1. TileTest

- **Coverage**: Achieved **100% test coverage**, focusing on the single responsibility of the Tile class.
- **Focus**: Verified the initialization of the Tile class to ensure it starts in a valid default state with no predefined image.
- **Tests**:
    - Testdefaulttile: Ensured that the Tile instance's image attribute is null upon creation.
- **Approach**: The simplicity of the Tile class made achieving full coverage straightforward, as its functionality is limited to initialization.

- **Result:** All the tests passed with full accuracy and can be concluded that the UI rendering has been done correctly.

2. TerrainHandlerTest

- **Coverage**: Achieved **90% test coverage**, thoroughly testing the logic related to terrain handling, tile management, and rendering.
- **Focus**: Covered a wide range of features:
    - Initialization and size verification of terrainTiles and tileMap.
    - Loading and verifying individual tile images for game components like grass, roads, rewards (e.g., diamonds, nitro), and hurdles (e.g., spikes, potholes).
    - Rendering of game elements, including cops, the thief, and tiles at specific locations.
    - Correct mapping of active rewards and hurdles to the appropriate positions on the tileMap.
- **Tests**:
    - Initialization: terraintilesInitialization, tilemapinitialization, tilemapsizetest.
    - Image Loading: grasstest, roadtest, diamondtest, spiketest, etc.
    - Rendering: testCop1Render, testThiefRender, testCorrectTileImageIsRenderedAtLocation.
    - Dynamic Mapping: testTileMapInitializedWithNitro, testTileMapInitializedWithPothole, etc.
- **Approach**: Used Mockito to mock dependencies like Map, Graphics2D, and various game items. This isolation allowed testing of the TerrainHandler logic independently of other components.
- **Coverage Gaps**: Remaining 10% is due to runtime-dependent rendering logic that is impractical to simulate in tests and is better validated during actual gameplay.

- **Result:** All the tests passed with full accuracy and can be concluded that the UI rendering has been done correctly

3. GamePanelTest

- **Coverage**: Achieved **52% test coverage**, focusing on critical UI features while acknowledging the limitations of testing visual elements programmatically.
- **Focus**: Validated essential GamePanel functionalities:
  - Time formatting logic for displaying the timer during gameplay.
  - Updates to collectible displays for diamonds and nitro.
  - Creation and properties of control labels like PLAY and EXIT.
- **Tests**:
  - Timer: testFormatTime tested various edge cases, including zero seconds, full hours, and mixed durations.
  - Display Updates: testUpdateCollectiblesDisplay ensured UI components for collectibles are initialized and accessible.
  - Control Labels: testCreateGameControlLabels verified the text and properties of PLAY and EXIT labels.
- **Approach**: Mocked the Map dependency and used component-based assertions to test UI elements effectively.
- **Coverage Gaps**: The uncovered 48% involves animations and dynamic rendering of game components during runtime, which are better validated through manual testing as they are highly visual and context-dependent.
- **Result:** All the tests passed with full accuracy and can be concluded that the gamepannel has all the necessary elements to it

4. MapTest

- **Coverage**: Achieved 74% branch test coverage and 77%-line coverage, primarily focusing on thief and cop movement on map. All critical paths of the thief/cop's movement, reward collection, and hurdle penalties were tested, as well as edge cases like overlapping items or invalid positions.
- **Focus**:
  Verified correct movement of thief:
  - Thief's movement constraints (restricted to roads and avoiding gardens/buildings).
  - Proper handling of rewards: ensuring they can only be collected once, and inactive rewards are ignored.
  - Hurdle mechanics: ensuring penalties apply only once upon collision.

  Thoroughly tested cop movement, ensuring:

  - Cops correctly follow the thief while respecting movement constraints (e.g., valid tiles only).
  - Avoidance of merging into the same location as another cop.
  - Integration of the A* pathfinding algorithm for the shortest path generation.
- **Tests**:
  - Movement tests: Verified that the thief can only occupy valid road tiles and cannot enter gardens or buildings.
    - moveCopValidTest: Ensured that cops follow the thief correctly using valid paths.
    - copInvalidMoveTest: Tested that cops do not move to invalid locations, such asgardens or non-road tiles.

  - Direction tests: Ensured the thief moves in the intended direction based on player inputs or game logic.
  - Reward tests: Checked that reward are only collected once and inactive rewards (inactive Nitro)are ignored.
  - Hurdle tests: Validated that penalties from hurdles are only applied once upon collision.
  - **Pathfinding Tests**: Validated the correctness of the A* algorithm in generating the shortest path to the target.
  - **Collision Tests**: Ensured two cops avoid merging into the same location when moving toward the thief.

- **Approach**: Used Mockito to mock dependencies like Object Manager, MapSite, Cop1 and Cop2. Also created limited list of rewards and hurdles to test correct behavior in terms of reward value and penalty upon collision with rewards and hurdles respectively. Utilized parameterized tests for cop movement (getCopMoveValues) to test various scenarios systematically. Mocked dependencies such as Cop, Thief, and ObjectManager to isolate the map logic and validate individual functionalities.
- **Coverage Gaps**: The remaining 26% branch coverage consists of getter functions, which do encompass any critical logic of the game.
- **Result**: All other test cases passed except the one checking leftwards movement of thief. A bug was found while performing boundary testing. Thief won’t change position to the left if x was 1. It has been fixed in the code. All cop test cases passed, there were some logic issues with A-star pathfinding at first (expected position vs actual), but it ended up being a problem with the tests’ expected values, not the code.

5. **ObjectManagerTest**

- **Coverage**:

  - Achieved 62% line coverage and 65% branch coverage.

- Focused primarily on testing object initialization and error handling.

- **Focus**: Validated core ObjectManager functionalities:

  ◦ Initialization of rewards (Diamond, Nitro) and hurdles (Spike, Pothole) from valid map files.
  ◦ Proper handling of empty map files to ensure no objects are created.
  ◦ Testing of invalid file handling, ensuring that errors are caught appropriately.
  - **Tests**:
    ◦ **Rewards Initialization**: testRewardsInitialization checked if Diamond and Nitro objects are correctly initialized from a valid map file.
    ◦ **Hurdles Initialization**: testHurdlesInitialization validated if Spike and Pothole hurdles are correctly initialized.
    ◦ **Nitros Initialization**: testNitrosInitialization ensured Nitro objects are included in he rewards list.
  - **Empty File Handling**: testHandlingEmptyFile confirmed that no rewards or hurdles are created when provided with an empty map file.

- **Approach**:

- Used predefined map files (valid_rewards_map.txt, empty_rewards_map.txt) to simulate various scenarios and ensure proper object initialization. Invalid map files were passed to test exception handling. Additionally, tests were written to check for proper handling of missing or corrupted map data.
- **Coverage Gaps**: The uncovered 35% includes:
  - The actual reading and parsing of map files to create other game objects (e.g., roads, buildings).
  - These areas should be tested in more detail, especially for malformed or corrupted map files.
  - Further testing is needed for boundary cases, such as extremely large or invalid data sets.
- **Result:**
  - All tests cases passed, and it can be concluded that the exceptions are being handled correctly.

6. MapSiteTest Report

Coverage:

- **Achieved**: 85% test coverage.

- **Focus**: Verified map initialization, proper placement of Garden and Building, and checking for the absence of Road. Error handling for invalid files and invalid data formats should be considered for future tests.

- **Tests**:
    - **Initialization**: Verified that the map correctly loads a Garden at (0, 0) and a Building at (1, 1).
    - **Game Items**: Ensured that Garden and Building are correctly placed and that no Road objects are created.
    - **Error Handling**: While basic validation is conducted for map file correctness, additional tests for invalid or malformed files (e.g., INVALID_MAP_FILE) are needed.
- **Approach**:
    - Used a predefined test map file (test_map.txt) for testing.
    - Checked the placement and type of GameItem objects (Garden, Building), and confirmed no Road was placed.
    - No explicit tests for corrupted or malformed map files were performed and the current tests don't simulate scenarios where map files are improperly formatted, such as missing or extra characters, invalid data entries, or incorrect file extensions. This leaves potential for unhandled errors in production.
- **Findings**:
    - The MapSite class loads and initializes the map correctly.
    - Garden and Building items are correctly placed, while no Road items are created.
    - Map files with invalid content or incorrect file paths are not currently tested.
    - No changes are required to production code.
- **Coverage Gaps**:
    - Missing tests for error handling with invalid or corrupted map files.
    - Tests for edge cases or malformed data formats could be added.
    - Tests for additional GameItem types, such as Road, should be expanded to check for proper inclusion when applicable.
- **Result:**
    - Achieved 85% test coverage. Verified correct initialization and placement of Garden and Building, with absence of Road objects. Error handling for invalid or malformed map files remains untested. No changes needed to production code, but additional tests for error handling, edge cases, and other GameItem types are recommended.