada.

- Código fuente completamente implementado
- Entorno de desarrollo configurado
- Dependencias instaladas
- Casos de prueba diseñados

4.2. Criterios de Salida.

- Buena aceptación de scripts de prueba ejecutados
- Todos los resultados documentados

4.3. Validación y Gestión de Defectos.

Clasificación de defectos por gravedad:

- Crítico: Bloquea completamente la funcionalidad
- Alto: Impide uso principal
- Medio: Funcionalidad parcialmente afectada
- Bajo: Problemas cosméticos o menores

5. Requisitos Ambientales.

5.1. Entornos de Prueba.

- Python 3 o superior
- Pygame
- Matplotlib
- Pytest
- Sistema operativo: Windows

Referencias

1. Python Software Foundation.: Pytest. Python Software Foundation (2024).