Test Summary Report
Pixel Arena

Riferimento	
Versione	0.2
Data	18/01/2024
Destinatario	Carmine Gravino
Presentato da	NC10: Antonio Ferrentino, Emanuele Rosapepe, Francesco Perilli
Approvato da	

Sommario

Team Composition

Nome	Matricola	Contatti
Antonio Ferrentino	0512113367	a.ferrentino50@studenti.unisa.it
Emanuele Rosapepe	0512113418	e.rosapepe2@studenti.unisa.it
Francesco Perilli	0512113802	f.perilli2@studenti.unisa.it

Revision History

Data	Versione	Descrizione	Autori
17/01/2024	0.1	Prima stesura	Antonio Ferrentino Emanuele Rosapepe Francesco Perilli
18/01/2024	0.2	Completamento e revisione	Antonio Ferrentino Emanuele Rosapepe Francesco Perilli

1. Introduzione

1.1 Scopo del Sistema

Nel corso degli anni l'evoluzione del mondo videoludico è stata sia veloce che impressionante, spalancando le porte ad un'era d'oro per i videogiochi e per le aziende che ne trattano la loro creazione. La nostra idea è quella di portare alla luce un platform ortogonale con annessi combattimenti, il tutto rappresentato da una grafica pixelata: ciò potrebbe far sembrare il nostro gioco non molto dinamico rispetto ad altri giochi che si adattano perfettamente agli sviluppi sia hardware che software a cui abbiamo assistito in questi anni, unendo una grafica mozzafiato ad un'accuratezza nel gameplay. Ma per le dinamiche e per le sfide che il nostro gioco "Pixel Arena" ha intenzione di proporre, metterà sicuramente alla prova le abilità del videogiocatore, non negandogli anche il giusto divertimento.

1.2 Scopo del documento

Il seguente documento riporta e descrive le attività di Testing effettuate per garantire il corretto funzionamento del Sistema.
All'interno del documento saranno riportate le funzionalità testate, con i relativi strumenti e strategie utilizzati.

Le funzionalità testate saranno le seguenti:

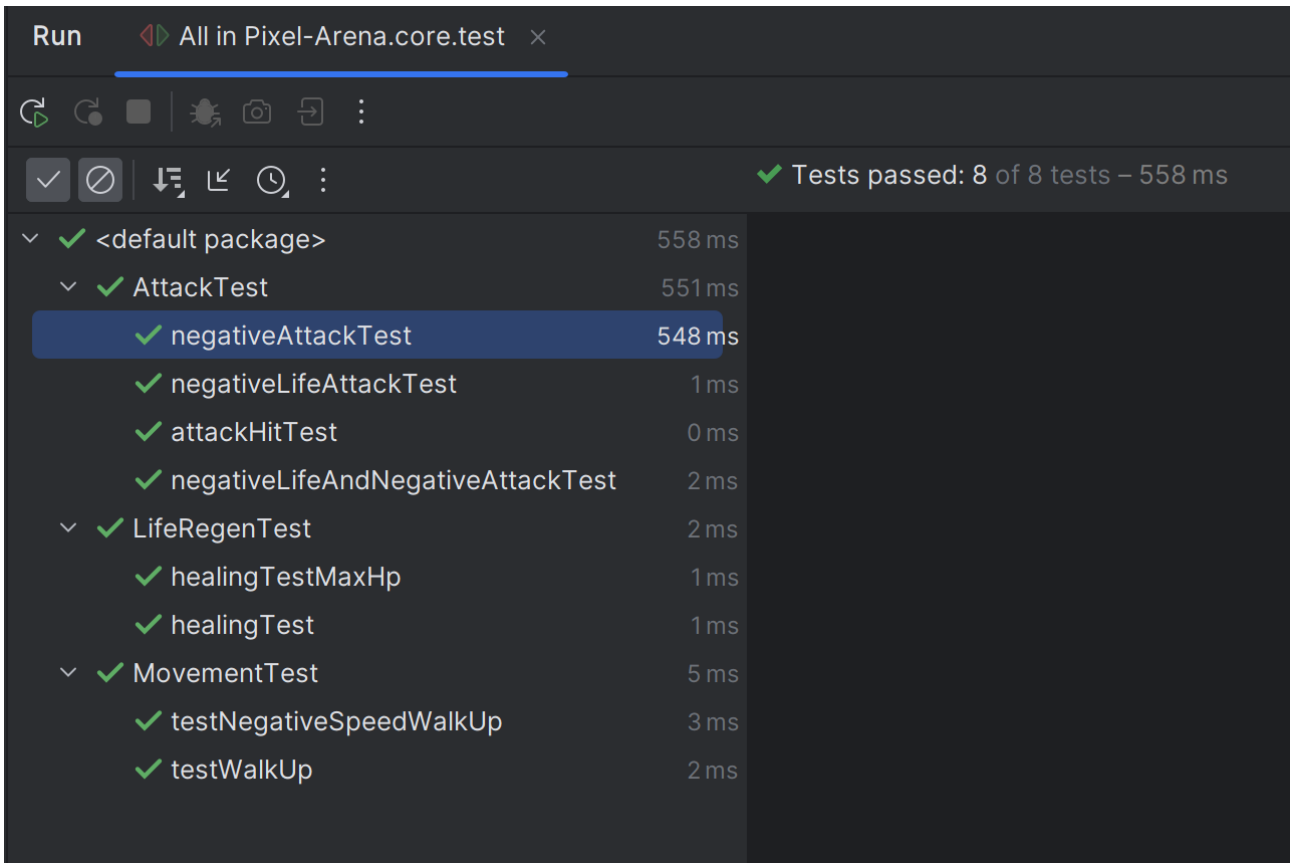
- Attacco
- Movimento
- Rigenerazione Vita

1.3 Riferimenti

- Test Plan (TP)
- Test Case Specification (TCS)
- Test Incident Report (TIR)

2. Testing di unità

I test di unità sono stati svolti al termine dell'implementazione dai Team Member, utilizzando le librerie JUnit e Mockito. In caso di errori, lo sviluppatore ha provveduto alla correzione necessaria.




```java
import ...

public class MovementTest {
    19 usages
    private Character mainCharacter;

    @Before
    public void setUp(){
        mainCharacter = Mockito.spy(Character.class);
ape\StudioProjects\Pixel-Arena\core\src\test

    @Test
    public void testWalkUp(){
        mainCharacter.setPosition( x: 0, y: 0);
        mainCharacter.setSpeed(1f);
        mainCharacter.setDirection(Character.Directions.NORTH);
        mainCharacter.doWalk();
        Assert.assertEquals( expected: 0, mainCharacter.getX(), delta: 0 - mainCharacter.getX());
        Assert.assertEquals( expected: mainCharacter.getY() + mainCharacter.getStandardSpeed(), mainCharacter.getY(), mainCharacter.getStandardSpeed() );
    }

    @Test
    public void testNegativeSpeedWalkUp(){
        mainCharacter.setPosition( x: 0, y: 0);
        mainCharacter.setSpeed(-1f);
        mainCharacter.setDirection(Character.Directions.NORTH);
        mainCharacter.doWalk();
        Assert.assertEquals( expected: 0, mainCharacter.getX(), delta: 0 - mainCharacter.getX());
        Assert.assertEquals( expected: 0, mainCharacter.getY(), mainCharacter.getStandardSpeed() );
    }

}
```

```java
public class LifeRegenTest {
    10 usages
    private Character mainCharacter;

    @Before
    public void setUp(){
        mainCharacter = Mockito.spy(Character.class);
    }

    @Test
    public void healingTest(){
        mainCharacter.setHealthPoints(42);
        mainCharacter.doHeal( stateTime: 0);
        Assert.assertEquals(mainCharacter.getHealthPoints(),  actual: 52);
    }

    @Test
    public void healingTestMaxHp(){
        mainCharacter.setMaxHealthPoints(100);
        mainCharacter.setHealthPoints(mainCharacter.getMaxHealthPoints());
        mainCharacter.doHeal( stateTime: 0);
        Assert.assertEquals(mainCharacter.getHealthPoints(), mainCharacter.getMaxHealthPoints());
    }
}
```

```java
public class AttackTest {
    9 usages
    private Character mainCharacter;
    18 usages
    private Character villain;

    @Before
    public void setUp(){
        mainCharacter = Mockito.spy(Character.class);
        villain = Mockito.spy(Character.class);
    }

    @Test
    public void attackHitTest(){
        mainCharacter.setAttackPower(20);
        villain.setMaxHealthPoints(100);
        villain.setHealthPoints(100);
        villain.decreaseHealth(mainCharacter.getAttackPower());
        Assert.assertEquals(villain.getHealthPoints(),  actual: 80);
    }
```

```
26      @Test
27      public void negativeAttackTest(){
28          mainCharacter.setAttackPower(-1);
29          villain.setMaxHealthPoints(100);
30          villain.setHealthPoints(100);
31          villain.decreaseHealth(mainCharacter.getAttackPower());
32          Assert.assertEquals(villain.getHealthPoints(), villain.getMaxHealthPoints());
33      }
34
35      @Test
36      public void negativeLifeAttackTest(){
37          mainCharacter.setAttackPower(20);
38          villain.setMaxHealthPoints(100);
39          villain.setHealthPoints(-1);
40          villain.decreaseHealth(mainCharacter.getAttackPower());
41          Assert.assertEquals(villain.getHealthPoints(),  actual: 0);
42      }
43
44      @Test
45      public void negativeLifeAndNegativeAttackTest(){
46          mainCharacter.setAttackPower(-20);
47          villain.setMaxHealthPoints(100);
48          villain.setHealthPoints(-1);
49          villain.decreaseHealth(mainCharacter.getAttackPower());
50          Assert.assertEquals(villain.getHealthPoints(),  actual: 0);
51      }
52  }
```

3. Testing di sistema

Il testing di sistema è stato effettuato utilizzando manualmente l'applicazione, testandone ogni funzionalità attraverso tutta l'implementazione dell'intero sistema. Ogni qualvolta veniva aggiunta una singola funzionalità questa veniva prima testata, poi si poteva procedere con l'implementazione di un'altra funzionalità.