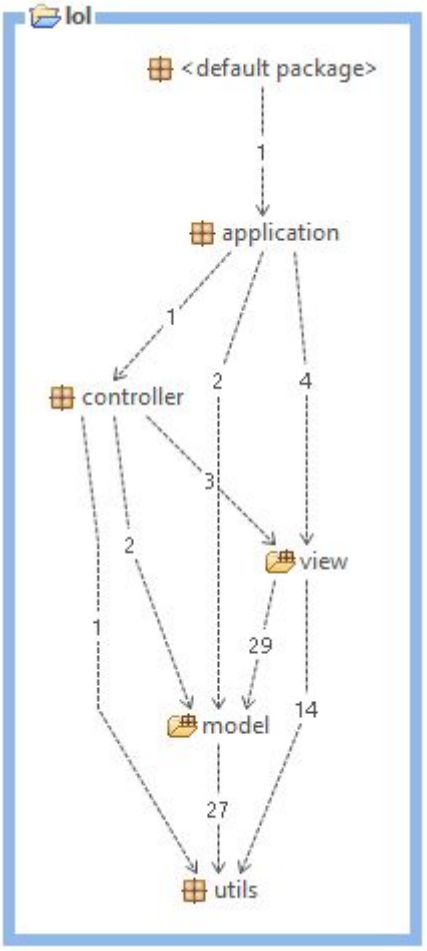
Peer review av grupp 9



Figur 1: Ett beroende-diagram av grupp 9s paket

Gruppen har skapat en objektorienterad kodbas som till hög grad följer SOLID-principerna. Både på hög och låg nivå så är delarna strukturerade på ett modulärt sätt. Kodbasen följer MVC-struktur vilket skapar en tydlig uppdelning på högre nivå. Uppdelningen av klasser i olika packages gör det lättare att förstå och enklare att hitta till specifika delar av programmet. Klasserna följer Single Responsibility Principle och upplevs sällan vara för stora eller för små. Funktionaliteten är uppdelade i submetoder för att undvika för stora metoder som gör mer än en sak. Uppdelningen möjliggör återanvändning av klasser och metoder vilket bidrar till Code Reuse. Tydliga Klass- och metodnamn som i kombination med dokumentation gör också att kodbasen blir mer lättförståelig. Abstrakta klasser har använts för att hålla gemensam kod. Både på hög och låg nivå så är delarna välstrukturerade på ett modulärt sätt, dvs att de är uppdelade i packages klasser och metoder på ett bra sätt. Projektet följer Open closed principle, det är lätt att utöka utan att behöva ändras; till exempel ärver de olika enemies klasserna en abstrakt klass, det är lätt att lägga till flera olika slags enemies.

För att minimalisera starka beroenden har interfaces i många fall använts som mellanlager, vilket bland annat skyddar klasser från oväntad mutering. Projektet använder flera olika designmönster däribland factory-pattern som tillsammans med interfacen bidrar till god abstraktion som skyddar objektens data. Modellen är oberoende av de resterande delarna och Observer Pattern har använts för att skapa en lösare koppling mellan modellen och vyn. Factory Pattern har även använts för att skapa lösare beroenden till klasser som ska instansieras. Lösa beroenden följer Open Closed Principle då koden blir svårare att modifiera av misstag. Koden blir också mindre stel (rigid) vilket gör programmet lättare att utöka. Privata instansvariabler har använts för att skydda innehåll och iterator pattern har kommit till god användning av samma syfte.

Facade pattern har kommit till god användning för att kommunicera med modellen. Modellen innehåller några klasser som inte är färdiga än, det är inget dåligt tecken utan fungerar som ett skelett att fylla i och pekar på god planering, särskilt med dokumentation. Vissa interfaces som inte används så mycket kan för tillfället verka lite små, men om kommer till större användning i framtiden så kan det vara användbart. Icke-muterbara interfaces har kommit till god användning för att skydda muterbara klasser, exempelvis Health och Mutable Health.

Inom modellen finns det mycket dokumentation och testklasser, men utanför finns det instanser av klasser och metoder som saknar både tester och dokumentation. Bredare tester som testar större eller fler klasser kan vara fördelaktigt.

Sammanfattning:

- Modellen är väldokumenterad vilket gör det lätt för utomstående läsare att följa olika klasser och metoders funktion.
- Många interfaces för att skapa svagare beroenden och högre återanvändbarhet.
- Klasserna är väl uppdelade efter MVC-mönstret i Model, View och Controller paket.
- Använder interfaces för att encapsulate muterbara klasser.
- Använder factory mönster för att skapa objekt som det bildas många av.
- Använder sig av arv för återanvändning av kod
- Använder Observer pattern mellan view och modell.
- Allt är väldigt modulärt uppdelat
- Koden följer Single responsibility principle, varje klass har ett väl definierat ansvarsområde
- De genererade UML-diagrammen är stora och svårförståeliga.
- Fler tester och mer dokumentation hade varit fördelaktigt