LINKÖPINGS UNIVERSITET

TDP005 Projekt: Objektorienterat system

Designspecifikation

Författare

Niklas Åsberg, nikas214@student.liu.se

INSTITUTIONEN FÖR DATAVETENSKAP

Höstterminen 2023
Version 0.2

2023-11-24

1 Revisionshistorik

Ver.	Revisionsbeskrivning	Datum
0.1	Utkast	2023-11-21
0.2	Ändrad design efter feedback på kravspecen	2023-11-24

Innehåll

2 Introduction

I det här dokumentet så beskrivs designen av mitt spel. Jag kommer att gå igenom två av mina klasser i detalj och sen visa UML-diagrammet över alla mina klasser.
Avslutningsvis diskuterar jag mina designval.

3 Detaljbeskrivning

Två av mina klasser kan vara extra intressanta att analysera: Playerklassen och Squareklassen.

3.1 Playerklassen

Playerklassen ska representera spelaren. Klassen implementerar Characterklassen som i sin tur ärver från GameObjectklassen.
Playerklassen interagerar direkt med en annan klass, Squareklassen.

3.1.1 Attribut

Dessa attribut är specificerade i Characterklassen som är en interface som i sin tur ärver av GameObject-klassen.

- int: health - antalet liv spelaren har.
- float: speed - hur snabbt spelaren kan röra sig.
- int: damage - hur mycket skada spelaren gör när den attackerar.
- sf::Time: timeSinceDamaged - tiden sen spelaren sist blev skadad.
- sf::vector2f: location - positionen som spelaren har just nu.
- string: sprite - filvägen till spelarens bild.
- bool: collidable - bool som visar om det går att kollidera med spelaren
- int: size - storleken på spelaren.

3.1.2 Metoder

Metoden attack() är den enda metoden som är unik för Playerklassen, de andra metoderna är specificerade i interfacet.

- onCollision() - vad som händer om en spelare kolliderar med någonting annat.
- move() - metod för att flytta spelaren.
- attack() - metod för att attackera.

3.2 Mapklassen

Mapklassen representerar spelplanen. Spelplanen innehåller alla objekt som implementerar GameObjectklassen.
Mapklassen interagerar med GameObjectklassen för att kunna avgöra om en kollision har inträffat.

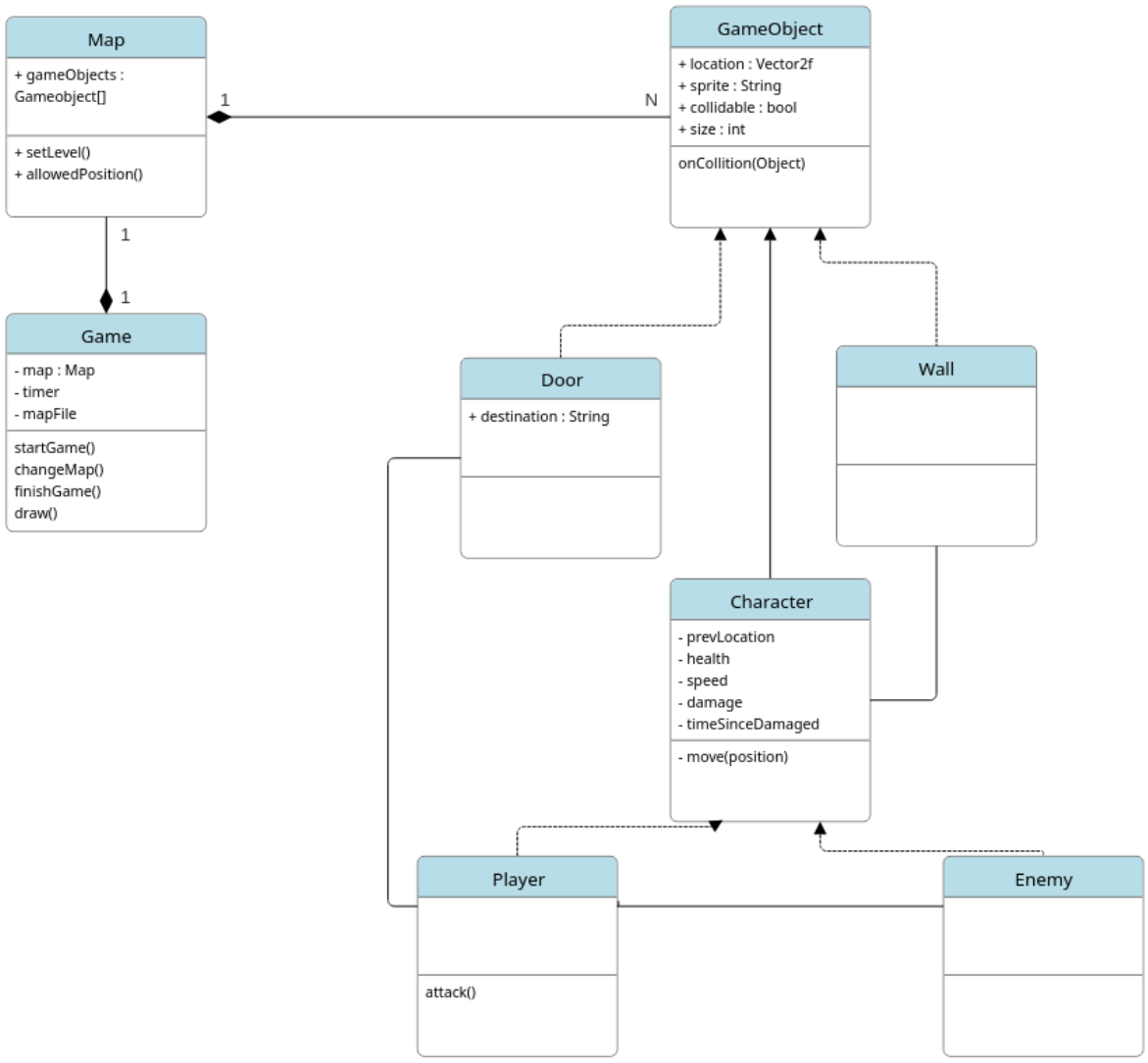
3.2.1 Attribut

- GameObject[]: gameObjects - en lista av referenser till alla instanser av GameObjectklassen.

3.2.2 Metoder

- allowedPosition() - anropas av ett objekt som vill flytta för att se om en kollision kommer ske.

4 UML-diagram



5 Diskution

I mitt spel så utgår allt från en instans av Gameklassen, Gameklassen innehåller en instans av Mapklassen. Gameklassen har även ansvaret att byta instansen
av map när en ny bana påbörjas, Gameklassen sköter inladdningen av ban-filen och skapar Map-instansen från den. Map-instansen innehåller alla GameObject-instanser, och har koll på om objekt har kolliderat. Jag valde den här typen av design då det blir väldigt lätt att lägga till och ändra hur spelet ska se ut.
Om man till exempel vill lägga till andra hinder eller karaktärer så behöver vi inte ändra i de allra flesta klasser. Kollisionshanteringen sker mellan spelobjekt så om vi skulle vilja ändra vad som händer vid kollision så ändrar vi bara i det spelobjektets klass. Tanken är att minska coupling och ha hög cohesion.

6 Externa filformat

Om vi bortser från png-filerna som bara är där för esteriska skäl så är det enda externa filformatet det för banorna. Mitt spel använder sig av en enkel fil där specifikationen för banan är skiven i text, ett tecken för varje ruta med olika tecken för varje objekt.