CS 1632 – FINAL DELIVERABLE:

True Test-Driven Development of Baccarat

Brandon S. Hang

https://github.com/brandonhang/Pitt-Projects/tree/master/cs1632/Final%20Deliverable

For my final project, I decided to develop the card game *baccarat* in a true test-driven development cycle. When I mean true, I refer to the red-green-refactor cycle that is the critical to test-driven development. I.e., I specifically wrote JUnit test cases before writing any implementation code whatsoever. I also made sure these tests covered as many common, edge, and base cases as I could think of. This resulted in 100% code coverage of my non-main methods according to the EclEmma tool. While I was able to cover all of my non-main methods, it is nigh impossible to cover 100% of possible bugs. However, by trying to cover 99.99…% of them, I can at least approach this level without having to exert an exponential amount of effort.

I also tried to stick with SOLID and YAGNI paradigms, particularly the single-responsibility principle of SOLID. Wherever possible, each of the classes in my project is responsible for only one function or one set of related functions. For example, the Card class handles individual cards while the Deck class handles collections of Card objects. The Baccarat class then utilizes the Deck class as a dealer.

Being that baccarat is an interactive card game, it is also necessary to write many things to the console. However, printing output is a side effect and cannot be directly tested. To alleviate this, I wrote all (nearly) all my print statements as functions that return a Boolean. If the print was successful, the function will return true. If an invalid parameter was used, the print statement will return false or throw an expected exception depending on the specific function.

Despite the code coverage and sheer number of JUnit tests, I still needed to perform manual testing of my program. It was only through manually testing the game that I discovered several problems not found through unit testing. The first major one was that my Deck class was not correctly generating the standard 52 playing cards. This had to be fixed by rewriting both the deck creation algorithm and unit tests pertaining to it. The other problem, albeit minor, was that my CardPrint class was not formatting the ASCII card artwork correctly. This is a bug that was only detectable through manual testing. Another minor issue related to the CardPrint class was the character encoding of the text artwork. While the Eclipse IDE uses CP-1252 character encoding by default, many of those special characters are not displayable in either the Windows Command Prompt or the OS X Terminal without changing environment variables. As such, I had to limit my text artwork to ASCII encoded characters.

Repo: https://github.com/brandonhang/Pitt-Projects/tree/master/cs1632/Final%20Deliverable

While I don't foresee any problems with this program (short of a meteor impact or EMP blast), I'm not sure I would develop a program strictly under test-driven development moving forward. As I was developing this game under the red-green refactor cycle, I wasn't able to make the design choices regarding the overall structure of the game. For example, I probably wouldn't create a separate Tableau class to decide the drawing rules for the player and banker. I also wouldn't have created a separate GameUI class with non-void print functions.

Re: Baccarat Quality Assessment

HB **Hang, Brandon S**
To: ☐ Laboon, William J; ⩔

Inbox
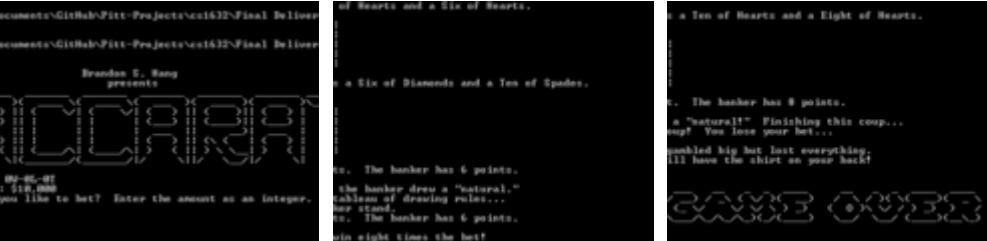
⩔ Show all 3 attachments (27 KB)   Download all   Save all to OneDrive - Pittsburgh Games Co.

Bill,

Below is a quality assessment of the *Baccarat App* being developed by the Text Games Department.

Subsystem	Status	Notes
Card Management	Green	Minor defects regarding creation of the playing deck were found and corrected. No further defects were found and this subsystem is completed.
Punto Banco Rules	Green	The latest version passed all test cases including edge and corner cases. Manual testing of game logic passed as well. This subsystem is completed.
User Interface	Green	A minor defect was resolved where incorrect symbols were displayed on the screen. The resulting test plan passed with no further defects. This subsystem is completed.

As you can see, all subsystems are green and completed. Overall, the app meets all requirements specified by management and passes all tests. See the attached screenshots for an example run of the game. As a result, I am happy to report that this app is ready to ship for the public. Please forward my recommendation to upper management.

If you have any further questions about what or how we tested the app, do not hesitate to reply to this email.

Thanks,

Brandon

Brandon S. Hang
QA Test Engineer Lead
412.123.4567
brandon.hang@pittgames.com

Finished after 1.127 seconds

Runs: 116/116   ☒ Errors: 0   ☒ Failures: 0

▲ baccarat_test_suite.GameUITestPlan [Runner: JUnit 4] (0.115 s)
- testNullPlayerGameOver (0.002 s)
- testPrintInitialPlayerHand (0.000 s)
- testUnparsableString (0.000 s)
- testNegativeIntegerString (0.000 s)
- testIllegalCoupConditionsBankerUnder (0.000 s)
- testPrintPlayerInvalidExtraCard (0.000 s)
- testInvalidPrintExtraCardParameter (0.000 s)
- testIllegalBankerPoints (0.000 s)
- testCutCardNullDeck (0.000 s)
- testPlayerTie (0.000 s)
- testPlayerWin (0.001 s)
- testNormalGameOverWonMoney (0.001 s)
- testPrintNaturalPlayerOnly (0.000 s)
- testNormalGameOverLostMoney (0.000 s)
- testParsableString (0.000 s)
- testNormalGameOverSameMoney (0.000 s)
- testPrintCoupPoints (0.000 s)
- testPrintIntro (0.000 s)
- testPrintNaturalBoth (0.000 s)
- testPrintHandInvalidParameter (0.000 s)
- testNormalGameOverBlewItAll (0.000 s)
- testCutCardFoundEdge (0.000 s)
- testUnparsableDecimal (0.000 s)
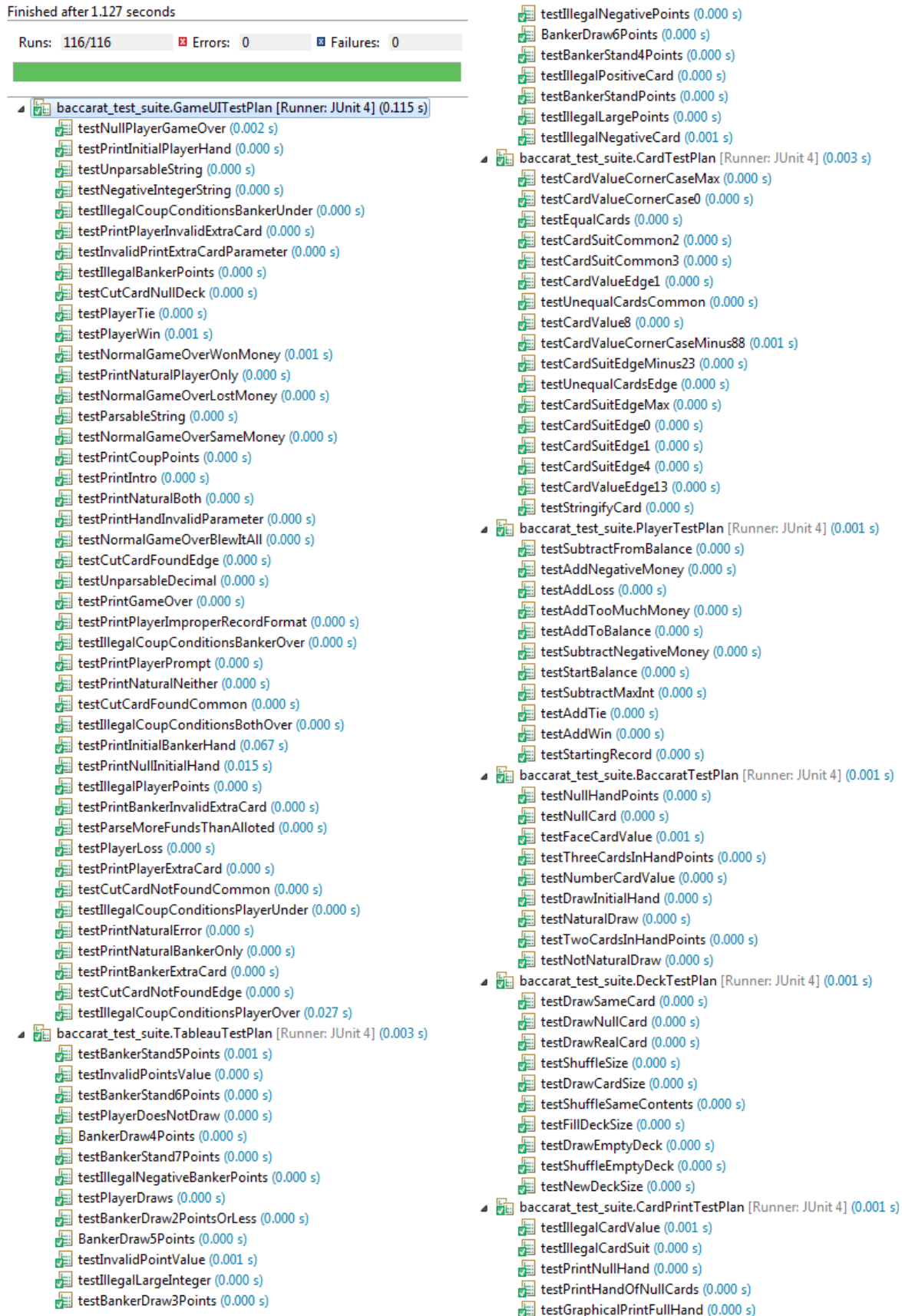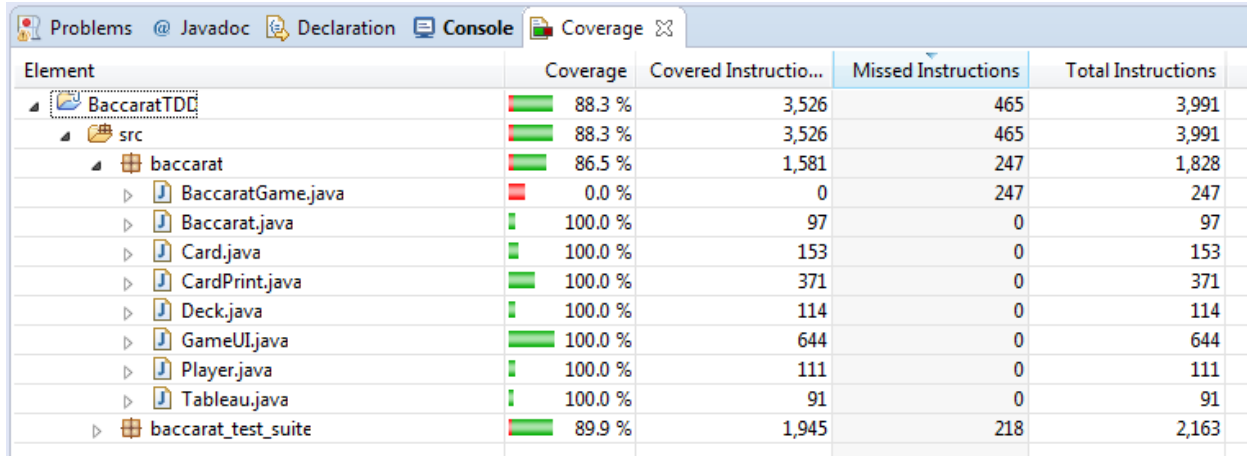- testPrintGameOver (0.000 s)
- testPrintPlayerImproperRecordFormat (0.000 s)
- testIllegalCoupConditionsBankerOver (0.000 s)
- testPrintPlayerPrompt (0.000 s)
- testPrintNaturalNeither (0.000 s)
- testCutCardFoundCommon (0.000 s)
- testIllegalCoupConditionsBothOver (0.000 s)
- testPrintInitialBankerHand (0.067 s)
- testPrintNullInitialHand (0.015 s)
- testIllegalPlayerPoints (0.000 s)
- testPrintBankerInvalidExtraCard (0.000 s)
- testParseMoreFundsThanAlloted (0.000 s)
- testPlayerLoss (0.000 s)
- testPrintPlayerExtraCard (0.000 s)
- testCutCardNotFoundCommon (0.000 s)
- testIllegalCoupConditionsPlayerUnder (0.000 s)
- testPrintNaturalError (0.000 s)
- testPrintNaturalBankerOnly (0.000 s)
- testPrintBankerExtraCard (0.000 s)
- testCutCardNotFoundEdge (0.000 s)
- testIllegalCoupConditionsPlayerOver (0.027 s)

▲ baccarat_test_suite.TableauTestPlan [Runner: JUnit 4] (0.003 s)
- testBankerStand5Points (0.001 s)
- testInvalidPointsValue (0.000 s)
- testBankerStand6Points (0.000 s)
- testPlayerDoesNotDraw (0.000 s)
- BankerDraw4Points (0.000 s)
- testBankerStand7Points (0.000 s)
- testIllegalNegativeBankerPoints (0.000 s)
- testPlayerDraws (0.000 s)
- testBankerDraw2PointsOrLess (0.000 s)
- BankerDraw5Points (0.000 s)
- testInvalidPointValue (0.001 s)
- testIllegalLargeInteger (0.000 s)
- testBankerDraw3Points (0.000 s)

- testIllegalNegativePoints (0.000 s)
- BankerDraw6Points (0.000 s)
- testBankerStand4Points (0.000 s)
- testIllegalPositiveCard (0.000 s)
- testBankerStandPoints (0.000 s)
- testIllegalLargePoints (0.000 s)
- testIllegalNegativeCard (0.001 s)

▲ baccarat_test_suite.CardTestPlan [Runner: JUnit 4] (0.003 s)
- testCardValueCornerCaseMax (0.000 s)
- testCardValueCornerCase0 (0.000 s)
- testEqualCards (0.000 s)
- testCardSuitCommon2 (0.000 s)
- testCardSuitCommon3 (0.000 s)
- testCardValueEdge1 (0.000 s)
- testUnequalCardsCommon (0.000 s)
- testCardValue8 (0.000 s)
- testCardValueCornerCaseMinus88 (0.001 s)
- testCardSuitEdgeMinus23 (0.000 s)
- testUnequalCardsEdge (0.000 s)
- testCardSuitEdgeMax (0.000 s)
- testCardSuitEdge0 (0.000 s)
- testCardSuitEdge1 (0.000 s)
- testCardSuitEdge4 (0.000 s)
- testCardValueEdge13 (0.000 s)
- testStringifyCard (0.000 s)

▲ baccarat_test_suite.PlayerTestPlan [Runner: JUnit 4] (0.001 s)
- testSubtractFromBalance (0.000 s)
- testAddNegativeMoney (0.000 s)
- testAddLoss (0.000 s)
- testAddTooMuchMoney (0.000 s)
- testAddToBalance (0.000 s)
- testSubtractNegativeMoney (0.000 s)
- testStartBalance (0.000 s)
- testSubtractMaxInt (0.000 s)
- testAddTie (0.000 s)
- testAddWin (0.000 s)
- testStartingRecord (0.000 s)

▲ baccarat_test_suite.BaccaratTestPlan [Runner: JUnit 4] (0.001 s)
- testNullHandPoints (0.000 s)
- testNullCard (0.000 s)
- testFaceCardValue (0.001 s)
- testThreeCardsInHandPoints (0.000 s)
- testNumberCardValue (0.000 s)
- testDrawInitialHand (0.000 s)
- testNaturalDraw (0.000 s)
- testTwoCardsInHandPoints (0.000 s)
- testNotNaturalDraw (0.000 s)

▲ baccarat_test_suite.DeckTestPlan [Runner: JUnit 4] (0.001 s)
- testDrawSameCard (0.000 s)
- testDrawNullCard (0.000 s)
- testDrawRealCard (0.000 s)
- testShuffleSize (0.000 s)
- testDrawCardSize (0.000 s)
- testShuffleSameContents (0.000 s)
- testFillDeckSize (0.000 s)
- testDrawEmptyDeck (0.000 s)
- testShuffleEmptyDeck (0.000 s)
- testNewDeckSize (0.000 s)

▲ baccarat_test_suite.CardPrintTestPlan [Runner: JUnit 4] (0.001 s)
- testIllegalCardValue (0.001 s)
- testIllegalCardSuit (0.000 s)
- testPrintNullHand (0.000 s)
- testPrintHandOfNullCards (0.000 s)
- testGraphicalPrintFullHand (0.000 s)

Figure 1. The JUnit test suite for the baccarat Java game.

Repo: https://github.com/brandonhang/Pitt-Projects/tree/master/cs1632/Final%20Deliverable

Element	Coverage		Covered Instructio...	Missed Instructions	Total Instructions
▲ 🗁 BaccaratTDD	▬	88.3 %	3,526	465	3,991
▲ 📂 src	▬	88.3 %	3,526	465	3,991
▲ ⊞ baccarat	▬	86.5 %	1,581	247	1,828
▷ 🗋 BaccaratGame.java	▬	0.0 %	0	247	247
▷ 🗋 Baccarat.java	▮	100.0 %	97	0	97
▷ 🗋 Card.java	▮	100.0 %	153	0	153
▷ 🗋 CardPrint.java	▬	100.0 %	371	0	371
▷ 🗋 Deck.java	▮	100.0 %	114	0	114
▷ 🗋 GameUI.java	▬	100.0 %	644	0	644
▷ 🗋 Player.java	▮	100.0 %	111	0	111
▷ 🗋 Tableau.java	▮	100.0 %	91	0	91
▷ ⊞ baccarat_test_suite	▬	89.9 %	1,945	218	2,163

Figure 2. EclEmma code coverage of the test suite.