Component Based Software Engineering

**AsteroidsFX**

**Course:**

- T510035101

**Examiner:**

- Jan Corfixen Sørensen

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Student:**

- Marcus Brink Thomsen

**Exam Nr:**

- 4118643

**Github username:**

- VinciDa33

**Project repository:**

- <https://github.com/VinciDa33/AsteroidsFX_MBT>



# Abstract

Describe the problem that the report addresses in context of the game domain.

Outline how the developed game addresses the requirement – its key characteristics and fundamental principles (establishing a solution).

Contents

[Abstract 1](#_Toc163746894)

[Introduction 3](#_Toc163746895)

[Requirements 4](#_Toc163746896)

[Analysis 5](#_Toc163746897)

[Design 6](#_Toc163746898)

[Implementation 7](#_Toc163746899)

[Test 8](#_Toc163746900)

[Discussion 9](#_Toc163746901)

[Conclusion 10](#_Toc163746902)

[References 11](#_Toc163746903)

# Introduction

The introduction must describe the game.

This project aims to mimic the classic asteroid game by Atari from 1979, using the Java language. The game consists of a simple playing field with a player character, enemy spaceships and asteroids. Both the player and enemies have the ability to shoot, splitting asteroids in two or destroying each other.

The game loop is potentially infinite, with the goal of getting the highest score possible. The difficulty of the game goes up over time, with more enemies and asteroids spawning. Each time an asteroid is split or destroyed, as well as when an enemy perishes, points are awarded to the player’s overall score.

The project uses JavaFX as its graphics framework, and game elements are rendered using simple polygon shapes with flat colours.

The game is created using Java with Maven as build system, with a focus on the open/closed principle. Using the Java Platform Module System, game elements are split out into separate modules. This modular approach allows for extension modules to be integrated at a later point, without the need for recompilation of the core system.

# Requirements

Describe the component-based game in terms **of interface contracts**, functional and non-functional requirements.   
The game must include **Player**, **Enemy**, Asteroids, **Weapon** and Map components.

The Player, Enemy and Weapon components must implement service provided interfaces that allow the components to be updated and removed without recompilation.

|  |  |  |
| --- | --- | --- |
| **Functional requirements** | | |
| **ID** | **Title** | **Description** |
| F1 | Service interfaces | The project must contain service interfaces for module interactions |
| F2 | Player module | The project must contain a player module |
| F3 | Enemy module | The project must contain an enemy module |
| F4 | Asteroid module | The project must contain an asteroid module |
| F5 | Bullet system | The project must contain a system for spawning and handling bullets |
| F6 | Pythagorean collision detection | The project must have a collision system using the Pythagorean theorem to check for circular collisions |
| F7 | Scoring system microservice | The project must have a microservice to handle the point score |
| F8 |  |  |
| F9 |  |  |

|  |  |  |
| --- | --- | --- |
| **Non-functional requirements** | | |
| **ID** | **Title** | **Description** |
| FN1 | Unit testing | The game should have at least 1 unit test |
| FN2 | Modularity | It must be possible to remove non-common modules without recompilation |
| FN3 | Portability | The game should run similarly on hardware with varying specs  (Game elements should move at similar speeds on different hardware) |
| FN4 |  |  |
| FN5 |  |  |

# Analysis

Analysis describes only **what** the system should do and not **how** it is done.

In analysis, you can come up with a rough draft of the interfaces and the entities of the game.

Furthermore, you should document use cases/gameplay, the object model using a UML class diagram and the communication between components with sequence diagrams.

# Design

The design describes **what** the structure of the system should be to fulfill the requirements.

Document the architecture and abstractions of the system.

Design develops those abstractions into realizable components.

Describe and sketch the **component models** of the game using a UML component diagram, see [[UML]](https://mcas-proxyweb.mcas.ms/certificate-checker?login=false&originalUrl=https%3A%2F%2Fyoutu.be.mcas.ms%2FKQUGFFN4M90%3FMcasTsid%3D15600&McasCSRF=e3d0355ef58220a2ce832c3102938d848870c939fdd45a3fc240b2c2cf9cca75) .

The component contracts in the system must be described in terms of pre- and post-conditions.

Furthermore, the different elements of the game and how they are connected must be described.

# Implementation

In implementation, you document the implementation (code) of the components from design.

Describe the details of how the components are registered and accessed.

How are reliable dependencies and strong encapsulation enforced in your project?

What component models are applied and where in the source code?

Provide a descriptive explanation of each element in the implementation and provide arguments for your choices.

You should describe how you register components and how you access them.

Note, you are allowed to reuse the game logic, but you must implement the **[GameLab], [JavaLab]**, **[JPMSLabs]**, **[SpringLab],** **[TestLab]** and [MicroServiceLab] labs based on your own GitHub branches.

# Test

Describe how experimental validation was performed through deployment of the game on top of the component container in a real setting.

Test the system's software-abilities such as dynamic updates using integration and unit test.

# Discussion

Discuss how well the game solved the identified essential problems (module updates etc.).

To which extent did your design meet the requirements?

# Conclusion

First summarize the report.

Remember that you are summarizing the report for a reader that has read the introduction and the body of the report already and has a strong sense of key concepts and applied technologies.

Explain the potential impacts of your system in relation to the main issue.

Direct future work directions related to the main issue.

However, this should not be seen as an opportunity to develop new ideas in significant detail and should be clearly linked to the work described in your report.

# References