Text adventure - Zuul

Dit project gaat over 'Software Engineering'; het schrijven van goed onderhoudbare, gedocumenteerde, aan versiebeheer onderhevige, efficiente, leesbare code. "Loose coupling, high cohesion" (zie glossary).

De code die je dit project schrijft kun je onder andere gebruiken voor je portfolio voor het keuzedeel Verdieping Object Georienteerd Programmeren.

Doel

- Object Oriented Programming (OOP)
- *C#*
    - 'Moderner' dan Java
    - Ook gebruikt in Unity3D, optioneel Godot
    - 'C++ with sidewheels'

Onderwerpen

- UML/OOP (Composition / Aggregation)
- cohesion / coupling / encapsulation
- versiebeheer / git

Software

Google, download en installeer:

- Visual Studio Community Edition (en/of VS Code)
    - dotnet core 3.1 (of hoger)
- git (versiebeheer)
- yEd (diagrammen)
- maak een account aan op github.com

Aangeleverd

- textadventure.pdf (deze file)
- opdrachten.pdf
- basic source code Zuul in c#

Opdracht

- Integreer/verwerk op creatieve wijze bijgeleverde opdrachten (en inhoud van de workshops) in jouw textadventure.
- Houd op github (compileerbare!) versies bij van jouw textadventure.

> Let op!! Breidt de code uit door nieuwe classes te schrijven (Player, Item, Inventory, Key, Medkit, Enemy etcetera). Het is *NIET* de bedoeling om de code 'uit te breiden' door hele verhalen te schrijven verdeeld over tientallen Rooms.

Eindproduct

- klassendiagram
- compileerbare source code op github
- de sourcecode is SCHOON en bevat dus GEEN executables/bytecode/object files (let op je .gitignore file).
- evaluatieverslag ('evidence') met klassendiagram: geef voorbeelden van je code die een hoge mate van cohesion en loose coupling vertoont, waarom dat zo is, waar dat beter kan en hoe dat zou moeten. Beschrijf ook (geef voorbeelden) waar je code goed 'encapsulated' is.

Glossary

Coupling

The term 'coupling' describes the 'interconnectedness' of classes. We strive for loose coupling in a system -- that is, a system where each class is largely independent and communicates with other classes via a small well-defined interface.

Cohesion

The term 'cohesion' describes how well a unit of code maps to a logical task or entity. In a highly cohesive system, each unit of code (method, class, or module) is responsible for a well-defined task or entity. Good class design exhibits high degree of cohesion.

Encapsulation

Encapsulation refers to the bundling of data ('fields') with the methods that operate on that data. Encapsulation is used to hide the values or state of a structured data object inside a class, preventing unauthorized parties' direct access to them.