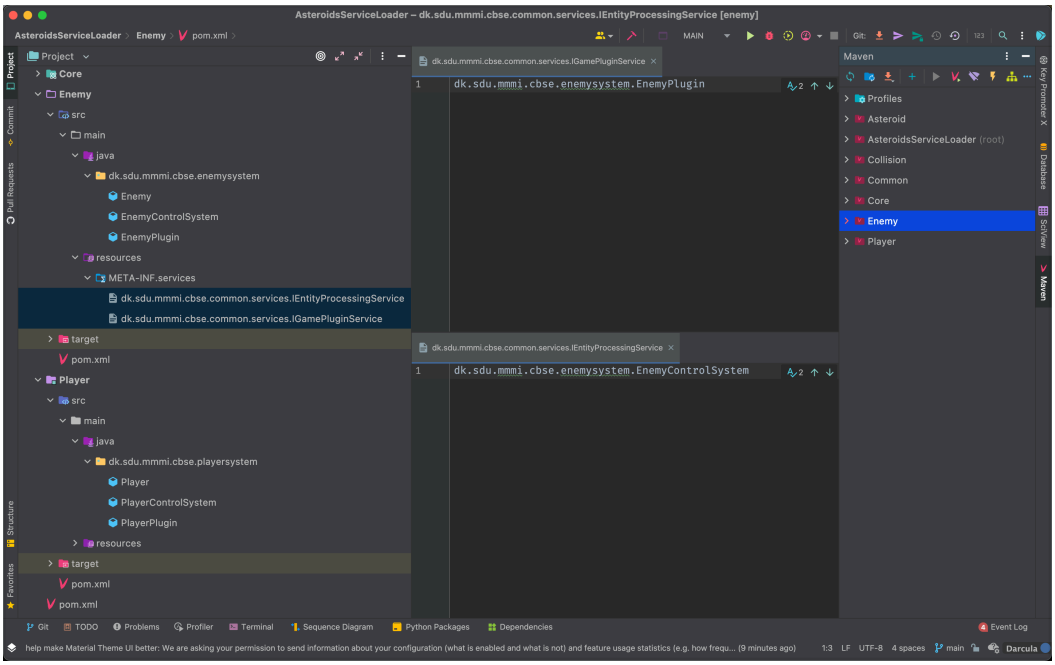
Assignment 1 - Component-based systems

JavaLab with AsteroidsServiceLoader

To test the service loader, an enemy module was created by copying the player directory and giving it the name Enemy, but it can also be done by adding a new module in project structure in IntelliJ. The new Enemy component has the same directory structure as the other components.
The code in the control system (EnemyControlSystem) and plugin (EnemyPlugin) take a lot of inspiration from the Asteroids and Player module. META-INF.services files are also adapted to the enemy component, and is required for the serviceloader to find the components. In IntelliJ, a small check in the maven tabs done to make sure everything shows up and is running. In order to make the enemy show up I also had to add the enemy dependency to the Core -> pom.xml file.

The screenshot below shows the project structure, the META-INF files, and the Enemy component shown in the maven tab.



NetBeansLab1 with AsteroidsNetbeansModules

Running the minimal run-time container

We need to uncheck the following in the project properties -> library to run the minimal run time container as shown in the video:

org-netbeans-bootstrap
org-openide-filesystems

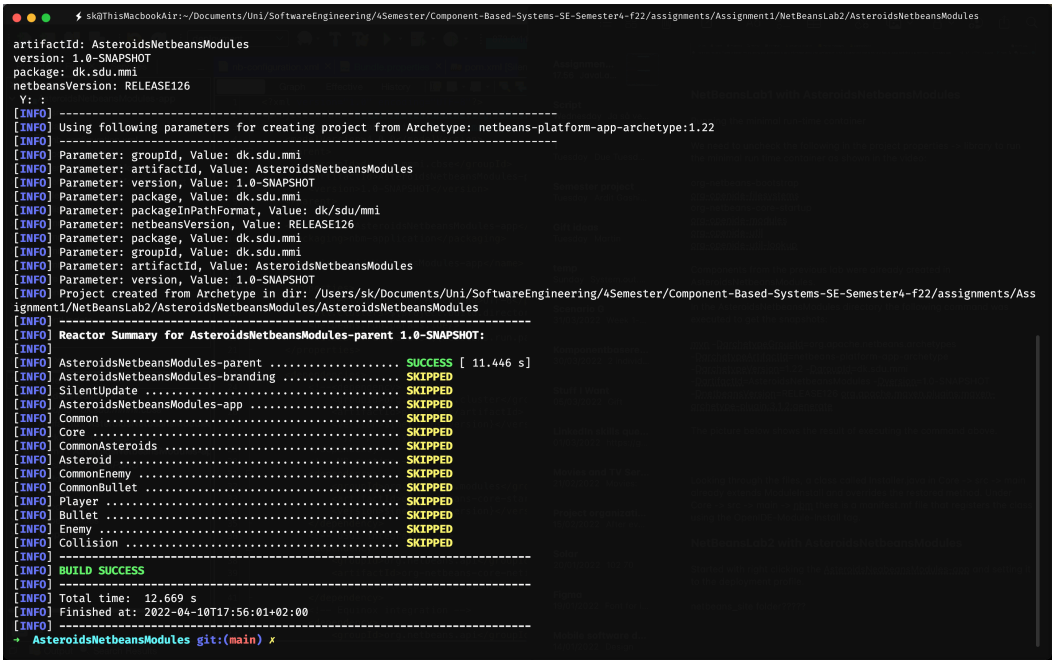
org-netbeans-core-startup
org-openide-modules
org-openide-util
org-openide-util-lookup

Components from the previous lab were already created in AsteroidsNetbeansModules.

In the AsteroidsNetbeansModules directory the following command was executed to get the snapshots:

```
mvn -DarchetypeGroupId=org.apache.netbeans.archetypes -DarchetypeArtifactId=netbeans-platform-app-archetype -DarchetypeVersion=1.22 -DgroupId=dk.sdu.mmi -DartifactId=AsteroidsNetbeansModules -Dversion=1.0-SNAPSHOT -DnetbeansVersion=RELEASE126 org.apache.maven.plugins:maven-archetype-plugin:3.1.2:generate
```

The picture below shows the result of executing the command above.

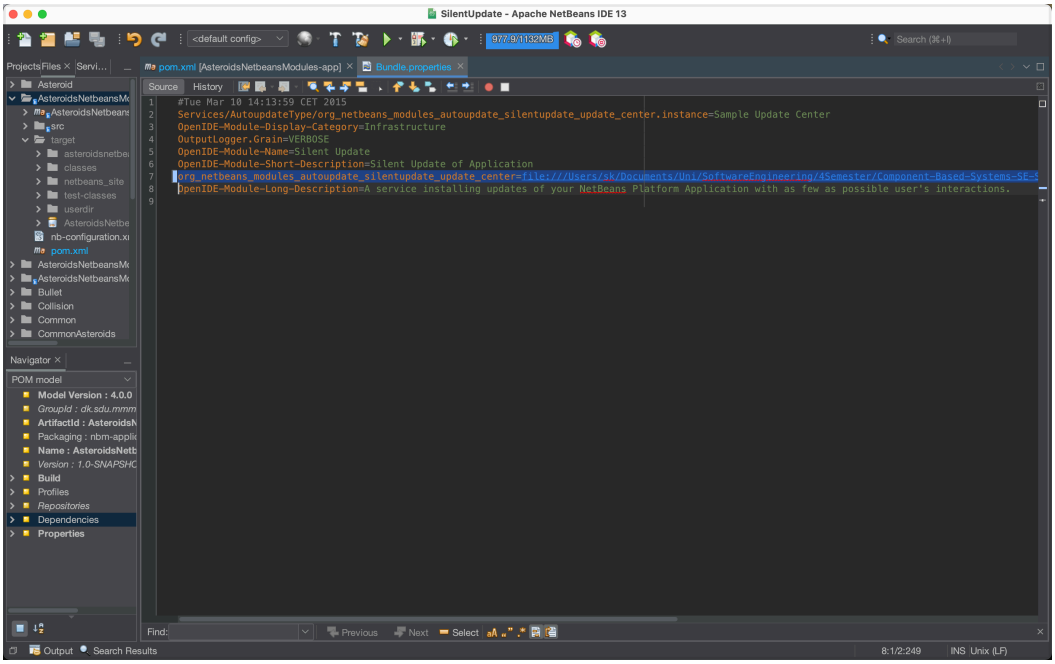


Looking through the files, a class called Installer.java in Core -> src -> main already extends ModuleInstall and overrides the restored method. Under Core -> src -> main -> nbm there is a manifest.mf file that registers the class using the OpenIDE-Module-Install tag.

NetBeansLab2 with AsteroidsNetbeansModules

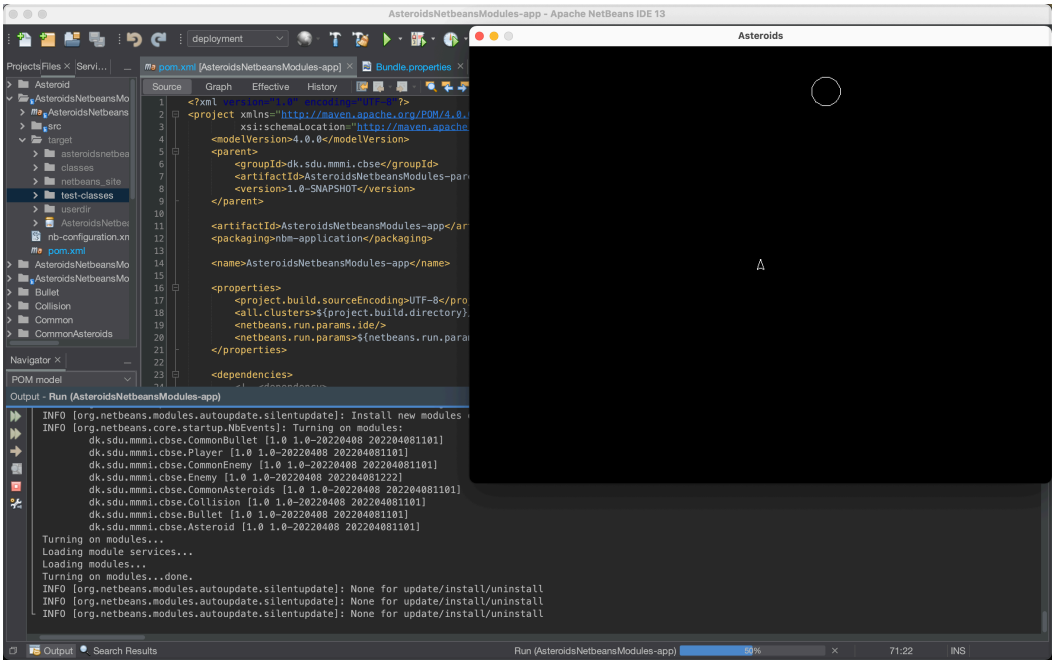
Started with right clicking the AsteroidsNeabeansModules-app and setting it to the deployment profile. After setting the configuration to deployment and executing a clean and build with the dependencies in the application pom.xml, a

netbeans_site folder is created in the application -> target. In the file SilentUpdate module -> Bundle.properties, the path to the netbeans_site -> updates.xml is changed. This is shown in the screenshot below.



There was an issue with the core loading where it failed to build when trying to generate the netbeans_site folder through the development configuration. Right clicking on the core and going to properties -> actions and adding a property to skip the tests for both the clean and build and the run actions fixed that issue.

The game can be seen with the SilentUpdate turning on the modules in the screenshot below.



DesignLab with IntroLab and JavaLab

We don’t include Libgdx in our dependency analysis.

The Monolithic Asteroids Game with IntroLab (AsteroidsLibGDX)

The Main class is not included in this dependency analysis.

ID	Class	Dependencies	Dependency Depth
1	Player	Game SpaceObject	2
2	SpaceObject	Game	1
3	GameState	GameStateManager	1
4	PlayState	Player GameKeys GameStateManager	3
5	Game	GameInputProcessor GameKeys GameStateManager	3
6	GameInputProcessor	GameKeys	1
7	GameKeys	-	0
8	GameStateManager	GameState PlayState	2

The Asteroids Game with JavaLab (AsteroidsServiceLoader)

ID	Class	Dependencies	Dependency Depth
1	Asteroid	Common	1
2	Collision	Common	1
3	Core	Common	1
4	Enemy	Common	1
5	Player	Common	1
6	Common	-	0

Influence of Component-oriented Design on large systems

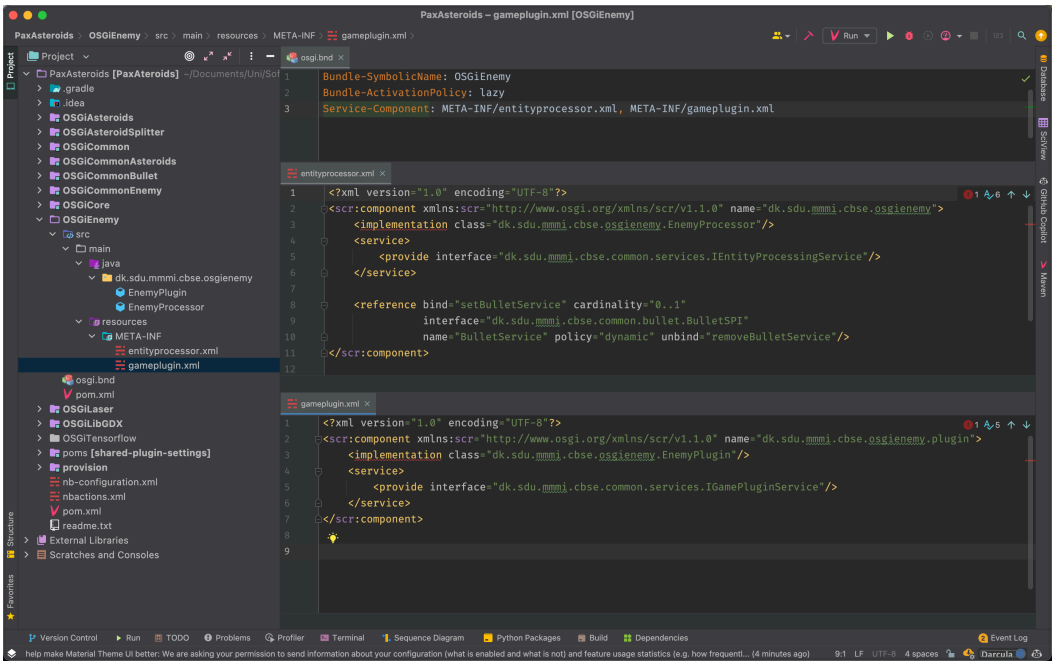
The amount of dependencies in the dependency analysis of the AsteroidsServiceLoader is much lower than that of the first table showing that it can be beneficial for larger systems to use a module component system for development. Having a modular system makes the system more scalable while reducing coupling since the amount of dependencies are significantly reduced. This reduction happens through the way modules use and communicate with their Service Provider Interfaces.

Component-based systems - Assignment 2

You should also create a more through explanation on how you have created your test.

OSGiLab with PaxAsteroids

Modules are stated and declared in the projects pom.xml. OSGi components are then declared using the manifest file sometimes being helped with the OSGi bnd file for each module. In the case of the enemy bundle, the osgi.bnd points to service components in the meta-inf folder that specifies the service components entityprocessor.xml and gameplugin.xml. Bnd helps create the required manifest from the classes and service components. We can see that the enemy has a dynamic dependency injection with the setBulletService from the declared services in the entityprocessor.xml.



The game can then be run by creating a maven configuration in IntelliJ and writing install pax:provision in the command line parameter.
In order for PaxAsteroids to run I had to out-comment a line in the pom.xml, namely:

```
<param>&#45;&#45;vmOptions=-agentlib:jdwp=transport=dt_socket,server=y,suspend=y,address=5005</param>
```

To unload and load the modules while the game is running we use the following commands: lb, start ID, stop ID.

g! lb lists all the bundles and we can start and stop bundles with their respective IDs as shown below. Stopping bundle number 11 removes the enemy from the game screen, and starting it again starts the

```
g! lb
ID|State      |Level|Name
    0|Active     |    0|System Bundle (5.4.0)|5.4.0
    1|Active     |    1|Apache Felix Gogo Command (0.16.0)|0.16.0
    2|Active     |    1|Apache Felix Gogo Runtime (0.16.2)|0.16.2
    3|Active     |    1|Apache Felix Gogo Shell (0.12.0)|0.12.0
    4|Active     |    5|Apache Felix Declarative Services (1.8.0)|1.8.0
    5|Active     |    5|OSGiLaser (1.0.0.SNAPSHOT)|1.0.0.SNAPSHOT
    6|Active     |    5|OSGiLibGDX (1.0.0.SNAPSHOT)|1.0.0.SNAPSHOT
    7|Active     |    5|osgi.core (4.3.0.201102171602)|4.3.0.201102171602
    8|Active     |    5|OSGiCore (1.0.0.SNAPSHOT)|1.0.0.SNAPSHOT
    9|Active     |    5|OSGiAsteroidSplitter (1.0.0.SNAPSHOT)|1.0.0.SNAPSHOT
   10|Active     |    5|OSGiAsteroids (1.0.0.SNAPSHOT)|1.0.0.SNAPSHOT
   11|Active     |    5|OSGiEnemy (1.0.0.SNAPSHOT)|1.0.0.SNAPSHOT
   12|Resolved   |    5|com.diffplug.osgi.extension.sun.misc (0.0.0)|0.0.0
   13|Resolved   |    5|com.diffplug.osgi.extension.sun.reflect (0.0.0)|0.0.0
g! stop 11
g! start 11
```

A module can also be started with an Activator class specified in the manifest file, for example, the Bundle-Activator in the OSGiLaser module.

SpringLab with AsteroidsServiceLoader

Started with a new JavaLab project with AsteroidsServiceLoader. The following beans files that we register are:

AsteroidBeans.xml
CollisionBeans.xml
CoreBeans.xml
PlayerBeans.xml

In the XML files we have the following IDs that are tied to their class path.

asteroidControlSystemBean
asteroidPluginBean
colliderBean
playerControlSystemBean
playerPluginBean

They are then used in Game.java with application context to call upon the modules.

```
public class Game implements ApplicationListener {
  ...
ApplicationContext context = new ClassPathXmlApplicationContext("CoreBeans.xml");
  ...
  }


  public void create( ) {
  ...
    IGamePluginService asteroidPlugin = (AsteroidPlugin) context.getBean("asteroidPluginBean");
    IEntityProcessingService asteroidService = (AsteroidControlSystem)
context.getBean("asteroidControlSystemBean");
    pluginProcessors.add(asteroidPlugin);
    entityProcessors.add(asteroidService);

    IPostEntityProcessingService collidorPlugin = (Collider) context.getBean("colliderBean");
    postEntityProcessors.add(collidorPlugin);

    IGamePluginService playerPlugin = (PlayerPlugin) context.getBean("playerPluginBean");
    IEntityProcessingService playerService = (PlayerControlSystem)
context.getBean("playerControlSystemBean");
    pluginProcessors.add(playerPlugin);
    entityProcessors.add(playerService);

    ...
```

TestLab with AsteroidsServiceLoader

A simple JUnit test on the collision module was made in the AsteroidsServiceLoader. Right clicking the module and saying new JUnit test on an existing class gives us a nice template to start from. A new entity needed both a position part and radius and then the test ran without issue.