Hang Man Project

Name: Ahmad Anbarje (aa224rm@student.lnu.se)

Date: 2019-03-31

Release in GitHub: https://github.com/Ahmadooof/aa224rm_1dv600/releases/tag/4.0

Logo

Contents

1 | Revision History

Date	Version	Description	Author
05/02/2019	1.0	Project Plan	Ahmad Anbarje
22/02/2019	1.0	Software Design	Ahmad Anbarje
08/03/2019	1.0	Software Testing	Ahmad Anbarje
22/03/2019	1.0	Final Project	Ahmad Anbarje

2 | General Information

Project Summary	
Project Name	Project ID
Hangman Game	1.0
Project Manager	Main Client
Ahmad Anbarje	For people above 13, who they are interested about the Hang man game.
Key Stakeholders	
End User, Project Manager, Teachers.	
Executive Summary	
We have made hangman game because it’s an assignment in the course, the game starts when the user tries to guess some characters to make a comprehensive word, there are limit numbers for the guesses, if the user has not guess the word, then will lose, otherwise the user wins.	

3 | Vision

The vision is to make a hangman game, we start with a texted based fashion, we will put three points in the first interface:

- Start the game
- Result
- Quit the game

In Start the game we have also question which is asking the user to guess a word which contains i.e 8 characters, the number of characters will be differ depends on the word itself, if the user doesn't guess the word after 8 tries, then will get message " worng, you did not guess the word" otherwise, will get "Right, You guess the word ". Then the user can navigate to the result to see how many times have tried and how many times have passed or fail for each try. The number of wrongs that the player can have is about eight parts which are used to hang the man (vertical pole, horizontal pole, head, body, left arm, right arm, left leg and right leg).

Reflections on Vision:

The vision is a good part with the project, we could let all the stakeholders see what the project will be about to make them also understand some parts which they need to think about while designing and testing some code.

The vision is also useful for the end user to realize how to play the game also to give an information about how many times could he try to guess for each word before loose or win. In addition, the vision is good for the project manager to let him/her think and design about the whole idea.

4 | Project Plan

We are going to plan for the hang man game, we also want to define the Stakeholders, first we have project manager who is responsible on the entire plan of the project even about the hardware problems. we have a developer who is responsible to create the hang man application, writing the code, in addition debug and execute the code. We also have a tester who is responsible to make unit tests, unit use cases to make the game works as it should be, finding bugs if exists. In addition, the end user who is going to play the game. We are going to finish the project by doing some iterations, each iteration represents steps about how to build the entire game.

First iteration:

- will be about plan the entire project,
- the deadline for this iteration will be on **Tuesday, 5 February 2019.**

Second iteration:

The goal of this iteration is to make a playable version of the hang man game. let us put some steps:

- add use cases for each scenario.
- adding a scenario about how to play the game.
- Make a UML diagram to help us to understand and visualize all the parts of the system life-cycle.
- we also going to add class diagram to realize the structure of our classes in the system.
- the deadline for this iteration will be on **Friday, 22 February 2019.**

Third iteration:

This iteration will be on software testing here we have steps as well:

- We are going to make manual test cases.
- Unit test for the important methods in the code.
- the deadline of this iteration will be on **Friday, 08 March 2019.**

Fourth iteration:

will be about preparing the entire project to be runnable, some features will be added, we will focus on the documentation of this iteration to be as clear as possible.

- The deadline of this will be on **Friday, 22 March 2019.**

4.1 Introduction

The objective of this project is to make a simple hangman game with java programming language within 2 months.

4.2 Justification

We are doing hangman game because it is a project assignment in software technology course.

4.3 Stakeholders

Project Manager: who is responsible on the plan of the project.

Developer: who is going to implement the plan of the project, also is responsible to execute and debug the code.

Tester: who is responsible on the unit testing, finding the bugs.

End User: the person who is going to play the game.

4.4 Resources

Software engineering book 10 Edition.

4.5 Hard and Software Requirements

We are using Eclipse IDE to develop our project, we could run our project with any system which has Java development kit and Java runtime environment.

4.6 Overall Project Schedule

First Iteration: Tuesday, 5 February 2019.

Second Iteration: Friday, 22 February 2019.

Third Iteration: Friday, 08 March 2019.

Fourth Iteration: Friday, 22 March 2019.

4.7 Scope, Constraints and Assumptions

The scope: This game will be playable from Eclipse IDE or any different IDE, we will not make design for the game, so it will be a texted fashion game. This game is not a web application, it is just a console application. We will put a feature in the game which is the result, so the user after playing many rounds, could navigate to the result to see how many times Win and lose for each round.

Constraints: We could take longer time for each stage after the deadline because of the short time of the project.
Assumptions: the user should know how to run the game by any IDE, also the user should have JDK and JRE on his operating system to run the game.

Reflection on Project Plan:
The project plan is useful for the stakeholders
Even for developer who are going to add new features to the project, but I think It is difficult to manage all the ideas and features inside the project plan, so the Best think is that we need to focus on the time for each iteration and do our best To finish as we are planning to.

5 | Iterations

Iteration 5.1

It contains the project plan and how we are going to create the entire project, we will focus on the deadline for each iteration to make everything planed as possible, also it contains skeleton code of our project.
The deadline for this iteration on **Tuesday, 5 February 2019.**
The resources we going to use is the book, reading chapter **2,3,22,23.**

ID	Description	Estimated Time	Actual Time	Dead Line
D1	Documentation of the game	20 hours	26 hours	**Tuesday, 5 February 2019**
D2	Implementing skeleton code	2 hours	1 hour	**Tuesday, 5 February 2019**

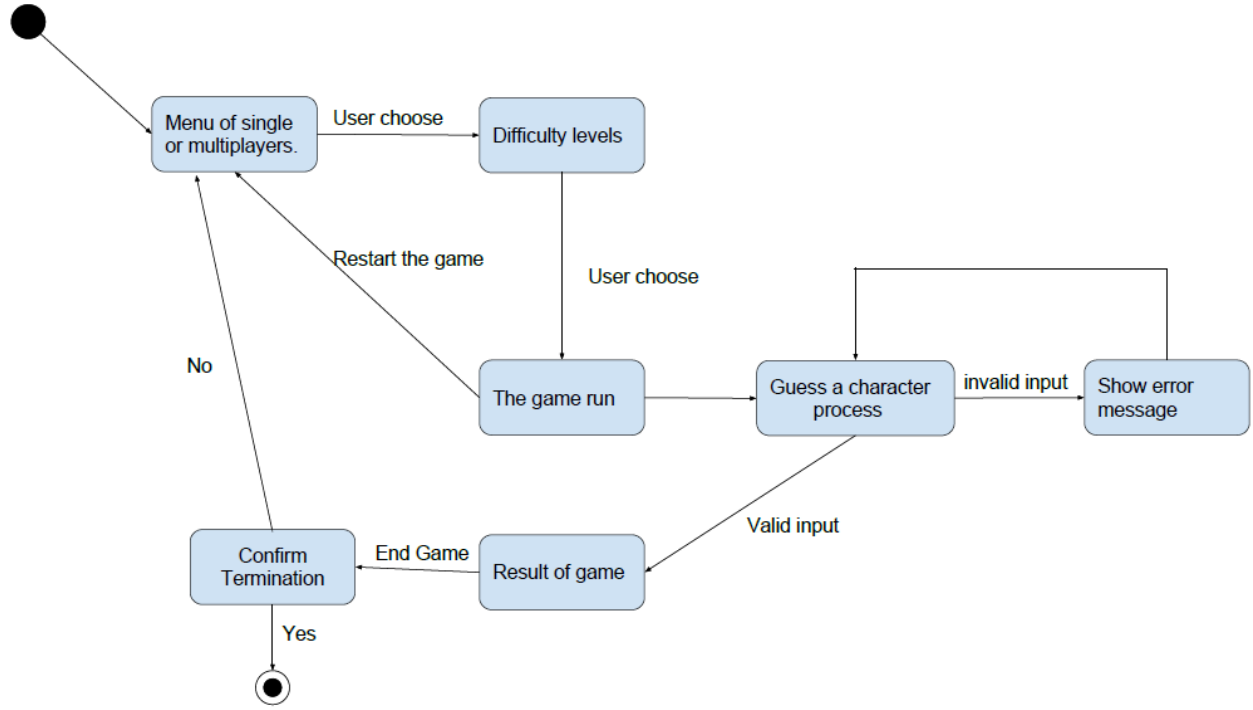
Iteration 5.2

This iteration will contain the following:

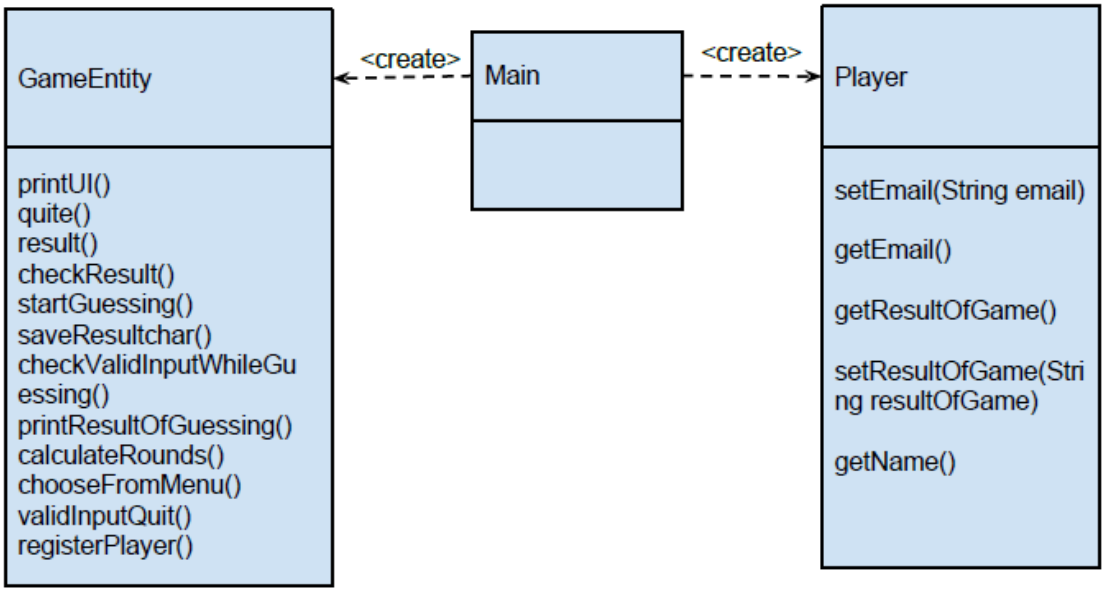
- State machine diagram.
- Class diagram.
- Play game scenario.
- Use case diagram.
- Time Log.

The game should be playable in this iteration.
The deadline for this iteration on **22 February 2019.**
The resources we going to use is the book, reading chapter **6,7,15.**

State Machine diagram:



Class Diagram:



Play game scenario:

Precondition: The user starts the game

Postcondition: The user chose a setting for a game and start to play.

Main scenario

1. Starts when the user wants to choose a single player or multiplayer game option.
2. The system presents a menu with a difficulty options.
3. The user chose a difficulty.
4. The system starts the game.
5. The user input character to guess the word.
6. The system shows result of guessing and ask for next guess.
7. The user input character again until the rounds finish.
8. The system shows the result, if the user win or lose.

Alternative scenarios

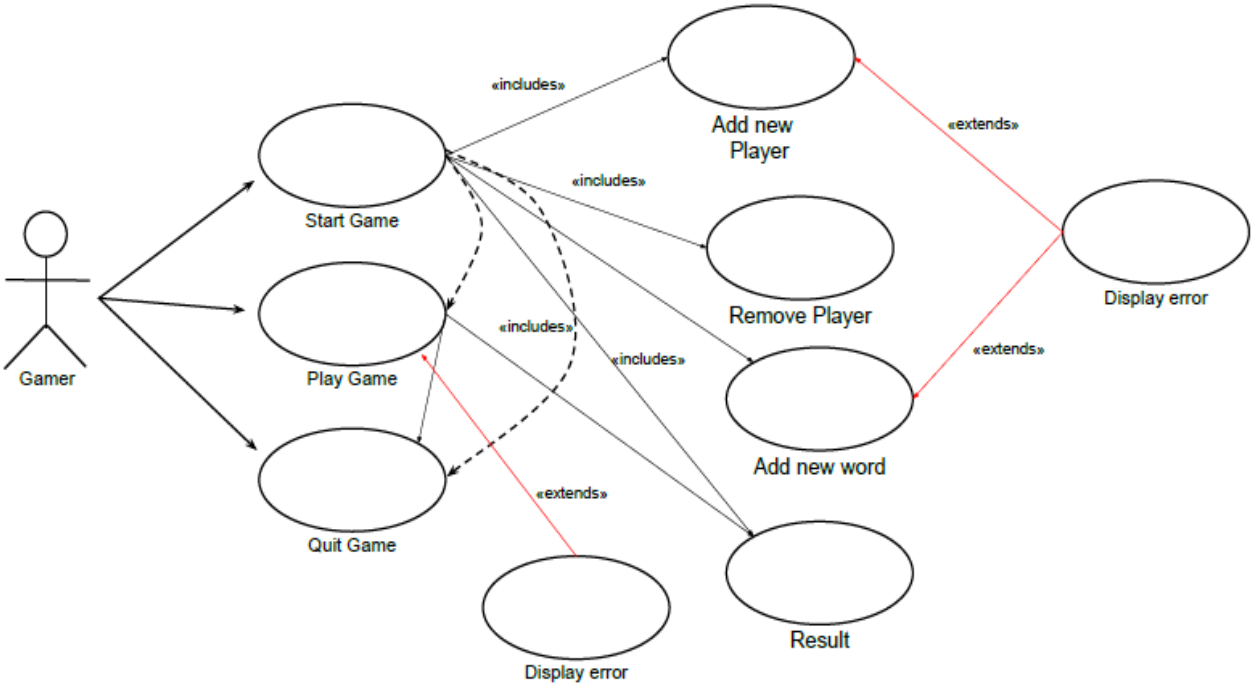
4.1 The Gamer makes the choice to quit the game.

1. The system quits the game.

5.1 The user pressed invalid character, while guessing the word.

1. The system shows a message that is invalid character.
2. The system does not count the round which it's invalid input.
3. The system asks again to input a valid character.

Use case diagram:



Time Log:

ID	Description	Estimated Time	Actual Time	Dead Line
D1	use case diagram	6 hours	10 hours	22 February 2019
D2	play game scenario	4 hours	7 hours	22 February 2019
D3	State machine diagram	2 hours	4 hours	22 February 2019
D4	Implementing user interface	4 hours	6 hours	22 February 2019
D5	Implementing player registration	3 hours	4 hours	22 February 2019
D6	Implementing result of guessing	10 hours	13 hours	22 February 2019
D7	class diagram	3 hours	2 hours	22 February 2019

Iteration 5.3

The objectives of testing, description of testing these objects.

- The objectives of the testing in this iteration is to test the code that was implemented the last iteration.
- We are going to test two use-cases by running dynamic manual two test cases for each. We are going to write automated unit tests that are testing two methods used in the **GameEntity** class in the class **GameEntityTest**.
- I have tested these methods because it is important to make the game work as it should be.
- I did not test the setters and getters methods because the IDE can generate these methods, so I don't think that will be a bug with generating it.

Time Plan

task	Estimated Time	Actual Time
Manual TC	2 hours	5 hours
Unit Tests	3 hours	6 hours
Running manual tests	30 minutes	2 hours
Code inspection	1 hour	4 hours
Test Report	30 minutes	30 minutes

Use-Cases

This is a simple application with two use-cases.

UC1 check user input successful while playing the game

Precondition: the user chose from menu to start the game.

Postcondition: the user has input a character.

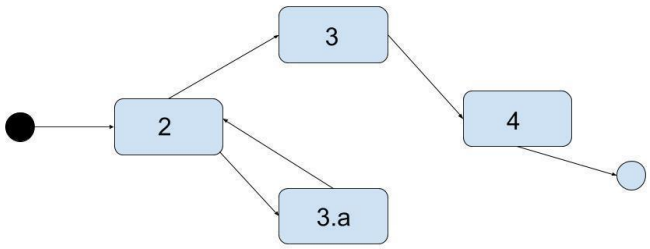
Main scenario

1. Starts when the user wants to guess a letter
2. The system asks to press a letter.
3. The user provides arbitrary letter.
4. The system stores the letter and continue to present new guessing.

Alternate scenarios

- 3a the user provides a number instead of letter.
- The system shows an error message.
- Enter again a valid input.

Activity diagram



Manual Test-Cases

TC1.1 User made a valid input

Use-case: UC1 check user input.

Scenario: check user input successful.

The main scenario of UC1 is tested where a user made a valid input.

Precondition: start the game.

Test steps

- Start the game
- System shows "guess a word by enter a letter”
- Enter a letter “h” and press enter.

Expected

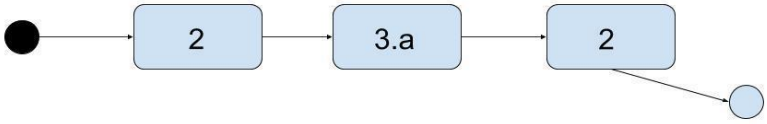
- The system should show if “h” is a letter which belongs to the word or not.
- The system continues with asking the user to guess next letter.

TC1.2 user made an invalid input

Use-case: UC1 check user input.

Scenario: invalid input force to input again.

The alternate scenario where the user enters a number instead of letter and is forced to enter a valid character again.



Precondition: the game already started.

Test steps

- Start the game
- System shows "guess a word by enter a letter”
- Enter a number “5” and press enter.

Expected

- The system should show “Invalid input”.
- System shows "please enter a letter to guess the word: " and waits for input.

UC2 check user input to quit the game

Precondition: the user has finished the game.

Postcondition: the user will decide to quit the game or not.

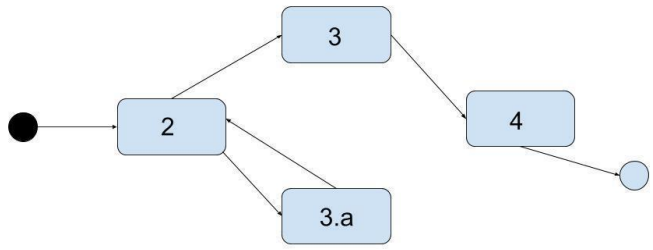
Main scenario

1 Starts when the game is finished.
2 The system asks the user to quit.
3 The user provides a choice to quit.
4 The system finished the game.

Alternate scenarios

- 3a the user provides an invalid input to quit.
- The system shows an error message.
- Enter again a valid input to quit or not.

Activity diagram

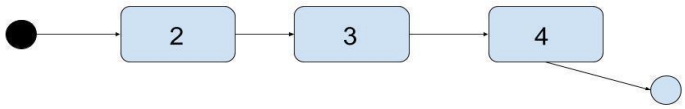


Manual Test-Cases

TC1.2 User made a valid input to quit the game

Use-case: UC2 check user input.

Scenario: check user input successful.



The main scenario of UC2 is tested where a user made a valid input to quit the game.

Precondition: the game is finished.

Test steps

- Finish the game
- System shows menu
    - "1. Restart the game"
    - "2. Show Result"
    - "3. Quit the game".
- Enter a number "3" to quit the game and press enter.

Expected

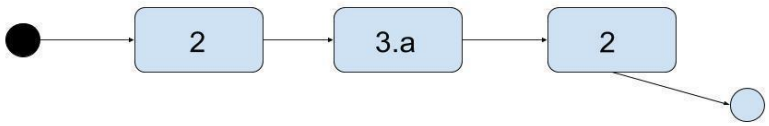
- The system has finished the game.

TC2.2 user made an invalid input to quit the game.

Use-case: UC2 check user input to quit the game.

Scenario: invalid input force to input again.

The alternate scenario where the user enters an invalid number and is forced to enter a valid number again.



Precondition: the game is finished.

Test steps

- Finish the game
- System shows menu
    - "1. Restart the game"
    - "2. Show Result"
    - "3. Quit the game".
- Enter a number "5" and press enter.

Expected

- The system should show "Invalid input"
- System shows "please choose a valid number from the menu" and waits for input.

Unit Test

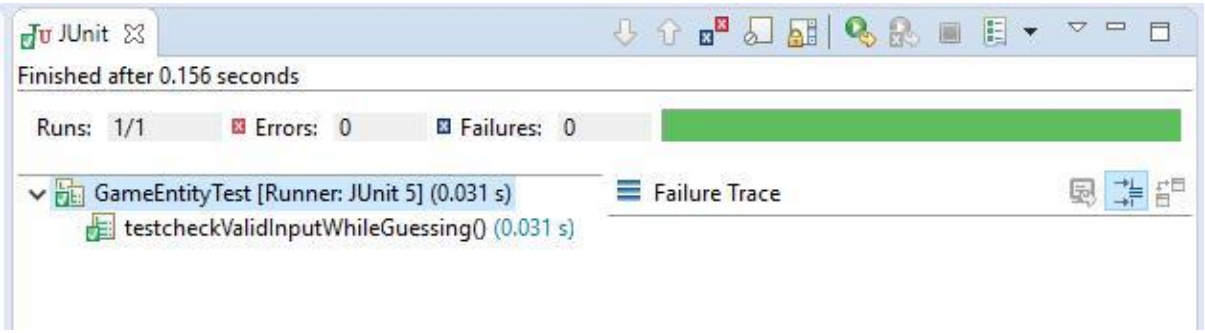
Source Code 1:

```
protected boolean checkValidInputWhileGuessing(String userInput) {
    if(userInput.length() == 0)
        return false;
    userCharInput = userInput.charAt(0);
    isLetter = Character.isLetter(userCharInput);
    if(userInput.length() == 1 && isLetter)
    {
        return true;
    }
    else {
        return false;
    }
}
```

Unit Test for Code 1:

```
class GameEntityTest {

    private String validInput = "f";
    private String inValidInput = "3";
    private String manyCharsInvalid = "sdfsdfgvsdvdsv";

    private String validInputToQuit = "3";
    private String notValidInputToQuit = "4";
    private String ManyCharactersInputToQuit = "sadfiojasdf";
    private String emptyString = "";

    private int ExpectedResultOfRounds = 6;  // all rounds are 7
    private char userInput = 'å';
    private String randomWord = "ahmad";

    GameEntity game = new GameEntity();

    @Test
    void testcheckValidInputWhileGuessing() {
        assertEquals(true,game.checkValidInputWhileGuessing(validInput));
        assertEquals(false,game.checkValidInputWhileGuessing(manyCharsInvalid));
        assertEquals(false,game.checkValidInputWhileGuessing(inValidInput));
        assertEquals(false,game.checkValidInputWhileGuessing(emptyString));
    }
```

Result of Code 1:



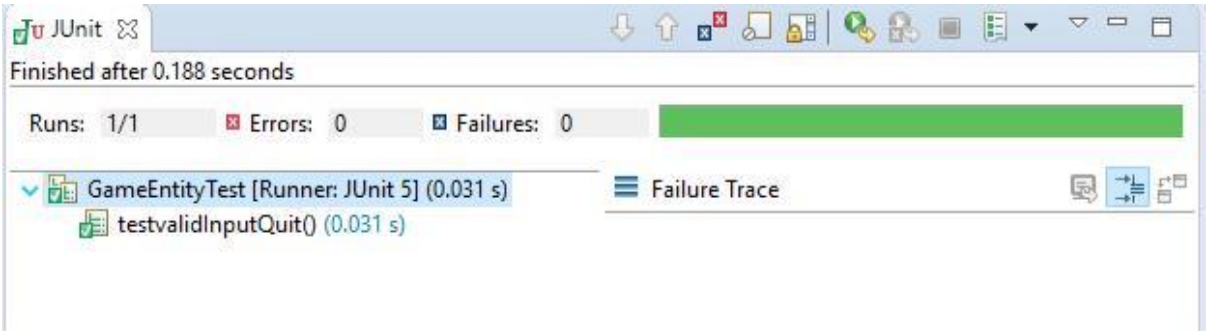
Source Code 2:

```
protected boolean validInputQuit(String choosenNumber) {
    if(choosenNumber.length() == 0)
        return false;
    if(choosenNumber.equals("3"))
        return true;
    else
        return false;
}
```

Unit Test for Code 2:

```
class GameEntityTest {

    private String validInput = "f";
    private String inValidInput = "3";
    private String manyCharsInvalid = "sdfsdfgvsdvdsv";

    private String validInputToQuit = "3";
    private String notValidInputToQuit = "4";
    private String ManyCharactersInputToQuit = "sadfiojasdf";
    private String emptyString = "";

    private int ExpectedResultOfRounds = 6;  // all rounds are 7
    private char userInput = 'å';
    private String randomWord = "ahmad";

    GameEntity game = new GameEntity();

    @Test
    void testvalidInputQuit() {
        assertEquals(true,game.validInputQuit(validInputToQuit));
        assertEquals(false,game.validInputQuit(notValidInputToQuit));
        assertEquals(false,game.validInputQuit(ManyCharactersInputToQuit));
        assertEquals(false,game.validInputQuit(emptyString));
    }
```

Result of Code 2:



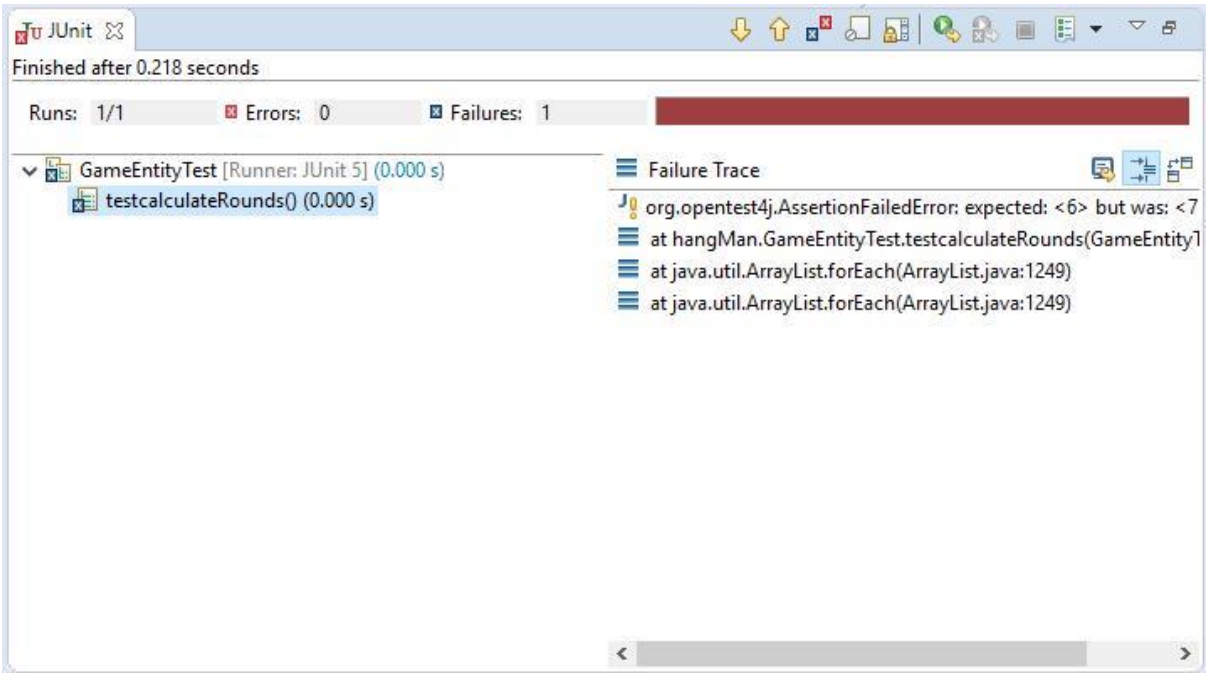
Source Code 3:

```
protected int calculateRounds(char userInputChar,String randomWord) {
    isEqual = false;
    if(userInputChar == 'å')
        userInputChar = 'a';
    for(int index = 0; index < randomWord.length() ; index++)
    {
        Character ch = randomWord.charAt(index);
        if(ch.equals(userInputChar)) {
            isEqual = true;
        }
    }
    if(!isEqual)
        return countRounds--;
    else
        return countRounds;
}
```

Unit Test for Code 3:

```
class GameEntityTest {

    private String validInput = "f";
    private String inValidInput = "3";
    private String manyCharsInvalid = "sdfsdfgvsdvdsv";

    private String validInputToQuit = "3";
    private String notValidInputToQuit = "4";
    private String ManyCharactersInputToQuit = "sadfiojasdf";
    private String emptyString = "";

    private int ExpectedResultOfRounds = 6;  // all rounds are 7
    private char userInput = 'å';
    private String randomWord = "ahmad";

    GameEntity game = new GameEntity();

    // this method return 7 (rounds) if the player has guessed the letter,
    // otherwise it reduce the rounds by one
    // the bug is when the player input 'å'  so,
    // the program understand that's is similar to 'a' but it should not be the same.
    @Test
    void testcalculateRounds() {
        // no bug
        assertEquals(7,game.calculateRounds('a',randomWord));

        // here is the bug test
        assertEquals(ExpectedResultOfRounds,game.calculateRounds(userInput,randomWord));

    }
```

Result of Code 3:



Test Report

Test traceability matrix and success

Test	UC1	UC2
TC1.1	1/OK	0
TC1.2	1/OK	0
TC2.1	0	2/Ok
TC2.2	0	2/Ok
COVERAGE & SUCCESS	2/OK	2/Ok

Automated unit test coverage and success

Test	Console View	GameEntity Controller	Main	Game Entity
`testvalidInputQuit`	0	0	0	100%/OK
`testcheckValidInputWhileGuessing`	0/NA	0/NA	0/NA	100%/OK
`testcalculateRounds`	0	0	0	50%/Ok
COVERAGE & SUCCESS	0/NA	0/NA	0/NA	75%/Ok

Reflection:

In this testing of the game I learned a lot of stuff. I improved a lot of skills in coding, and good way to learn how to make a manual and unit testing for the use cases. It is good idea to start project in software with a console game. But I didn't find simple at the first, especially when I reach to the point to find a bug. Even I feel the explanation is difficult for me, but it was a good try at the end. I think it is little hard to test an input from keyboard as testing method, so I feel the code is little bad. But at the end we learn new stuff and we reach our goal.

5.4 Iteration 4:

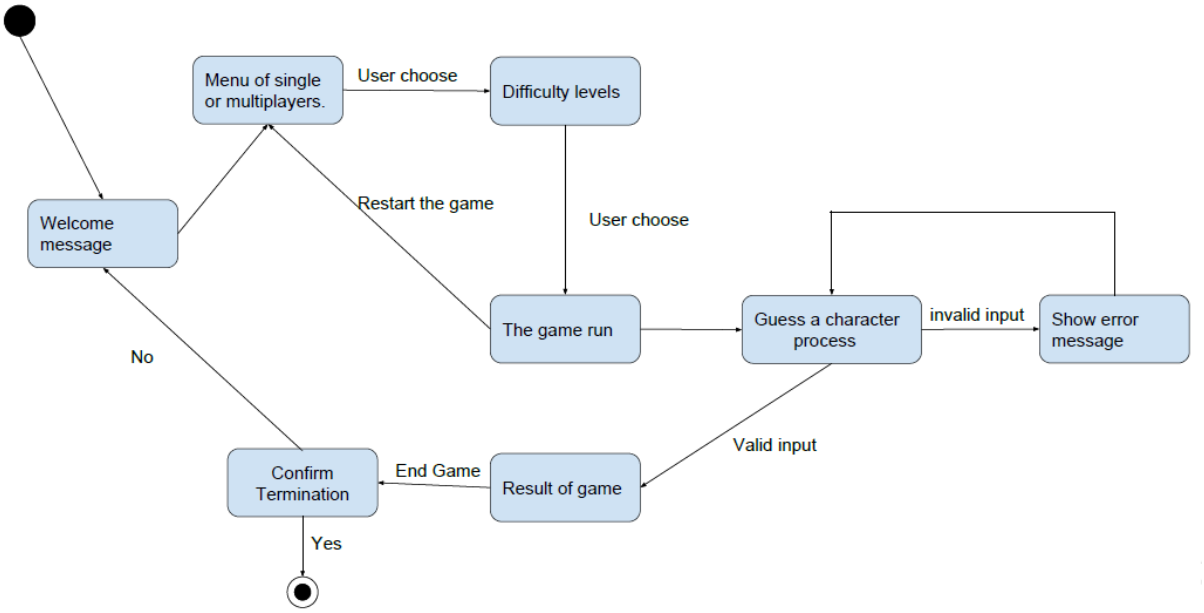
This iteration will be about making good looking documentation, testing, implementation for the entire project. In addition, adding some features for the game, we will put the steps:

1. **State machine diagram.**
2. **Class diagram.**
3. **Welcome massage scenario.**
4. **Manual test case.**
5. **Unit test**
6. **Test report.**
7. **Risk analysis.**
8. **Strategies.**
9. **Time Log.**

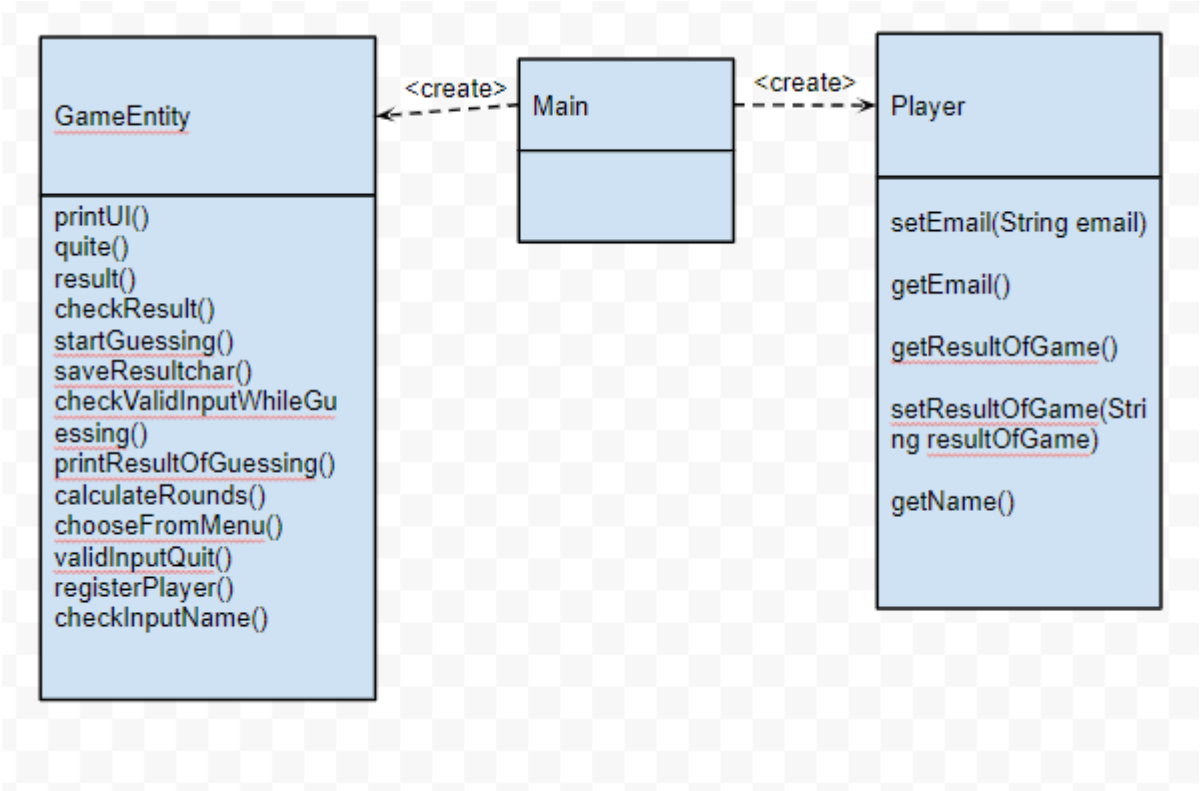
The deadline for this iteration on **Friday, 22 March 2019.**

Note: we do not provide a use case diagram because we done that in iteration 2, thus it will be the same diagram.

4.1 state machine diagram



4.2 Class diagram:



4.3 Welcome massage scenario:

Precondition: The player starts the game.

Postcondition: welcome massage shown to the player.

Main scenario

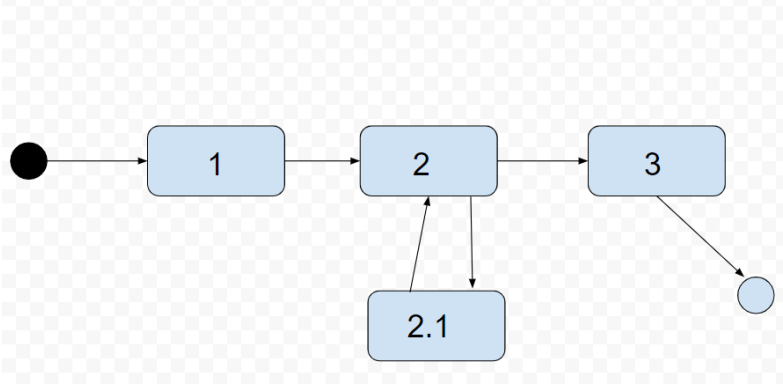
1. System shows a message to enter a name.
2. Player enters letters providing a name.
3. System shows welcome massage.

Alternative scenarios

2.1 The player input a number instead of letters.

- The system shows invalid input massage.
- The system shows a massage to enter name again.

Activity diagram



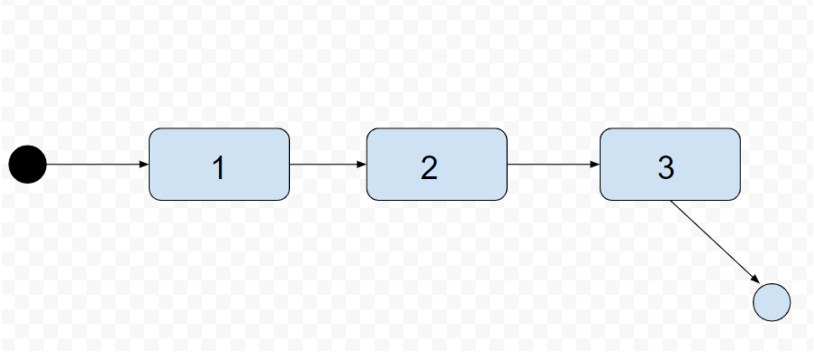
4.4 Manual test case:

TC1.1 User made a valid input

Use-case: check user input.

Scenario: welcome message scenario.

The main scenario is tested where a user made a valid input.



Precondition: start the game.

Test steps

- Start the game
- System shows "please enter your name”
- Player input “Fredrik” and press enter.

Expected

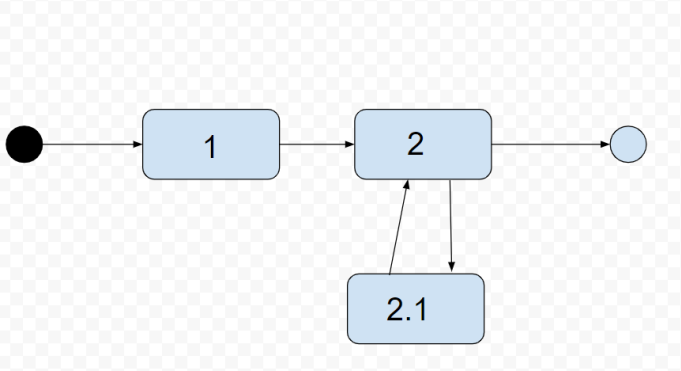
- The system should show “Welcome Fredrik” and continue the game.

TC1.2 user made an invalid input

Use-case: check user input.

Scenario: invalid input force to input again.

The alternate scenario where the user enters a number instead of letters and is forced to enter a valid character again.



Precondition: the game already started.

Test steps

- Start the game
- System shows "please enter your name”
- Player input a number ”5” and press enter.

Expected

- The system should show ”Error input”.
- System shows "please enter your name" and waits for input.

4.5 Unit test

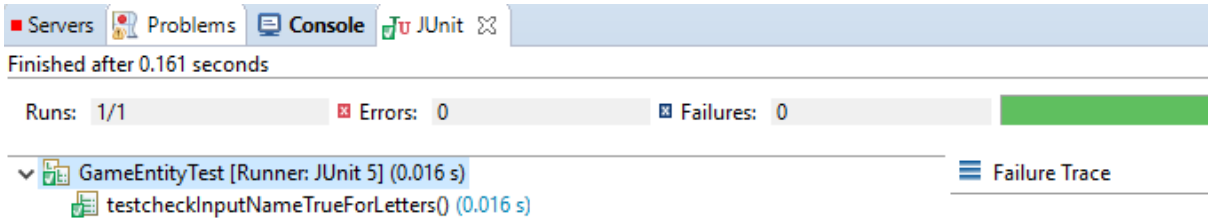
Source code 1:

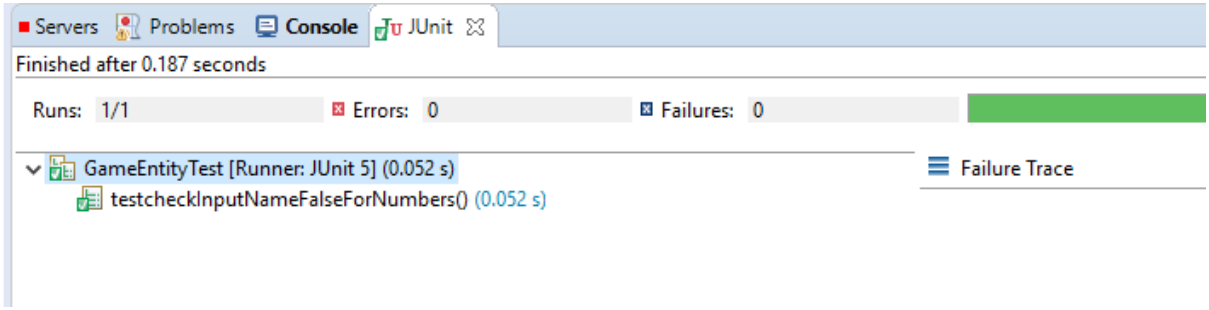
```
@SuppressWarnings("static-access")
public boolean checkInputName(String name) {
    for(int i = 0;i<name.length();i++) {
        Character x = name.charAt(i);
        if(!x.isLetter(x))
            return false;
    }
    return true;
}
```

Unit test for code 1:

```
@Test
void testcheckInputNameTrueForLetters() {
    assertEquals(true, game.checkInputName("Fredrik"));
    assertEquals(true, game.checkInputName("whatEverLettersAreOk"));
}

@Test
void testcheckInputNameFalseForNumbers() {
    assertEquals(false, game.checkInputName("Fredrik123"));
    assertEquals(false, game.checkInputName("whatEverLettersAreOk3254"));
    assertEquals(false, game.checkInputName("2"));
}
```

Result of testing source code 1:





Unit test for code 1 contains Bug:

```
@Test
void testcheckInputNameWithBug() {
    // this should return true because, all of string are letters
    // but we have made this bug because of the requirements.
    assertEquals(true, game.checkInputName("Fredrik Fredo"));
}
```

Result of unit test for code 1 contains Bug:

4.6 Test report

Test traceability matrix and success:

Test	UC1	UC2
TC1.1	1/OK	0
TC1.2	1/OK	0
COVERAGE & SUCCESS	2/OK	2/Ok

Automated unit test coverage and success:

Test	Console View	GameEntity Controller	Main	Game Entity
`testcheckInputNameTrueForLetters()`	0	0	0	100%/OK
`testcheckInputNameFalseForNumbers()`	0/NA	0/NA	0/NA	100%/OK
`testcheckInputNameWithBug()`	0	0	0	50%/Ok
COVERAGE & SUCCESS	0/NA	0/NA	0/NA	75%/Ok

4.7 Risk analysis:

ID	Description	Impact	probability
R1	sickness	5/5	2/5
R2	Hard disk problems	4/5	2/5
R3	Viruses	5/5	2/5
R4	Software Failures	4/5	2/5

4.8 Strategies:

We need to think about some strategies to avoid the risks which we could have.

Sickness Issue:

We need to protect our self as much as possible, by taking some vitamins, washing hands before eating food, don't share personal items with another people, avoid touching wild animals.

Hard disk problems Issue:

We could protect the hard disk by reduce the data load, monitor drive health, manage drive life cycle.

Viruses Issue:

We could protect our machine by keep our software's up to date, don't click links which comes from junk emails, make a backup plan each period of time, use a firewall.

Software Failures Issue:

It is difficult to prevent this kind of failures because, it is almost impossible to use a software which has no bug, but the best thing we can do is to make all software's up to date.

Reflection of Risk Analysis:

The risk analysis is useful to prevent our project to be failed, we need to make a good plan for the risks, analysis them, and present them to the stakeholders, to make all the team have a good idea about the possible risks, also, to protect them which will protect our project to get fail or damaged. So, we need to analysis good strategies for the risks which we could face in daily life or in software and hardware issues, so we need to deal seriously with it as well.

4.9 Time Log:

	Estimated Time	Actual Time	Dead Line
Check and fix what we have done in the project before.	4 hours	4 hours	**Friday, 22 March 2019.**
State machine diagram	2 hours	2 hours	**Friday, 22 March 2019.**
Class diagram	1 hour	2 hours	**Friday, 22 March 2019.**
New feature scenario	3 hours	2 hours	**Friday, 22 March 2019.**
Manual test case	6 hours	2 hours	**Friday, 22 March 2019.**
Implement new feature	3 hours	4 hours	**Friday, 22 March 2019.**
Unit test	4 hours	3 hours	**Friday, 22 March 2019.**
Test report	3 hours	1 hour	**Friday, 22 March 2019.**
Risk analysis	2 hours	3 hours	**Friday, 22 March 2019.**
Strategies	2 hours	1 hour	**Friday, 22 March 2019.**