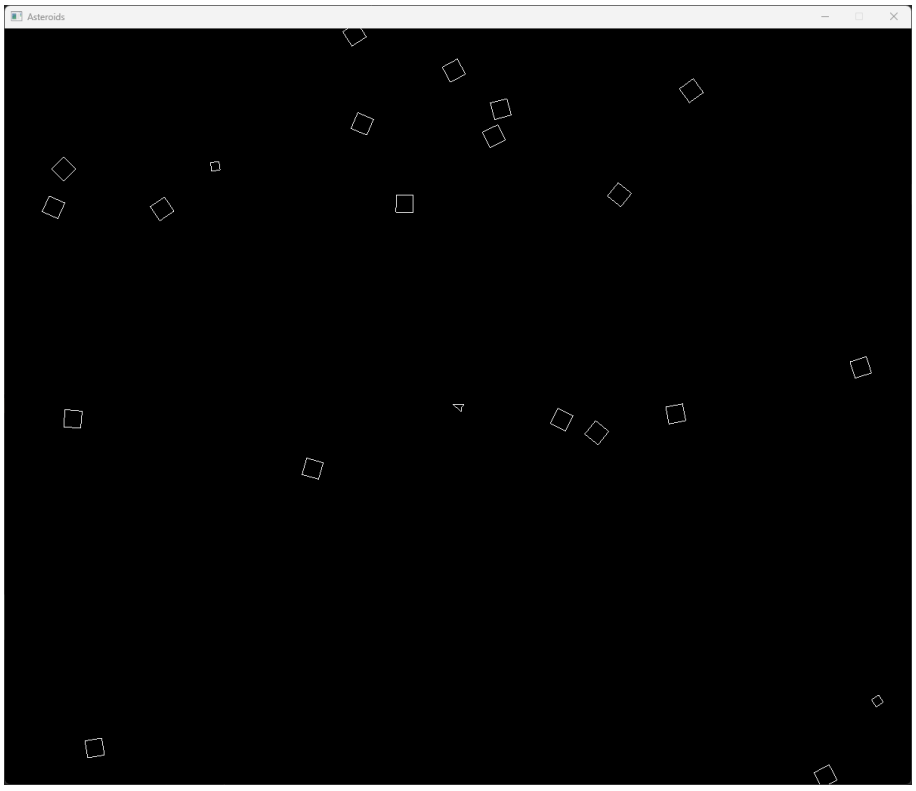
A component-based game

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 Username: Autowinto
Git Repository: https://github.com/Autowinto/SB4-KOM

Contents

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 the assignment specification.

2.1 Functional Requirements

The game must be a component- and data-oriented Asteroids game implemented using the LibGDX Java library. The game must be based on the provided AsteroidsEntityFramework project in the provided repository[1]. The game must include the components Player, Enemy, Weapon and some sort of Map to play on. The Player, Enemy and Weapon components must all implement the provided interfaces from the aforementioned AsteroidsEntityFramework project. If a module is removed from the game, the game must still function properly, meaning that there must be no hard dependencies between the different modules and must be independent of eachother.

2.2 Non-Functional Requirements

Whenever an asteroid is hit, either by another asteroid or by a player/enemy bullet, it must split into two smaller asteroids. If the asteroid in question is already a small asteroid, it should be removed from the world. Likewise with the enemy and player entites. If either are hit, they should be removed from the world. The player must be able to be moved around using the arrow keys and be able to shoot with the spacebar.

[1] https://github.com/sweat-tek/SB4-KOM

3 Analysis

In order to figure out how the game should function exactly, we need to define what the game and the different components in the game should do.

3.1 LibGDX

As per the formal definition of the assignment, the project makes use of the LibGDX library to handle the technical Game Engine side of things. LibGDX game loops

3.2 Maven

Maven is a dependency management system for Java. Using Maven, it is possible to automate download and install of dependencies on a per-component level as well as on a project as-a-whole level.

3.3 Module Systems

3.3.1 ServiceLoader

The Java ServiceLoader is a Java class useful for loading classes implementing specific interfaces or "services", hence the name. The ServiceLoader can find all implementations of an interface and fetch the classes, allowing for instantiation without explicit import of a class. This is incredibly useful when building component-based software, as it allows us to have a shared set of interfaces, which components can implement. Said components can then be loaded into the core part of the software without any hard dependencies.

3.3.2 Java Module System

The Java Module System or JPMS Java's built-in module system since Java version 9. To use the Java Module System you simply have to add a module-info file to the root of the module with module definitions. TODO: Explain this module definition?

3.3.3 Spring

Java Spring is a dependency injection framework for Java

3.4 Components

To fulfill the requirements of the game, several components need to be developed.
TODO: FOCUS ON WHAT AND NEVER HOW

3.4.1 Weapon

Both the Player and the Enemy component require some sort of weapon in order to be able to shoot. This needs to handle spawning some kind of bullet entity which will then be handled by a system that moves the bullet and handles collision between bullet and entity.

3.4.2 Player

Player

3.4.3 Enemy

Enemy

3.4.4 Map

Since the Asteroids game is relatively simple, there's no reason to have a particularly advanced map. Therefore, the Map component in this case isn't actually its own component. The map is represented by entities having coordinate positions and them being placed in the "map" by their coordinates.

Bullet is my weapon

3.5 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

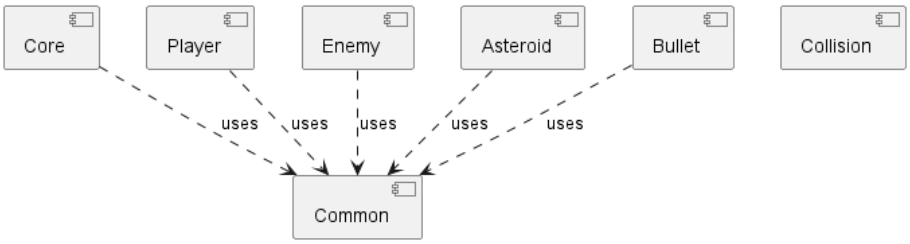


Figure 1: A UML Component diagram over the project

4.1.2 Class

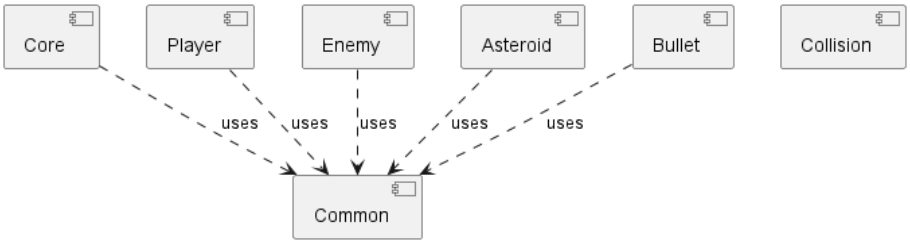


Figure 2: A UML Class diagram over the project

4.1.3 Sequence

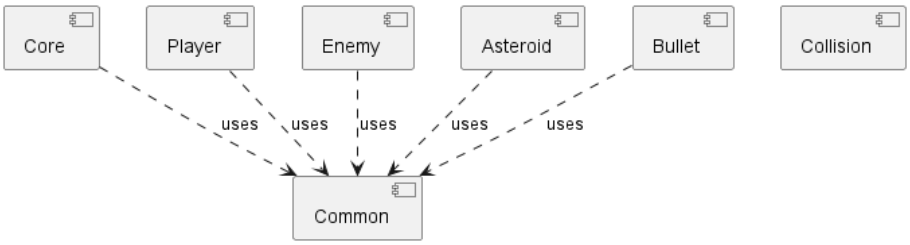


Figure 3: A UML Sequence diagram over the project

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?

MAP IS NOT AN EXPLICIT COMPONENT, BUT SIMPLY REPRESENTED BY COORDINATE POINTS.

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