Team 5 – Tests

Boat Test Class

```
package com.introduction.rowing;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertTrue;

public class BoatTest {

    private Boat boat;
    private GameInputProcessor inputProcessor;
    private ShopBoat shopBoat;

    @BeforeEach
    public void setUp() {
        // Set up necessary objects for the test
        Position position = new Position(0, 0);
        MyRowing myRowing = new MyRowing(); // Make sure you have appropriate constructor or mock
        inputProcessor = new GameInputProcessor(myRowing);
        shopBoat = new ShopBoat(1, "Standard Boat", 1000, "boat.png", 10, 10, 10, 10, 10, 10, true, false);
        boat = new Boat(1, position, true, inputProcessor, shopBoat);

    }

    @Test
    public void testBoatInitialization() {
        // Verify initialization of Boat attributes
        assertEquals(1, boat.getId());
        assertEquals(10, boat.getSpeedFactor());
        assertEquals(10, boat.getAcceleration());
        assertEquals(10, boat.getRobustness());
        assertEquals(10, boat.getMomentumFactor());
        assertEquals(10, boat.getFatigue());
    }

    @Test
    public void testUpdateKeysMovingUp() {
        inputProcessor.moving = true;
        inputProcessor.direction = 0;
        boat.setIsAcceleratorAvailable(true);

        float initialY = boat.getPosition().getY();
        boat.updateKeys(1.0f, 0);

        assertTrue(boat.getPosition().getY() > initialY);
    } }
```

CatPowerUp Test Class

```
package com.introduction.rowing;

import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertNotNull;
import static org.mockito.Mockito.mock;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import com.badlogic.gdx.graphics.Texture;

public class CatPowerupTest {

    private MyRowing mockedMyRowing;
    private CatPowerup catPowerup;

    @BeforeEach
    public void setUp() {
        // Mock MyRowing
        mockedMyRowing = mock(MyRowing.class);
        catPowerup = new CatPowerup(mockedMyRowing);
    }

    @Test
    public void testUse() {
        // Verify that use() method correctly sets invulnerability time of player boat
        catPowerup.use();
        assertEquals(5, mockedMyRowing.getPlayerBoat().getInvulnerabilityTime());
    }

    @Test
    public void testGetTexture() {
        // Verify that a non-null texture is returned
        Texture texture = catPowerup.getTexture();
        assertNotNull(texture);
    }
}
```

CoockiePowerUp Test Class

```
package com.introduction.rowing;

import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertNotNull;
import static org.mockito.Mockito.mock;

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

import com.badlogic.gdx.graphics.Texture;

public class CookiePowerupTest {

    private MyRowing mockedMyRowing;
    private CookiePowerup cookiePowerup;
    @BeforeEach
    public void setUp() {
        // Mock MyRowing
        mockedMyRowing = mock(MyRowing.class);
        cookiePowerup = new CookiePowerup(mockedMyRowing);
    }

    @Test
    public void testUse() {
        // Verify that use() method correctly increases boat health
        Boat playerBoat = mockedMyRowing.getPlayerBoat();
        int initialHealth = playerBoat.getBoatHealth();
        cookiePowerup.use();
        assertEquals(initialHealth + 25, playerBoat.getBoatHealth());
    }

    @Test
    public void testGetName() {
        // Verify that correct name is returned
        assertEquals("Fortune Cookie", cookiePowerup.getName());
    }

    @Test
    public void testGetPrice() {
        // Verify that correct price is returned
        assertEquals(150, cookiePowerup.getPrice());
    }
}
```

Datamanagert Test Class

```
package com.introduction.rowing;

import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertNotNull;
import static org.mockito.Mockito.mock;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.files.FileHandle;

public class DataManagerTest {

    private DataManager dataManager;

    @BeforeEach
    public void setUp() {
        dataManager = new DataManager();
    }

    @Test
    public void testReadBalance() {
        // Verify that balance is correctly read from the file
        int expectedBalance = 0;
        FileHandle fileHandle = Gdx.files.local(DataManager.MONEY_BALANCE_FILE_PATH);
        if (fileHandle.exists()) {
            expectedBalance = Integer.parseInt(fileHandle.readString());
        }
        assertEquals(expectedBalance, dataManager.getBalance());
    }
}
```

Datamanagert Test Class

```
package com.introduction.rowing;

import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertNotNull;
import static org.mockito.Mockito.mock;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.files.FileHandle;

public class DataManagerTest {

    private DataManager dataManager;

    @BeforeEach
    public void setUp() {
        dataManager = new DataManager();
    }

    @Test
    public void testReadBalance() {
        // Verify that balance is correctly read from the file
        int expectedBalance = 0;
        FileHandle fileHandle = Gdx.files.local(DataManager.MONEY_BALANCE_FILE_PATH);
        if (fileHandle.exists()) {
            expectedBalance = Integer.parseInt(fileHandle.readString());
        }
        assertEquals(expectedBalance, dataManager.getBalance());
    }
}
```

Dragonhead Test Class

```
package com.introduction.rowing;

import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.mockito.Mockito.mock;

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

import com.badlogic.gdx.graphics.Texture;

public class DragonHeadTest {

    private DragonHead dragonHead;
    private MiniGameInputProcessor mockedInputProcessor;

    @BeforeEach
    public void setUp() {
        Position position = new Position(0, 0);
        Texture texture = mock(Texture.class);
        mockedInputProcessor = mock(MiniGameInputProcessor.class);
        dragonHead = new DragonHead(position, 50, 50, texture, mockedInputProcessor);
    }

    @Test
    public void testUpdateKeysMovingUp() {
        // Verify that the position updates correctly when moving up
        mockedInputProcessor.moving = true;
        mockedInputProcessor.direction = 0;
        float initialY = dragonHead.getPosition().getY();
        dragonHead.updateKeys(1.0f);
        float updatedY = dragonHead.getPosition().getY();
        assertEquals(Math.max(0, initialY + (250 * 1.0f)), updatedY);
    }


    @Test
    public void testUpdateKeysMovingLeft() {
        // Verify that the position updates correctly when moving left
        mockedInputProcessor.moving = true;
        mockedInputProcessor.direction = 1;
        float initialX = dragonHead.getPosition().getX();
        dragonHead.updateKeys(1.0f);
        float updatedX = dragonHead.getPosition().getX();
        assertEquals(Math.max(0, initialX - (250 * 1.0f)), updatedX);
    }

    @Test
    public void testUpdateKeysMovingDown() {
        // Verify that the position updates correctly when moving down
        mockedInputProcessor.moving = true;
        mockedInputProcessor.direction = 2;
        float initialY = dragonHead.getPosition().getY();
        dragonHead.updateKeys(1.0f);
        float updatedY = dragonHead.getPosition().getY();
        assertEquals(Math.max(0, initialY - (250 * 1.0f)), updatedY);
    }

    @Test
    public void testUpdateKeysMovingRight() {
        // Verify that the position updates correctly when moving right
        mockedInputProcessor.moving = true;
        mockedInputProcessor.direction = 3;
        float initialX = dragonHead.getPosition().getX();
        dragonHead.updateKeys(1.0f);
        float updatedX = dragonHead.getPosition().getX();
        assertEquals(Math.max(0, initialX + (250 * 1.0f)), updatedX);
    }
}
```

Entity Test Class

```
package com.introduction.rowing;
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertNotNull;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

import com.badlogic.gdx.graphics.Texture;

public class EntityTest {

    private Entity entity;
    private Position position;
    private Texture texture;

    @BeforeEach
    public void setUp() {
        position = new Position(0, 0);
        texture = new Texture("test_texture.png");
        entity = new Entity(position, 50, 50, texture);
    }

    @Test
    public void testGetPosition() {
        // Verify that the position of the entity is returned correctly
        assertEquals(position, entity.getPosition());
    }
    @Test
    public void testGetWidth() {
        // Verify that the width of the entity is returned correctly
        assertEquals(50, entity.getWidth());
    }

    @Test
    public void testGetHeight() {
        // Verify that the height of the entity is returned correctly
        assertEquals(50, entity.getHeight());
    }

    @Test
    public void testSetPosition() {
        // Verify that the position of the entity is set correctly
        int newX = 100;
        int newY = 200;
        entity.setPosition(newX, newY);
        assertEquals(newX, entity.getPosition().getX());
        assertEquals(newY, entity.getPosition().getY()); }}
```

Input Processor Test Class

```
package com.introduction.rowing;

import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertNotEquals;

import org.junit.Before;
import org.junit.Test;

public class InputProcessorTest {

    private InputProcessor inputProcessor;
    private MyRowing myRowing;

    @Before
    public void setUp() {
        myRowing = new MyRowing();
        inputProcessor = new InputProcessor(myRowing);
    }

    @Test
    public void testSetGameState() {
        InputProcessor.setGameState(GameState.RACING);
        assertEquals(GameState.RACING, InputProcessor.getGameState());

        InputProcessor.setGameState(GameState.LOBBY);
        assertEquals(GameState.LOBBY, InputProcessor.getGameState());
    }

    @Test
    public void testSetGameSubState() {
        InputProcessor.setGameSubState(GameSubState.RACE_END);
        assertEquals(GameSubState.RACE_END, InputProcessor.getGameSubState());

        InputProcessor.setGameSubState(GameSubState.RACE_LEG);
        assertEquals(GameSubState.RACE_LEG, InputProcessor.getGameSubState());
    }

    @Test
    public void testSwitchGameSubState() {
        InputProcessor.switchGameSubState();
        assertEquals(ShopSubState.POWERUPS, InputProcessor.getShopSubStates());

        InputProcessor.switchGameSubState();
        assertEquals(ShopSubState.BOATS, InputProcessor.getShopSubStates());
    }

    @Test
    public void testInitialStates() {
        assertEquals(GameState.LOBBY, InputProcessor.getGameState());
        assertEquals(GameSubState.RACE_LEG, InputProcessor.getGameSubState());
        assertEquals(ShopSubState.BOATS, InputProcessor.getShopSubStates());
    }

    @Test
    public void testChangeGameStateAndSubState() {
        InputProcessor.setGameState(GameState.RACING);
        InputProcessor.setGameSubState(GameSubState.RACE_END);

        assertNotEquals(GameState.LOBBY, InputProcessor.getGameState());
        assertNotEquals(GameSubState.RACE_LEG, InputProcessor.getGameSubState());

        assertEquals(GameState.RACING, InputProcessor.getGameState());
        assertEquals(GameSubState.RACE_END, InputProcessor.getGameSubState());
    }
}
```

Lane Test Class

```java
package com.introduction.rowing;

import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertFalse;
import static org.junit.jupiter.api.Assertions.assertTrue;
import static org.mockito.Mockito.mock;

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

import com.badlogic.gdx.graphics.Texture;

public class LaneTest {

    private Lane lane;
    private Boat boat;

    @BeforeEach
    public void setUp() {
        boat = mock(Boat.class);
        lane = new Lane(boat, 100);
    }

    @Test
    public void testGetBoat() {
        // Verify that the correct boat is returned
        assertEquals(boat, lane.getBoat());
    }

    @Test
    public void testSpawnObstacleReady() {
        // Verify that spawn obstacle ready status is true after a certain time
        assertTrue(lane.spawnObstacleReady(0.9f));
    }



    @Test
    public void testSpawnObstacles() {
        // Verify that obstacles are added to the lane
        lane.spawnObstacles();
        assertEquals(1, lane.getObstacles().size());
    }

}
```

Fish Power Up Test Class

```java
package com.introduction.rowing;

import static org.junit.Assert.assertEquals;
import static org.mockito.Mockito.*;

import com.badlogic.gdx.graphics.Texture;
import org.junit.Before;
import org.junit.Test;
import org.mockito.Mockito;

public class FishPowerupTest {
    private FishPowerup fishPowerup;
    private MyRowing myRowing;

    @Before
    public void setUp() {
        myRowing = Mockito.mock(MyRowing.class);
        fishPowerup = new FishPowerup(myRowing);
    }

    @Test
    public void testUse() {
        fishPowerup.use();
        verify(myRowing).deleteAllPlayerObstacles();
    }

    @Test
    public void testGetDescription() {
        assertEquals("All obstacles in your lane \n disappear", fishPowerup.getDescription());
    }

    @Test
    public void testGetName() {
        assertEquals("Koi", fishPowerup.getName());
    }

    @Test
    public void testGetPrice() {
        assertEquals(200, fishPowerup.getPrice());
    }

    @Test
    public void testGetTexture() {
        Texture texture = fishPowerup.getTexture();
        assertEquals("powerups/koi.png", texture.toString());
    }
```

Mini Game Input Processor Test class

```
import org.junit.Before;
import org.junit.Test;
import org.mockito.Mockito;

import static org.junit.Assert.*;
import static org.mockito.Mockito.*;

public class MiniGameInputProcessorTest {

    private MiniGameInputProcessor miniGameInputProcessor;
    private MyRowing myRowing;

    @Before
    public void setUp() {
        myRowing = Mockito.mock(MyRowing.class);
        miniGameInputProcessor = new MiniGameInputProcessor(myRowing);
    }

    @Test
    public void testKeyDownUp() {
        miniGameInputProcessor.keyDown(Input.Keys.UP);
        assertTrue(miniGameInputProcessor.moving);
        assertEquals(0, miniGameInputProcessor.direction);
        miniGameInputProcessor.keyUp(Input.Keys.UP);
        assertFalse(miniGameInputProcessor.moving);
    }

    @Test
    public void testKeyDownLeft() {
        miniGameInputProcessor.keyDown(Input.Keys.LEFT);
        assertTrue(miniGameInputProcessor.moving);
        assertEquals(1, miniGameInputProcessor.direction);

        miniGameInputProcessor.keyUp(Input.Keys.LEFT);
        assertFalse(miniGameInputProcessor.moving);
    }

    @Test
    public void testKeyDownDown() {
        miniGameInputProcessor.keyDown(Input.Keys.DOWN);
        assertTrue(miniGameInputProcessor.moving);
        assertEquals(2, miniGameInputProcessor.direction);

        miniGameInputProcessor.keyUp(Input.Keys.DOWN);
        assertFalse(miniGameInputProcessor.moving);
    }

    @Test
    public void testKeyDownRight() {
        miniGameInputProcessor.keyDown(Input.Keys.RIGHT);
        assertTrue(miniGameInputProcessor.moving);
        assertEquals(3, miniGameInputProcessor.direction);

        miniGameInputProcessor.keyUp(Input.Keys.RIGHT);
        assertFalse(miniGameInputProcessor.moving);
    }

    @Test
    public void testKeyDownEscape() {
        miniGameInputProcessor.keyDown(Input.Keys.ESCAPE);
        verify(myRowing, times(1)).resetMiniGame();
        assertEquals(MiniGameState.NOT_STARTED, myRowing.miniGameState);
    }

    @Test
    public void testKeyDownEnterInSumScreen() {
        when(myRowing.miniGameState).thenReturn(MiniGameState.SUM_SCREEN);
        miniGameInputProcessor.keyDown(Input.Keys.ENTER);
        verify(myRowing, times(1)).resetMiniGame();
        assertEquals(MiniGameState.NOT_STARTED, myRowing.miniGameState);
    }
}
```