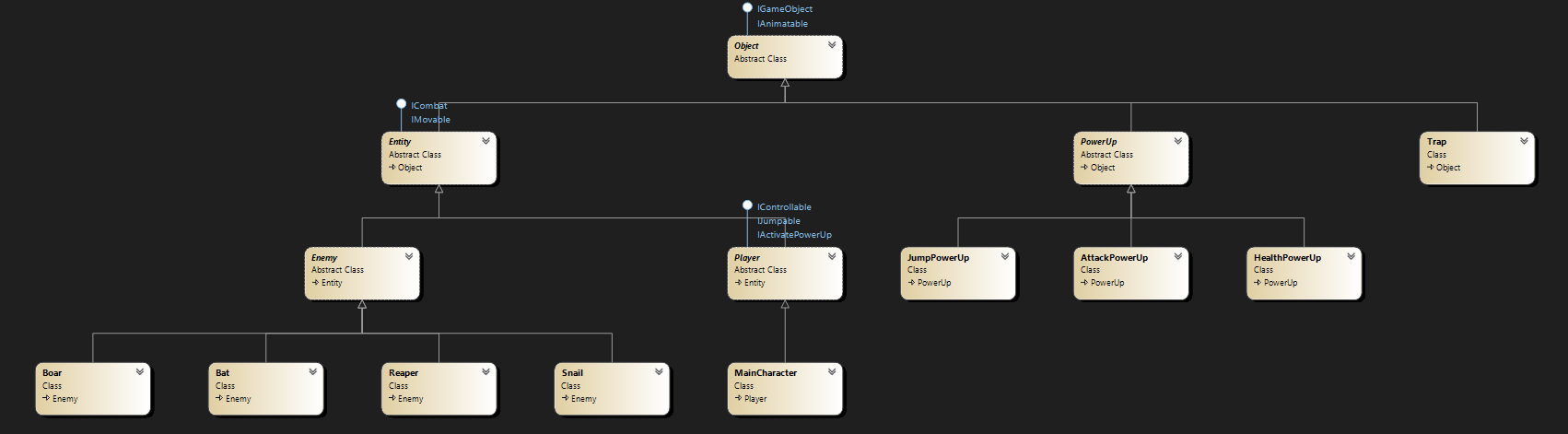
Documentatie Game Development

# **SOLID**

In dit project heb ik overal waar ik kon SOLID toegepast. Zodat het lezen en aanpassen van code gemakkelijker gaat, zonder te veel verwarring te brengen bij de programmeur. Ik probeerde een zo hoog mogelijke cohesion & een zo laage mogelijk coupling te verkrijgen in mijn project.

* S – SRP: Single Responsibility

Klasse Diagram van GameObjects:

Elke klasse hier heeft zijn eigen verantwoordelijkheid.



Bijvoorbeeld de klasse Bat:

Deze klasse heeft een hoge cohesion en een lage coupling.

Zo zijn alle klasse in het project ontworpen zodat we flexibelere code hebben.

* O – OCP: Open/Closed Principle

De code in alle klassen is open voor uitbreiding. Maar gesloten voor verandering. Dit wilt zeggen dat de klassen makkelijk uitgebreid kunnen worden. Bij verandering gaat er dan ook niet veel code moeten aangepast worden.

Dit wordt in het project gedaan door abstract klasse & interfaces zoals Entity, Player, Object, ... Dit kan u ook zien in het klasse diagram.

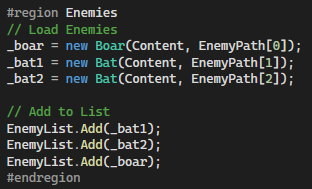
**Naam**: Raf Vanhoegaerden **Nummer**: s128965

* L – LSP: Liskovs Substitution Principle

Object zijn vervangbaar door instanties van hun subtypes. Zoals u kan zien in dit code voorbeeld uit mijn project. We kunnen dit ook bekijken als een Strategy Pattern. Ook het gebruik van interfaces zorgt ervoor dat elke overgervde klasse een ander gedrag kan hebben.

(Bat / Boar erven over van Enemy maar hebben elk opzich een ander gedrag)





* I – ISP: Interface Segregation Principle

Interface splitsen is beter dan 1 gehele grote interface te hebben. Zorg ervoor dat niet gebruikte delen van een interface niet geimplementeerd zijn in de klasse.



* D – DIP: Dependency Inversion Principle

High level modules mogen geen dependency hebben naar low level modules, maar mogen enkel afhangen van hun abstractie. Ik vermijd het gebruik van new keywords in de constructors van de klassen. Ik gebruik het new keyword in hun abstracte klas waar ze van overerven, waar mogelijk.

