Platform Asteroids

***Vak**: Kern Module 1*
***Naam**: Nathan Nieuwenhuizen*
3032208 | GDEV jaar 2
***Inleverdatum**: 19-9-2019*

Repository link

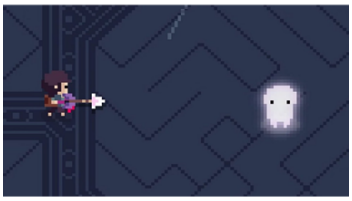
Spel concept

Voor de opdracht om een retro spel te maken met een unieke twist heb ik besloten om een klassieke asteroid spel te maken waarin je eindeloos waves van asteroides moet vernietigen en een zo hoog mogelijke score moet behalen.

Ik heb echter twee twists aan het concept gegeven namelijk:

+

+



1. **Platforming**

In plaats van dat je een schip bent die door de ruimte zweeft, ben je een entiteit dat met de zwaartekracht naar beneden wordt gebracht, je kan naar links en rechts bewegen en je kan springen. Deze concept is geïnspireerd uit klassieke platform spellen zoals Super Mario.

2. **Maar één kogel**

Je kan de asteroides nog steeds kapot schieten, maar je hebt maar één kogel. Als die kogel afgevuurd wordt, kun je het terughalen door weer naar je toe te trekken of er naartoe lopen. De kogel verdwijnt nooit uit de wereld en als de kogel een asteroïde raakt, heb je het automatisch weer in je pistool. Deze mechanic is geïnspireerd uit het spel Titan Souls.

Een ander soort twist is dat de omgeving eindeloos is en dat de camera je mee volgt om de platform idee vast te houden.

Code structuur

Design patterns

Ik heb gebruik gemaakt van de volgende design patterns:

- **Object Pool**
  Om performance te verbeteren over een lange tijd door middel van het hergebruik van objecten.
- **Singleton**
  Een instantie van een klasse waarvan er maar één is. Maakt toegankelijkheid beter
- **Listeners**
  Om classes meer loose coupled te krijgen door gebruik te maken van delegates.
  Polymorphism
  Zodat ik de classes abstracter maken en daarmee sneller kan aanpassen door in het parent class te werken.

Uitwerking

PoolManager, PoolInstance, PoolObjectInstance
Deze drie classes vormen samen de object pooling uit. De poolmanager houdt alle poolmanagers bij elkaar en definieert ook wat de classes van de poolmanager zijn. Hier heb ik gebruik gemaakt van een generics en de singleton. De pool instance is de pool die bijhoudt welke objecten actief zijn en zet ze in de lijst. En de pool Object Instance is het object zelf met extra informatie zoals zijn transform binnen de scene, of het een pool component heeft en of het actief is.

InputHandeler
Deze class regelt alle inputs die de speler neemt (met uitzondering to de button kliks) en stuurt deze door naar de karakter script of naar de game (pauzeren). Hierdoor wil ik het loose coupling versterken en de karakter class mee herbruikbaar maken.

GameManager en Wavemanager
Deze classes zijn singletons die de game beheren. De game manager is specifiek en afhankelijk aan de wave manager en de wave manager niet. De wave manager maakt ook

gebruik van delegates om te checken of alle asteroides binnen de wave vernietigd zijn om daarmee de volgende wave te starten.

Uitdagingen

Een van de uitdagingen die ik tegenkwam zijn het volgende:

- **De objectpool generiek maken.**
  Ik vond het concept van generics een beetje moeilijk te begrijpen en helemaal waarom ik het zou gebruiken. Maar heb het geprobeerd om het bij de objectpool toe te passen. Ik heb eerder met een objectpool gewerkt die ik heb gemaakt vanuit een tutorial. Maar ik wou het mijn eigen maken. Dus ik heb de objectpool class verdeeld in drie subclasses (met behulp van Aaron). Een is de poolmanager f, de tweede is de pool instance (die ik generic heb gemaakt) en de derde is de objectpool instance wat het herbruikbare object is dat ook generic is. Hierdoor kon ik de functionaliteit beter zien en kijken welke ik generic kon maken.

- **Delegates gebruiken**
  Ik heb nauwelijks met delegates gebruikt maar nu ik bij de les duidelijk uitleg heb gekregen zie ik de handigheid van delegates meer.

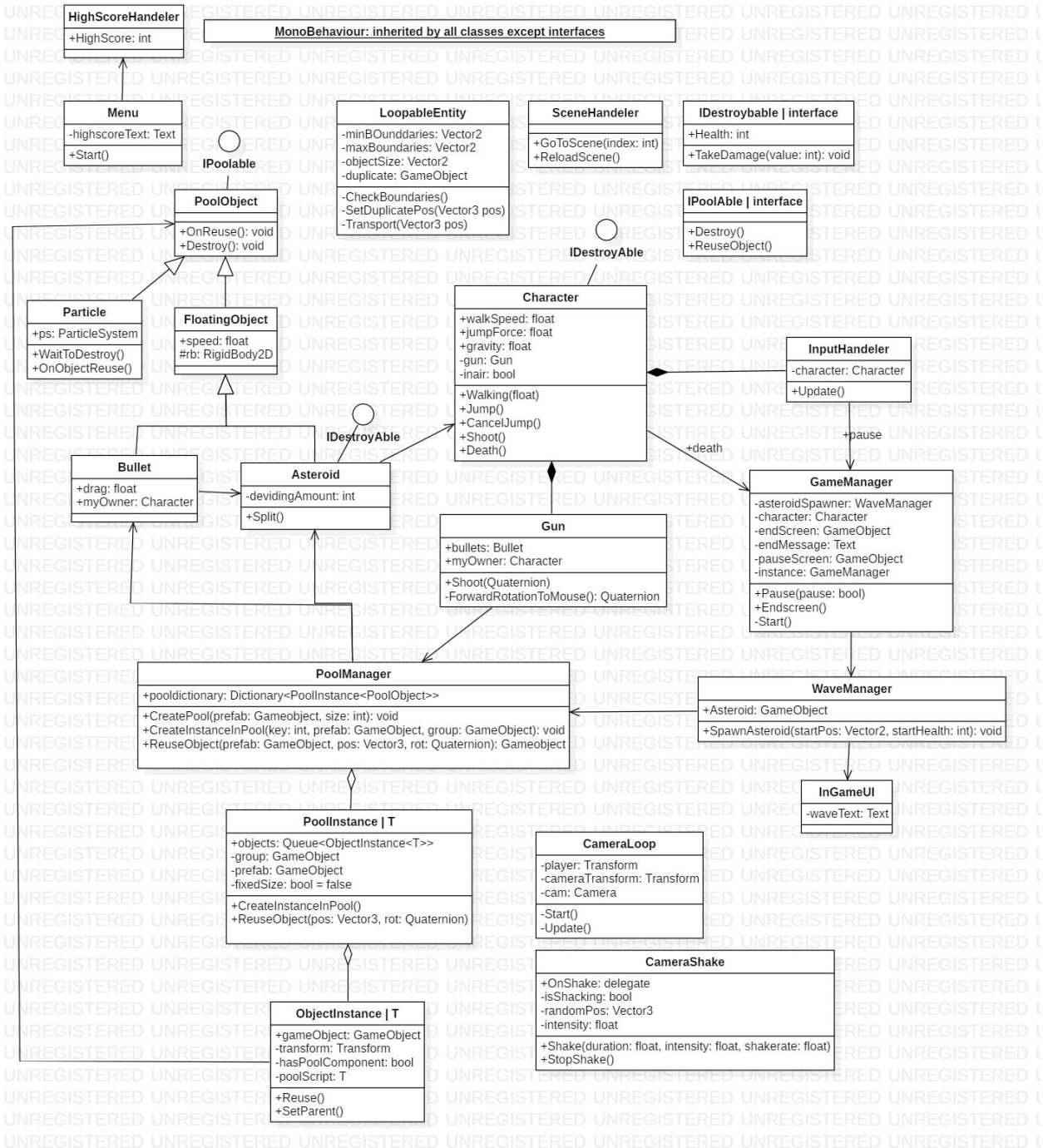
- **Omgeving oneindig groot maken**
  Deze is los van de code, maar ik kwam een probleem tegen om de camera de speler te laten volgen in een oneindig wereld. Ik heb uiteindelijk gebruik gemaakt van camera textures op planes en de main camera daar op laten focussen. Achteraf had ik gebruik willen maken van een shader die de camera scherm vermenigvuldigt en een offset geeft.

Wat kan beter?

Ik heb de feedback die ik heb gekregen zeer gewaardeerd tijdens de lessen en hoop dat ook straks bij het eindbeoordeling te zien. Dat zeggende vind ik ook dat ik kritisch naar mezelf moet kijken en hier zijn de punten die ik vind dat beter kan bij deze code structuur:

1. **Object Poolmanager is generic maar niet praktisch.**
   Misschien komt het doordat ik generics nog steeds niet goed begrijp. Maar omdat ik gebruik maak van een dictionary in de objectpool waarvan de datatype hetzelfde moet zijn. Hierdoor moet alle pool instances dezelfde type hebben en kan ik niet goed gebruik maken van de voordelen van generics.
2. **Delegates en singletons zijn onnodig bij elkaar in één class**
   Helemaal als ze static zijn. Ik zie het vooral in de Wave Manager, het is een singleton maar het heeft ook een delegate. Dit wil zeggen dat als ik een functie wil aanroepen met een delegate, kan ik het ook doen met een instance call. Ik heb dit helaas te laat gezien.

Class Diagram



Activity Diagram (game flow)