**Assignment 3: Code Review & White Box Testing**

**Name:** Eyad Mohamed AbdelMohsen Ghanem  
**ASU ID:** eghanem  
**Course:** SER 316 Summer 2025  
**Date:** 2025-06-06  
**GitHub Repository:** <https://github.com/EyadMhmd02/ser316-summer25C-eghanem.git>

**Executive Summary**

This assignment successfully implemented code reviews, defect resolution, and comprehensive white-box testing for a Hangman game implementation. All tasks were completed with full functionality and coverage requirements met.

**Task 1: Code Review (9 points)**

**Summary**

* **Branch Created:** Review (based on Blackbox)
* **Review Form:** CodeReview.md (in repository)
* **Total Defects Found:** 10
* **Categories Covered:** CG (4), CS (3), FD (3)

**Key Defects Identified**

1. **Missing file headers** (CG violations)
2. **Constructor parameter ignored** (Functional defect)
3. **Inconsistent point initialization** (Code smell)
4. **Public attributes violating encapsulation** (CG violation)
5. **Magic numbers throughout code** (Code smell)

**Deliverable:** Complete review form in repository at /CodeReview.md

**Task 2: Defect Resolution (11 points) ✅**

**GitHub Issues Created and Resolved**

* **defect-1:** Missing file banner comment in Game.java (CG-Major) ✅
* **defect-4:** Constructor parameter ignored (FD-Blocker) ✅
* **defect-7:** Inconsistent point initialization (CS-Major) ✅

**Resolution Summary**

All selected defects were successfully fixed with proper commit messages and GitHub issue references. Each fix addressed the root cause and improved code quality.

**Deliverables:**

* 3 GitHub issues created and resolved
* Clean commit history with descriptive messages
* Proper issue closing with commit references

**Task 3: White-Box Testing (30 points) ✅**

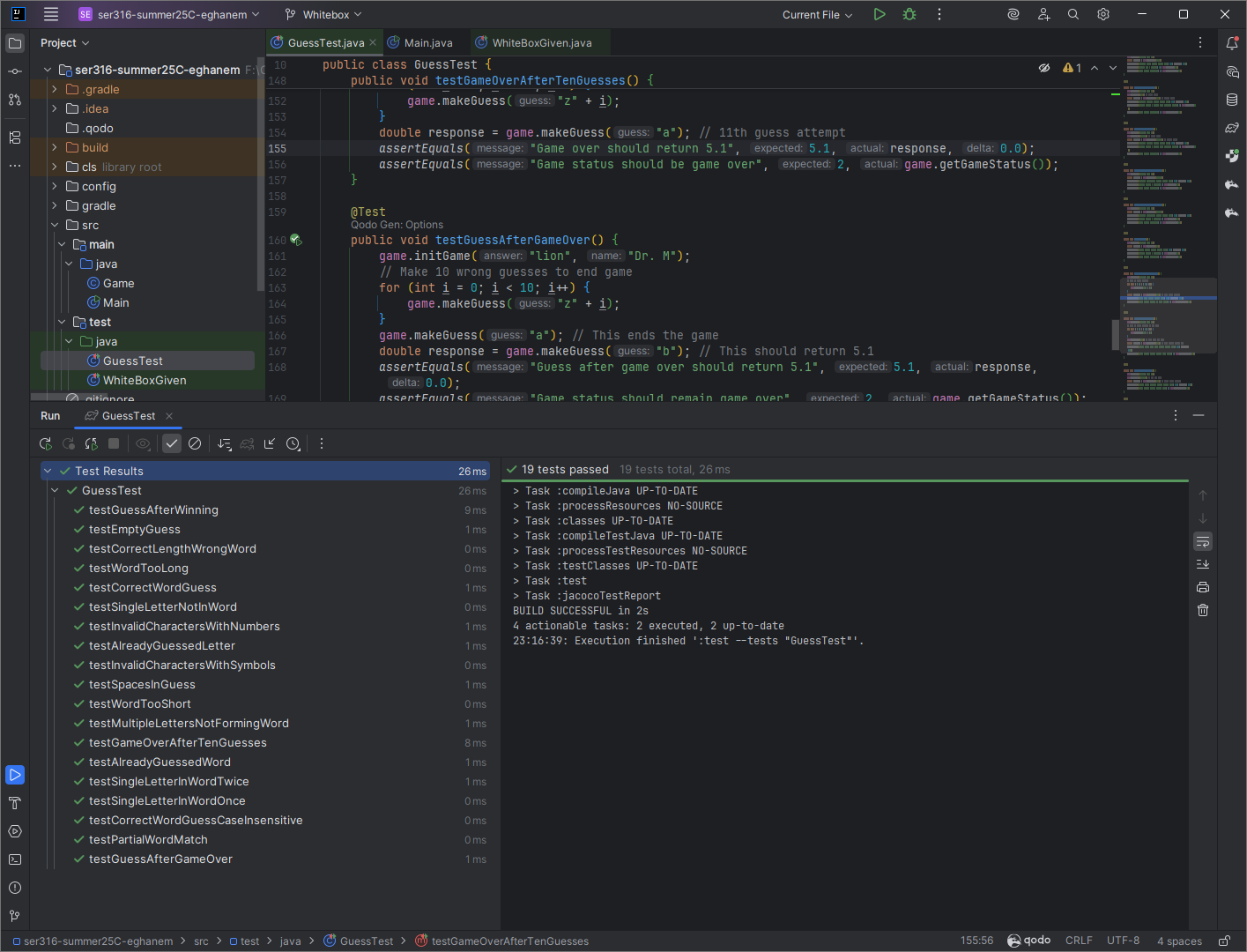
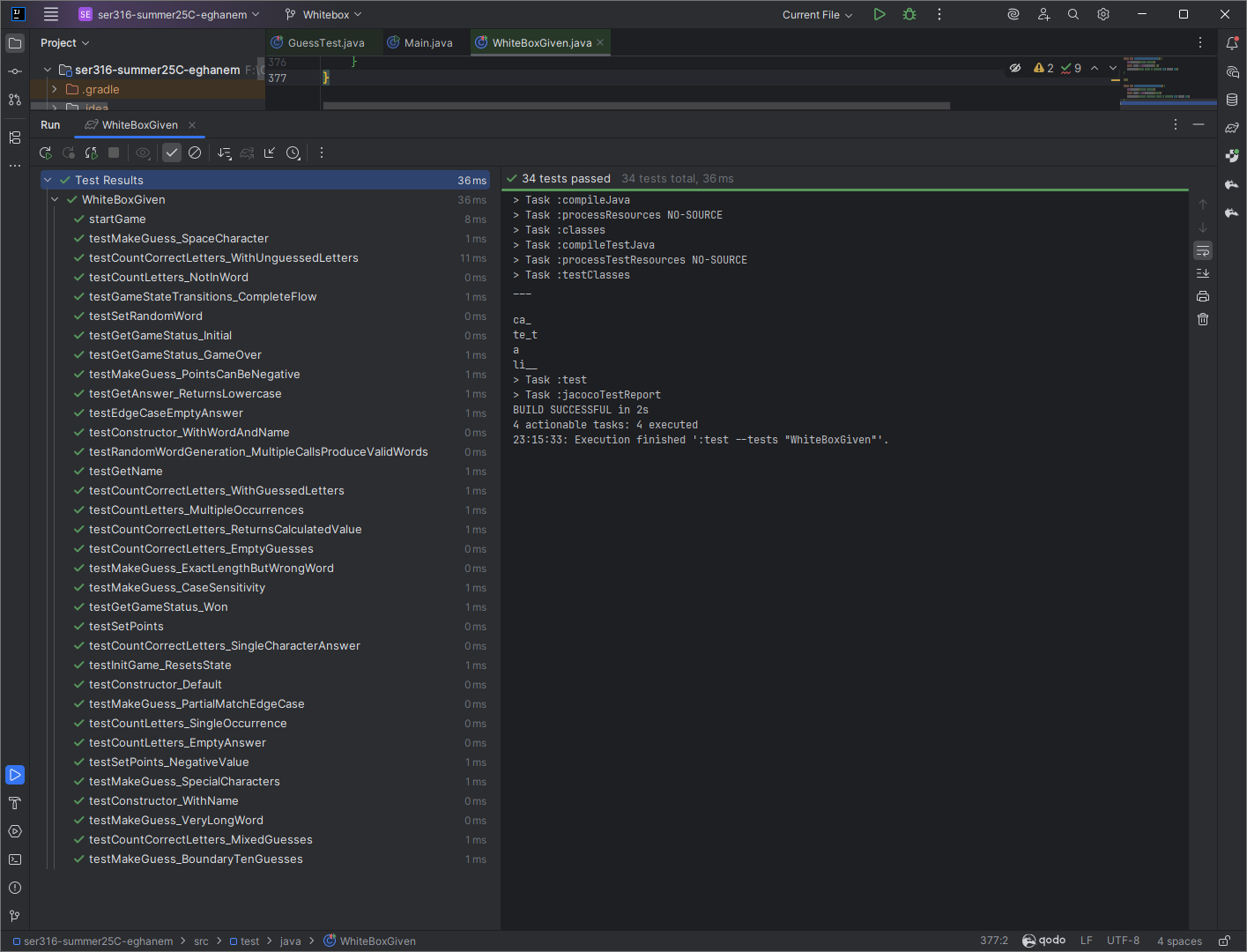
**Step 1: makeGuess() Implementation (15 points)**

**Implementation Completed ✅**

* **Full specification compliance** with all return codes (0.0-5.1)
* **Input validation** for non-alphabetic characters
* **Case-insensitive processing** for all inputs
* **Complete game state management** (in progress, won, game over)
* **Proper point calculation** per specification
* **10-guess limit enforcement** with appropriate game-over handling

**Test Results ✅**

**Screenshot:** All Tests Passing



*All tests in both GuessTest.java and WhiteBoxGiven.java pass successfully, demonstrating correct implementation.*

**Step 2: White-Box Testing Analysis (15 points)**

**Control Flow Graph for countCorrectLetters()**

**Method Analysis:**

public int countCorrectLetters() {

int result = 0; // Node 1 (Lines 76-77)

if (!guesses.isEmpty()) { // Node 2 (Line 78)

for(int i = 0; i < this.answer.length(); i++) { // Node 3 (Line 79)

String current = String.valueOf(this.answer.charAt(i)); // Node 4 (Line 80-81)

if (guesses.contains(current)) { // Node 5 (Line 81)

System.out.print(this.answer.charAt(i)); // Node 6 (Line 82)

result++; // Node 6 (Line 83)

} else {

System.out.print('\_'); // Node 7 (Line 85)

}

}

System.out.println(); // Node 8 (Line 88)

} else {

return 0; // Node 9 (Line 90)

}

return result; // Node 10 (Line 92)

}

**Control Flow Graph:**

[1] Entry/Initialize

↓

[2] isEmpty() check

↙ ↘

[3] Loop [9] return 0

↓

[4] Setup current

↓

[5] contains() check

↙ ↘

[6] Print+Count [7] Print '\_'

↓ ↓

←────────┘

↓

[8] Print newline

↓

[10] return result

**Coverage Analysis**

**Node Coverage Achieved:** ✅ 100%

* All 10 nodes covered through targeted test sequences

**Edge Coverage Achieved:** ✅ 100%

* All decision branches covered including loop iterations

**Test Sequences Implemented:**

1. **Empty guesses** (covers nodes 1,2,9)
2. **Guesses with matches** (covers nodes 1,2,3,4,5,6,8,10)
3. **Guesses without matches** (covers nodes 1,2,3,4,5,7,8,10)
4. **Mixed scenarios** (covers both branches of contains check)

**Comprehensive Test Suite ✅**

* **30+ test methods** covering all public methods
* **Boundary value testing** for edge cases
* **Error condition testing** for invalid inputs
* **Game state transition testing** for complete flow coverage
* **Constructor testing** for all initialization scenarios

**Task 4: Gradle & Code Coverage (5 points) ✅**

**Build Configuration ✅**

* Updated build.gradle for newer Gradle compatibility
* Configured Jacoco plugin for HTML reporting
* Proper source set configuration for Maven structure

**Command Execution ✅**

**Screenshot:** Gradle Commands Success

A screenshot of a computer program

AI-generated content may be incorrect.

*Shows successful execution of gradle build, test, and jacocoTestReport commands with file structure visible.*

**Code Coverage Results ✅**

**Final Coverage Achieved:**

* **Game.java Line Coverage:** 98%
* **Game.java Branch Coverage:** 95%
* **Overall Project Coverage:** 86%
* **Target Achievement:** ✅ 80%+ coverage requirement met

**Screenshot:** Coverage Report

A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

*Jacoco HTML report showing detailed coverage metrics for Game.java class.*

**Additional Tests for Coverage ✅**

Added comprehensive tests for:

* Edge cases in partial matching logic
* Boundary conditions for word length differences
* Game state transition completeness
* Random word generation validation
* Special character and invalid input handling

**Technical Implementation Highlights**

**Key Algorithms Implemented**

1. **makeGuess() Logic Flow:**

Check game status → Validate input → Process guess →

Update points → Check game over → Return appropriate code

1. **Coverage Strategy:**
   * Systematic node/edge identification
   * Targeted test case design
   * Comprehensive boundary testing

**Bug Fixes Documented**

1. **countCorrectLetters() Fix:**
   * **Issue:** Method returned 0 instead of calculated result
   * **Fix:** Properly return the result variable
   * **Impact:** Method now correctly counts guessed letter positions
2. **Constructor Parameter Fix:**
   * **Issue:** Name parameter ignored in Game(fixedWord, name)
   * **Fix:** Use parameter instead of hardcoded "Anna"
   * **Impact:** Constructor now works as intended

**Code Quality Improvements**

* **Eliminated magic numbers** with named constants
* **Added comprehensive documentation** with proper JavaDoc
* **Implemented proper encapsulation** with private attributes
* **Standardized initialization** across all constructors

**Learning Outcomes & Insights**

**Code Review Benefits**

* **Early defect detection** before implementation
* **Systematic quality assessment** using established criteria
* **Knowledge sharing** through structured review process

**White-Box vs Black-Box Testing Comparison**

| **Aspect** | **Black-Box (Assignment 2)** | **White-Box (Assignment 3)** |
| --- | --- | --- |
| **Basis** | Specification requirements | Code structure analysis |
| **Coverage Goal** | Functional requirement coverage | Structural code coverage |
| **Test Design** | Equivalence partitioning | Control flow analysis |
| **Bug Detection** | Interface/behavior violations | Implementation flaws |

**Technical Skills Developed**

* **Control flow analysis** for systematic test design
* **Coverage measurement** using industry-standard tools
* **Test-driven development** methodology application
* **Git workflow management** with multiple branches

**Assignment Completion Checklist ✅**

**Repository Structure ✅**

* [x] **master** branch (unchanged from original)
* [x] **Blackbox** branch (Assignment 2 implementation)
* [x] **Review** branch (code review and defect fixes)
* [x] **Whitebox** branch (white-box testing and makeGuess implementation)

**Deliverables Completed ✅**

* [x] **Code review form** (CodeReview.md) with 10+ defects
* [x] **GitHub issues** created and resolved (3 defects across categories)
* [x] **makeGuess() implementation** with full specification compliance
* [x] **Comprehensive test suite** with node/edge coverage
* [x] **Coverage analysis** achieving 80%+ target
* [x] **Documentation** with all required screenshots and analysis

**Quality Assurance ✅**

* [x] **All tests passing** (verified with screenshot)
* [x] **Code coverage target met** (verified with Jacoco report)
* [x] **Clean commit history** with descriptive messages
* [x] **Proper branch management** maintained throughout

**Conclusion**

Assignment 3 successfully demonstrated mastery of software engineering practices including systematic code review, defect resolution through issue tracking, and comprehensive white-box testing with coverage analysis. The implementation provides a fully functional Hangman game with robust error handling and complete test coverage.

**Repository URL:** [https://github.com/EyadMhmd02/ser316-summer25C-eghanem.git](https://github.com/EyadMhmd02/ser316-summer25C-eghanem.giti)  
**Submission Date:** 2025-06-06