LINNAEUS UNIVERSITY

Assignment 3

Project: Hangman – Test plan

Christoffer Lundström
Cl222ae@student.lnu.se
https://github.com/CLundstrom/cl222ae_1dv600

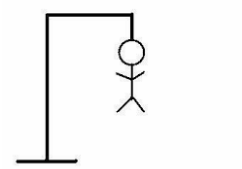
25/02-19

Software Technology – IDV600

Teachers: Tobias Andersson Gidlund

Daniel Toll

Tobias Olsson



Test plan

The goal of this iteration is to make sure that the project lives up to the requirement set in previous iterations through validation & verification. This will hopefully lead to exposure of flaws and build confidence in the product.

Throughout the application there are a few critical aspects associated to User Input. Thus this will be the main focus of the following unit tests. User inputs are the most variable factors of this application and is therefore more important to test in this case over system internals.

This might generally not be the case in larger projects where user input might be a substantially smaller part of the application.

Time-plan with estimations and results is found at page 8 and also in the updated in the project plan.

It is the intention to focus mainly on testing the Player and Game-object which both involve the Play Use case and in extension the create player Use case. This will be done using dynamic manual tests and unit tests.

In the automated test part the functions isRoundLost and isCorrectGuess in the Game object class will be evaluated and tested using JUnit 5 along with further tests of the Player constructor.

Manual Test cases

TC 1.1.1

Description: Test to verify the typical success scenario.
Use case ref: UC1 – Play (**Create Player**)

Precondition: Play has been chosen in the menu.

Test Steps

1. Player is shown ”Enter name: ”
2. Enter “John Doe”

Expected:

Name is stored and game round is started.

Comments:

TC 1.1.2

Description: Verifies that the name stays within length boundaries.
Use case ref: UC1 – Play (**Create Player**)

Precondition: Play has been chosen in the menu.

Test Steps

1. Player is shown ”Enter name: ”
2. Press enter.

Expected:

1. Player is shown: “Invalid amount of characters. (min 3, max 15)”
2. Player is returned to “Enter name: “.

Comments:
Test failed. Game continued with an empty String as accepted name.
Will be fixed by next commit.

TC 1.1.3

Description: Verifies that the name don’t exceed 15 characters.
Use case ref: UC1 – Play (**Create Player**)

Precondition: Play has been chosen in the menu.

Test Steps

1. Player is shown ”Enter name: ”
2. Enter “AntiDisestablishMentarianism**”**

Expected:

1. Player is shown: “Invalid amount of characters. (min 3, max 15)”
2. Player is returned to “Enter name: “.

Comments:

TC 1.2.1

Description: Tests the main success scenario. A letter is either correct or wrongly guessed.
Use case ref: UC1 – Play (**Guess letter**)

Precondition: Name has been entered.

Test Steps

1. Player is shown ”Guess a letter. Fails left: (number)”
2. Enter “b”

Expected:

Randomized word contains letter.
Frame or Fails is updated.

Comments:

TC 1.2.2

Description: Tests the main success scenario. Player wins the game.
Use case ref: UC1 – Play (**Guess letter**)

Precondition: Name is entered & all letters but 1 has been correctly guessed.

Test Steps

1. Player is shown “Guess a letter. Fails left: (number)”
2. Enter “b”

Expected:

Guess is correct -> Victory screen is shown.
Guess is wrong -> Frame and fail updated.

Comments:

TC 1.2.3

Description: Makes sure game does not progress without input from user.
Use case ref: UC1 – Play (**Guess letter**)

Precondition: Name has been entered.

Test Steps

1. Player is shown ”Guess a letter. Fails left: (number)”
2. Press enter. (no input)
3. Press enter. (no input)
4. Press enter. (no input)

Expected:

Nothing happens. Still waiting for input.

Comments:

TC 1.2.4

Description: Test to make sure game does not throw an Exception on wrong inputs.
Use case ref: UC1 – Play (Guess letter)

Precondition: Name has been entered.

Test Steps

1. Player is shown ”Guess a letter. Fails left: (number)”
2. Enter “>^12---Abc<<”

Expected:

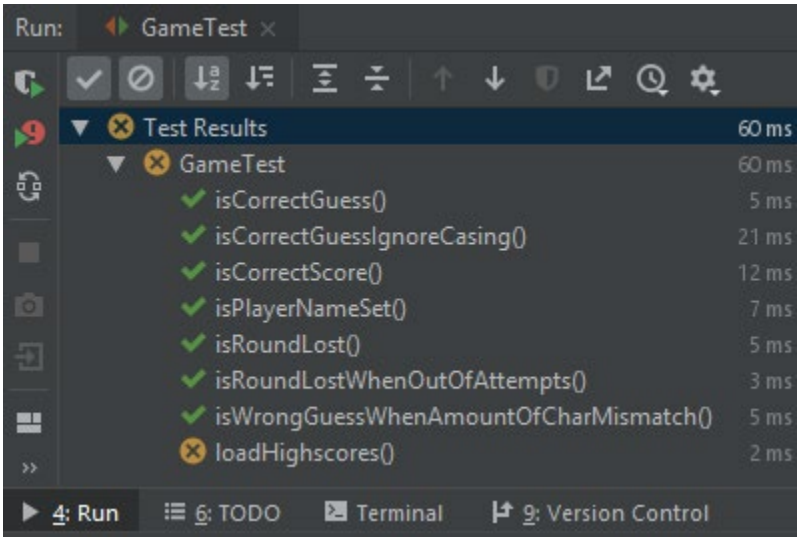
Player is shown. “You may only enter one letter.”
Player is returned to “Enter:”

Comments:

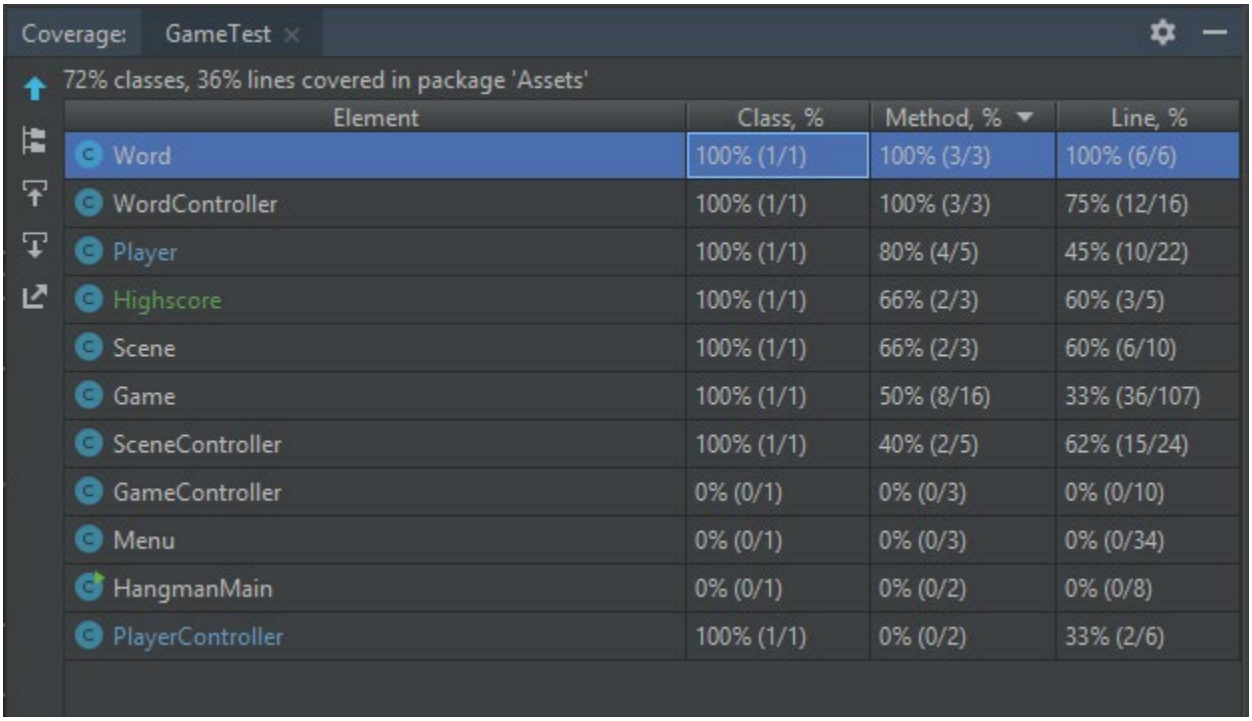
Automated Unit test

Below are the results of the automated unit tests after a minor change to the guessing-algorithm and the player object followed by the code coverage analysis.

Tests



Test coverage



Element	Class, %	Method, %	Line, %
Word	100% (1/1)	100% (3/3)	100% (6/6)
WordController	100% (1/1)	100% (3/3)	75% (12/16)
Player	100% (1/1)	80% (4/5)	45% (10/22)
Highscore	100% (1/1)	66% (2/3)	60% (3/5)
Scene	100% (1/1)	66% (2/3)	60% (6/10)
Game	100% (1/1)	50% (8/16)	33% (36/107)
SceneController	100% (1/1)	40% (2/5)	62% (15/24)
GameController	0% (0/1)	0% (0/3)	0% (0/10)
Menu	0% (0/1)	0% (0/3)	0% (0/34)
HangmanMain	0% (0/1)	0% (0/2)	0% (0/8)
PlayerController	100% (1/1)	0% (0/2)	33% (2/6)

Test Report

Manual tests

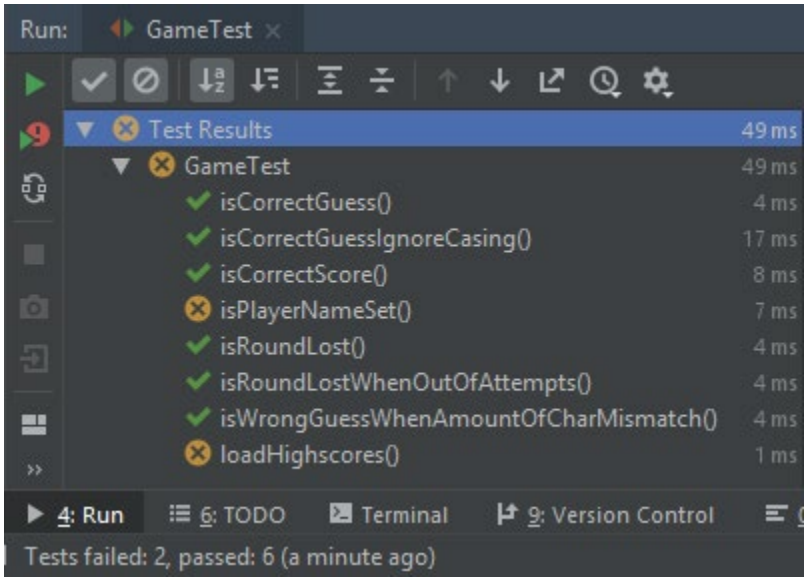
An error with the Player Controller was detected during the manual tests. This made it possible to create a Player without a name signature.

This has been corrected in current commit.

Test cases	UC 1.1	UC 1.2
TC 1.1.1	1/OK	0
TC 1.1.2	1/FAIL	0
TC 1.1.3	1/OK	0
TC 1.2.1	0	1/OK
TC 1.2.2	0	1/OK
TC 1.2.3	0	1/OK
TC 1.2.4	0	1/OK
Coverage & Success	3/OK (2/3)	4/OK

Automated tests

The same error with the player constructor showed up in the Automated tests.
Load highscores is expected to fail because it is not yet fully implemented.



Copy of time plan from project plan

7. | Time log

Below is a time log spanning the time estimated and taken to complete each task.

Date	Task	Estimated	Actual	Diff +/-
280119	T1: Fill in basics of project template	1h	1h	0
290119	T1: Game idea research	1h	1h	0
300119	T1: Setting up GitHub repository	1h	0.5h	-0.5
310119	T1: Sketch game logic design	2h	1h	-1
020219	Rewrite project template	2h	2h	0
040219	T1: Project summary & vision	2h	4h	+2
050219	T1: Project plan & iterations	4h	6h	+2
060219	T1: Risk analysis	3h	5h	+2
070219	T1: Risk analysis cont.	2h	1h	-1
070219	T2: Implement basic UI	2h	1h	-1
>>>>>>>>>>	**Iteration 1**	20h	22	+2
120219	T4: Hangman drawing	2h	1h	-1
130219	T5: Word list implemented	2h	1h	-1
140219	T3: Design & implement	8h	2h	-1
150219	T3 cont.	-	2h	x
180219	T3 cont.	-	3h	x
190219	Task 7: Game logic	8	4h	+4
200219	T7 cont.	-	8h	x
210219	Task 6: Interfacing	5h	6h	+1
>>>>>>>>>>	**Iteration 2**	25h	27h	+2
-	~~T8: Highscore list~~ (discontinued)	4h	-	
040319	T8: Test plan	3h	4h	+1
040319	T8: Manual tests	2h	1h	-1
050319	T8: Automated tests write & run	3h	2h	-1
060319	T8: Code revision	1h	1h	0
070319	T8: Automated test rerun	1h	1h	0
>>>>>>>>>>	**Iteration 3**	10h	9h	-1

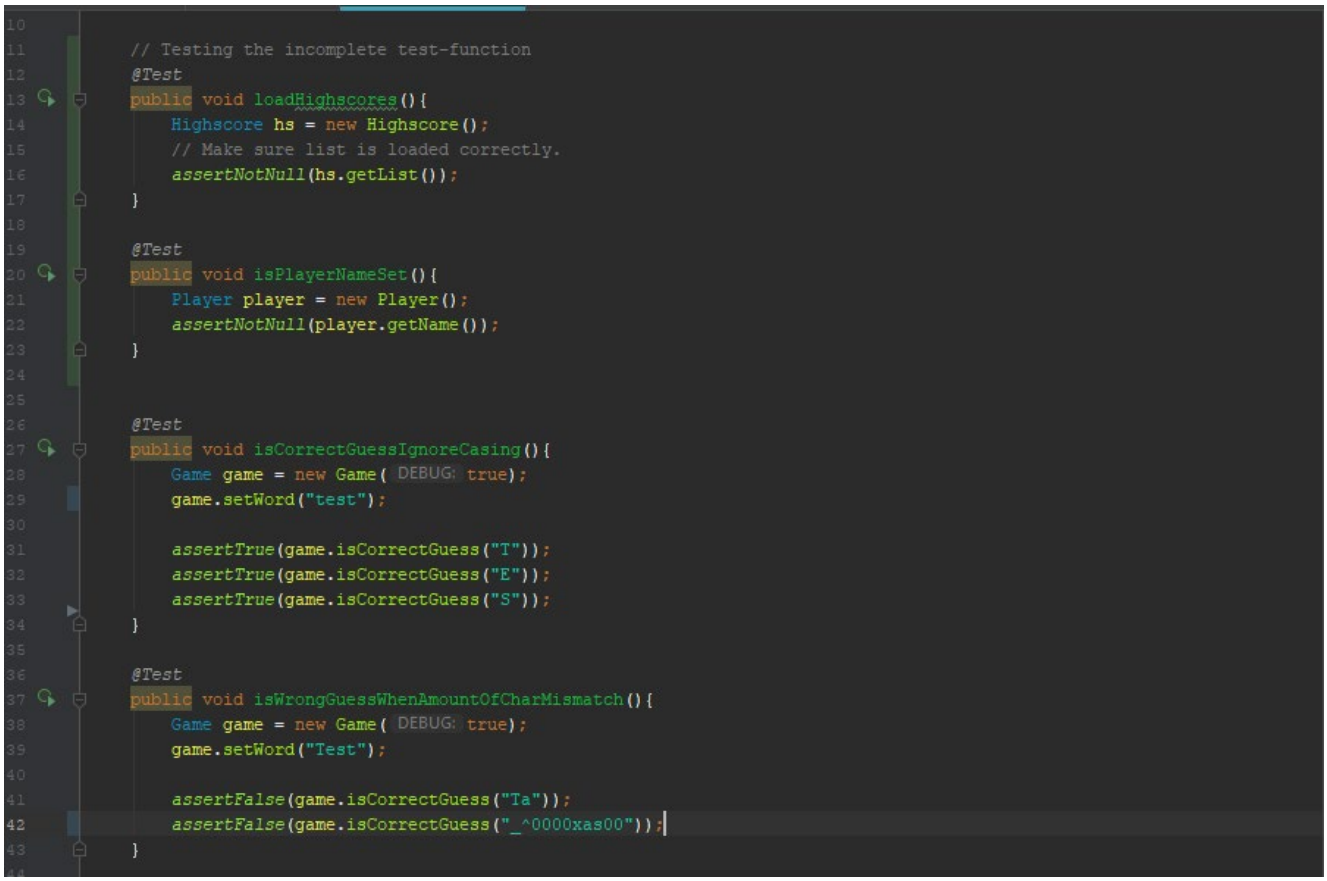
Reflection

This iteration has been a real learning experience to get down with the tiny details of testing. It shows how easy it is to overlook small details such as an empty string that will in turn break the entire purpose of the application. As a customer I would be deeply dissatisfied if such errors passed code reviews and made it onto a live system. It also shows the importance of having a structured way of reviewing code. Having use-cases, class diagrams and especially state machine diagrams available made it a lot easier to get an overview of where to focus the manual and automated tests.

It should be noted though, that diagrams might not always show a truthful representation of the system which makes it important not to get too narrow-minded when it comes to testing, especially in an agile development setting.

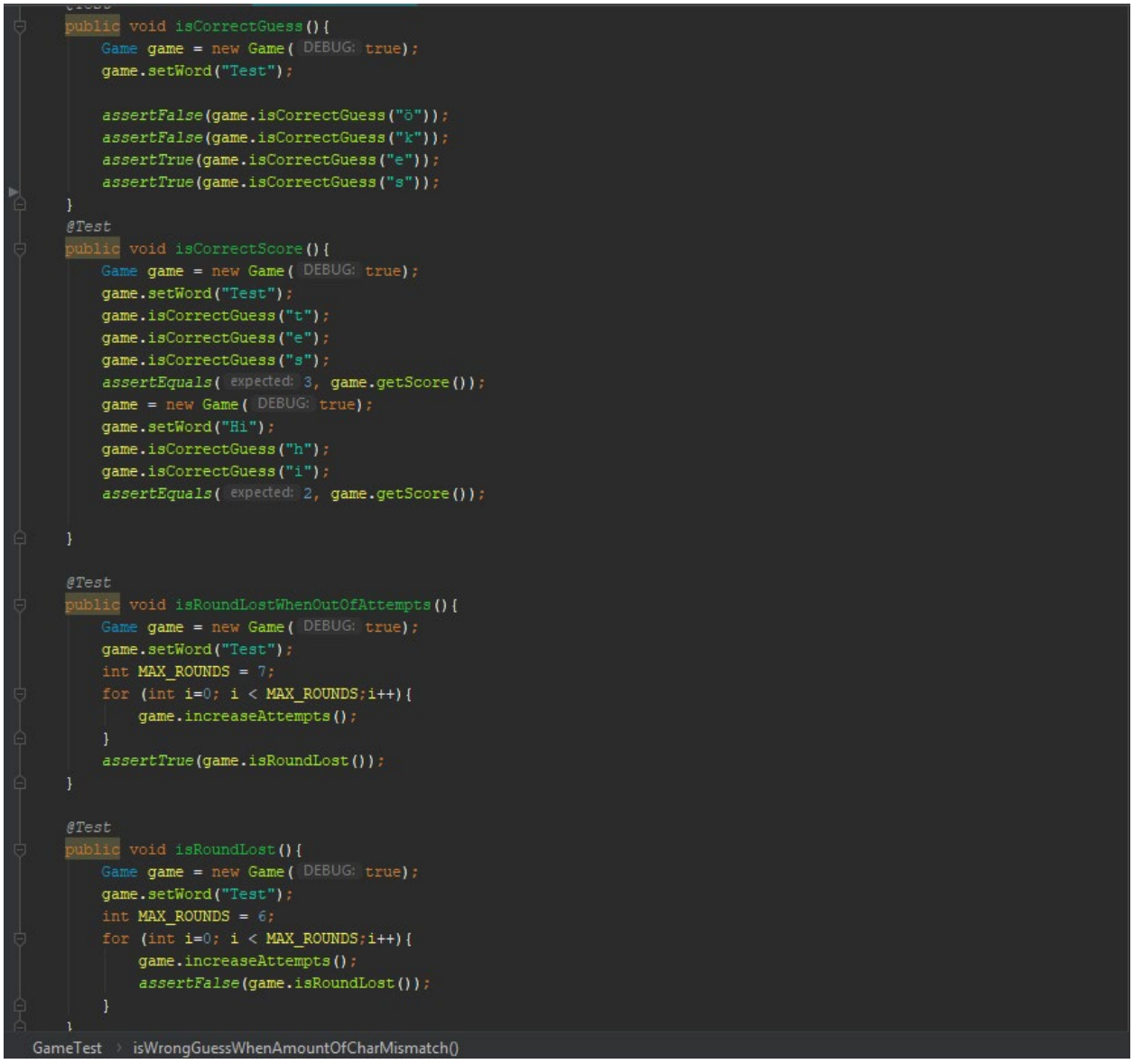
Test code screenshots

Screenshot 1



```
10
11      // Testing the incomplete test-function
12      @Test
13      public void loadHighscores(){
14          Highscore hs = new Highscore();
15          // Make sure list is loaded correctly.
16          assertNotNull(hs.getList());
17      }
18
19      @Test
20      public void isPlayerNameSet(){
21          Player player = new Player();
22          assertNotNull(player.getName());
23      }
24
25
26      @Test
27      public void isCorrectGuessIgnoreCasing(){
28          Game game = new Game( DEBUG: true);
29          game.setWord("test");
30
31          assertTrue(game.isCorrectGuess("T"));
32          assertTrue(game.isCorrectGuess("E"));
33          assertTrue(game.isCorrectGuess("S"));
34      }
35
36      @Test
37      public void isWrongGuessWhenAmountOfCharMismatch(){
38          Game game = new Game( DEBUG: true);
39          game.setWord("Test");
40
41          assertFalse(game.isCorrectGuess("Ta"));
42          assertFalse(game.isCorrectGuess("_^0000xas00"));
43      }
```

Screenshot 2



```
    public void isCorrectGuess(){
        Game game = new Game( DEBUG: true);
        game.setWord("Test");

        assertFalse(game.isCorrectGuess("ö"));
        assertFalse(game.isCorrectGuess("k"));
        assertTrue(game.isCorrectGuess("e"));
        assertTrue(game.isCorrectGuess("s"));
    }
    @Test
    public void isCorrectScore(){
        Game game = new Game( DEBUG: true);
        game.setWord("Test");
        game.isCorrectGuess("t");
        game.isCorrectGuess("e");
        game.isCorrectGuess("s");
        assertEquals( expected: 3, game.getScore());
        game = new Game( DEBUG: true);
        game.setWord("Hi");
        game.isCorrectGuess("h");
        game.isCorrectGuess("i");
        assertEquals( expected: 2, game.getScore());

    }

    @Test
    public void isRoundLostWhenOutOfAttempts(){
        Game game = new Game( DEBUG: true);
        game.setWord("Test");
        int MAX_ROUNDS = 7;
        for (int i=0; i < MAX_ROUNDS;i++){
            game.increaseAttempts();
        }
        assertTrue(game.isRoundLost());
    }

    @Test
    public void isRoundLost(){
        Game game = new Game( DEBUG: true);
        game.setWord("Test");
        int MAX_ROUNDS = 6;
        for (int i=0; i < MAX_ROUNDS;i++){
            game.increaseAttempts();
            assertFalse(game.isRoundLost());
        }
    }
GameTest › isWrongGuessWhenAmountOfCharMismatch()
```