By Stephanie Hvenegaard.

Asteroid Portfolio part 1

Introduction

The strategy behind this portfolio is to first build the Asteroid game with all its features, these being Split able asteroids, health components and shootable players and enemies as well as an end state even though it was not in the assignment. The Base version is built from the "Asteroids Entity component" version found in the git repo, from week 8's game lap exercise, I found this to be the most suited base as it was already well prepared for modular design as each "component" was its own separate project.

I felt that I would get the most from the portfolio assignment, by first implementing all of the features from the laps to the base game and then try the different module frameworks, that way I have full separation of concern as I do not need to implement new game features on top of trying to understand the frame work.

I ended up creating one subcommen component called common entity, the reasoning behind this component was that I need the entity player, asteroid and enemy in the collision component to check if they collided. And since a component can only know what is in common, I found this to be the best solution.

Controlle skeme is:
Up - moves the spaceship forward

Left - turns the space ship left

Right - turns the space ship right

Space - shoot a stream of lazers

Enter - resets the game.

01 - Service loader.

The service loader was the fastest and simplest of the bunch to get working. This was done with adding a new class named the service loader. This class implements functionality for looking up the different services modules. For this to work we need to add some meta data to each module.
this is done by adding the follow file structure to each module that depends on common.

\src\main\resources\META-INF\services

Inside here you add files named after the fully qualified name of the interface / services that the module is implementing, In my case it was these 3

- dk.sdu.mmmi.cbse.common.services.IGamePluginService
- dk.sdu.mmmi.cbse.common.services.IEntityProcessingService
- dk.sdu.mmmi.cbse.common.services.IPostEntityProcessingService

the hardest part is to get windows to understand that the name of the interface is not a file type. This is easiest done with visual studio code, as it does not care with file extensions

By Stephanie Hvenegaard.

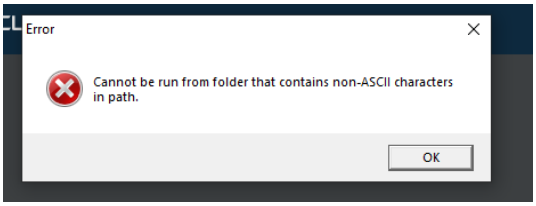
inside these files you put the fully qualified names of the classes that implements these interfaces.

And now you are ready to build and start the game. The service loader will look up all the modules that uses the 3 services and add them into the software, via whiteboard component model, however this make it so you cannot unload a module at runtime, as the service loader has already done its work and you cannot add more modules without restarting the program.

sadly, my project decided to stop working one day before deadline so a working portfolio will have to wait until an other date.

02 - Netbeans part 1

Well there is not much to be said here, got the NetBeans client running bare bones, changed the config file so that the app will run without the rich graphical user interface that is natively build in.
clicked build and then this happened



I checked the internet and there is no problems there. I'm only using ascii. So that was a fun quirck.

Sadly do to time constraints I did not have time to fix all the errors however the project is ported to a NetBeans app as the assignment asked for. I ran into dependency issues with the player module and sadly just have too leave it there.

03 - netbeans part 2

Yet another tale of not getting the project to work properly. But I do have a basic understanding of how the thing works, you create a Update center for the app, just like netbeans IDE has it's own for its modules, and in here you register the modules for you app. This would be running on a website but can be for test puruses be running on the file system. Through this system you can activate your modules, or implement a Silent updater (provided by lector) to have it monitor the UC for changes dynamicly.

02 - OSGi - Pax

This I didn't get up and running either my laptop decided to throw a crapton of Java Errors at me, how ever the Gogo shell is quite familiar because we are using it in our project.

05 - Conclusion

I know I could have used the supplied template from Jan to use but I really don't feel like a learn anything by just copy pasting other people's running software so far I have not been able to recreate the

steps needed to make any of these solutions run besides the service loader. That means that giving the task in the real world I cannot complete it, that disappoints me greatly.