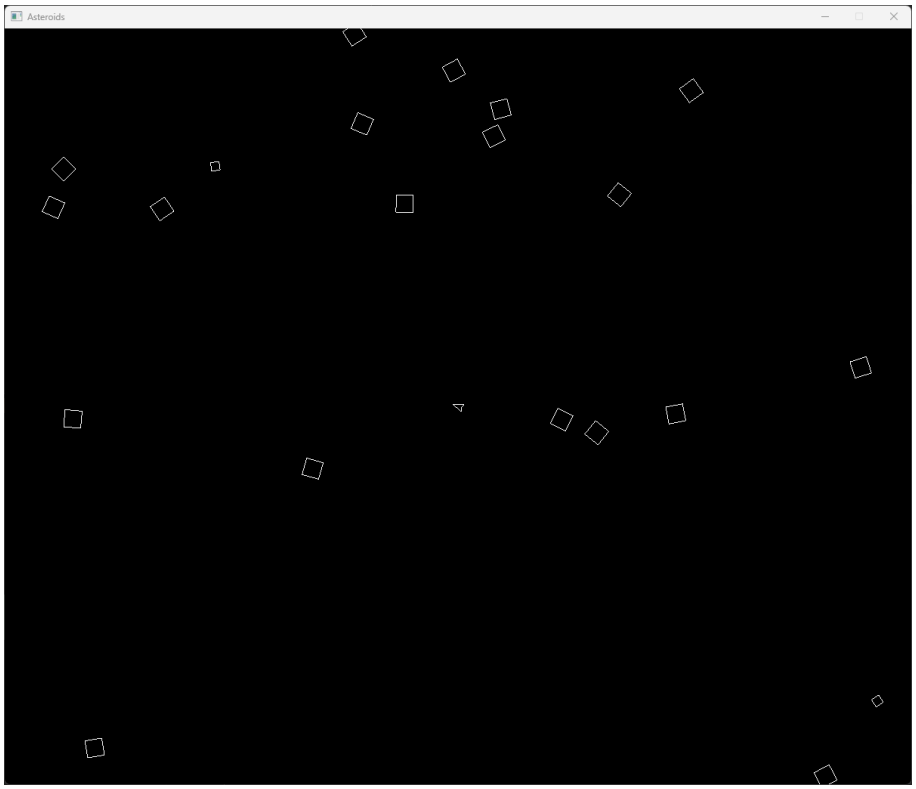
A component-based game

Asteroids

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).



Magnus Christian Larsen
Exam Number: XXXX
GitHub: Autowinto
Git Repo: https://github.com/Autowinto/SB4-KOM

Contents

1 Introduction **2**

2 Requirements **3**
2.1 Functional Requirements . . . 3
2.2 Non-Functional Requirements . . . 3
2.3 Interfaces . . . 3

3 Analysis **4**
3.1 Rough Interfaces . . . 4

4 Design **5**
4.1 UML Diagrams . . . 5
4.1.1 Component . . . 5
4.1.2 Class . . . 5
4.1.3 Sequence . . . 6
4.2 Component Contracts . . . 6
4.2.1 IGamePluginService . . . 6
4.2.2 IEntityProcessingService . . . 7
4.2.3 IPostEntityProcessingService . . . 7
4.2.4 IBulletFactory . . . 7
4.2.5 EntityPart . . . 8
4.3 Elements of the game . . . 8

5 Implementation **9**

6 Test **11**

7 Discussion **12**
7.1 Requirements Validation . . . 12

8 Conclusion **13**

1 Introduction

The introduction must describe the game.

2 Requirements

The requirements specifications for this project, are based on the formal requirements of the lab exercises from the course and don't veer too far from

2.1 Functional Requirements

F01	The game must include a player component.
F02	The game must include an enemy component.
F03	The game must include an asteroid component.
F04	The game must include a weapon component.

Table 1: Functional Requirements

As can be seen on Table 1

2.2 Non-Functional Requirements

F01	The game must include a player component.
F02	The game must include an enemy component.
F03	The game must include an asteroid component.
F04	The game must include a weapon component.

Table 2: Functional Requirements

2.3 Interfaces

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

3 Analysis

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

3.1 Rough Interfaces

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.

4 Design

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

4.1 UML Diagrams

4.1.1 Component

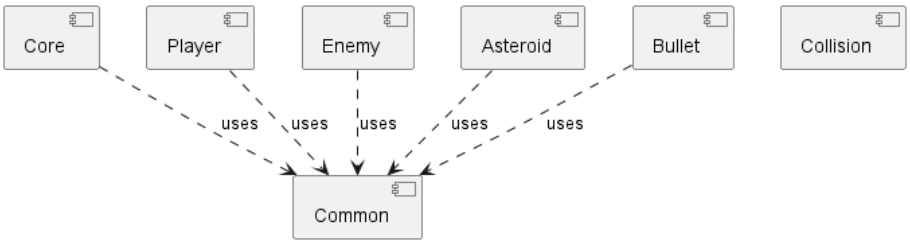


Figure 1: A UML Component diagram over the project

4.1.2 Class

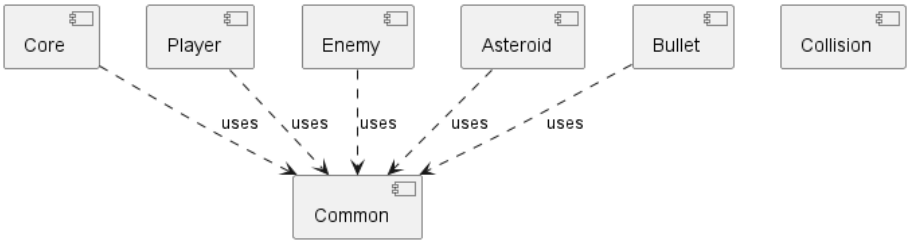


Figure 2: A UML Class diagram over the project

4.1.3 Sequence

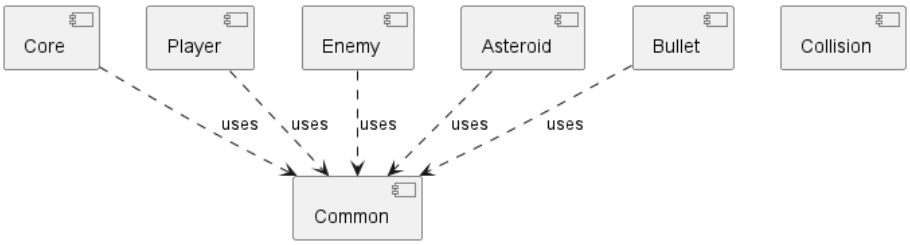


Figure 3: A UML Sequence diagram over the project

4.2 Component Contracts

4.2.1 IGamePluginService

The IGamePluginService interface, is implemented by the plugins that are used to initialize entities handled by the different services. This interface has two operations. One for starting the plugin and one for stopping the plugin.

Operation:	start(GameData gameData, World world)
Description:	Initializes the components' entities
Parameters:	Text
Preconditions:	text
Postconditions:	text

Figure 4: Component contract over the functions of the IEntityProcessingService interface.

Operation:	stop(GameData gameData, World world)
Description:	Cleans up entities that were initialized by the components
Parameters:	text
Preconditions:	text
Postconditions:	text

Figure 5: Component contract over the functions of the IEntityProcessingService interface.

4.2.2 IEntityProcessingService

The IEntityProcessingService interface is implemented by components' services, so that they can process and handle game logic every frame.

Operation:	process(GameData gameData, World world)
Description:	Processes game logic
Parameters:	text
Preconditions:	text
Postconditions:	text

Figure 6: Component contract over the functions of the IEntityProcessingService interface.

4.2.3 IPostEntityProcessingService

The IEntityProcessingService interface is, like the IEntityProcessingService, implemented by components' services, so that they can process and handle game logic every frame. The IPostEntityService, however, always processes after each IEntityProcessingService has finished processing.

Operation:	process(GameData gameData, World world)
Description:	Processes game logic.
Parameters:	text
Preconditions:	text
Postconditions:	text

Figure 7: Caption

4.2.4 IBulletFactory

The IBulletFactory is an interface implemented by classes that need implementation logic specifically for spawning bullets.

Operation:	create(Entity sourceEntity, GameData gameData)
Description:	Creates the bulletfactory, sets the source entity and returns a new bullet to spawn.
Parameters:	text
Preconditions:	text
Postconditions:	text

Figure 8: Caption

4.2.5 EntityPart

The EntityPart is an interface implemented by the different parts that can be attached to an entity to extend the logic of said entity.

Operation:	process(GameData gameData, Entity entity)
Description:	Processes game logic to do with the entity that said part is attached to.
Parameters:	text
Preconditions:	text
Postconditions:	text

Figure 9: Caption

4.3 Elements of the game

Describe the different elements of the game and how they are connected. Maybe do this together with component diagram?

5 Implementation

In implementation, you document the implementation (code) of the components from design. Describe the details of how the component 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.

```
LwjglApplicationConfiguration cfg =
    new LwjglApplicationConfiguration();

cfg.title = "Asteroids";
cfg.width = 500;
cfg.height = 400;
cfg.useGL30 = false;
cfg.resizable = false;

new LwjglApplication(new Game(), cfg);
```

Figure 10: Code from the Main.java file in the core module, as it looked when using the LWJGL2 backend.

Since the base example of the AsteroidsEntityFramework project made use of the LWJGL2 backend as opposed to the LWJGL3 backend for handling rendering and audio from LibGDX, an issue arose, where the example was possible.

```
Lwjgl3ApplicationConfiguration cfg =
    new Lwjgl3ApplicationConfiguration();

cfg.setTitle("Asteroids");
cfg.setResizable(false);
cfg.setWindowedMode(1200, 1000);

new Lwjgl3Application(new Game(), cfg);
```

Figure 11: Code from the Main.java file in the core module, as it looks when updated to the LWJGL3 backend.

6 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 dynamic updates using integration and unit test.

7 Discussion

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

7.1 Requirements Validation

To which extent did your design meet the requirements?

8 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.

Refs