Asteroid

Project naam: Asteroid
Student naam: Max Willekes
Studentnummer: 3029575
Vak: Kernmodule Game Development 1
Inleverdatum: 15:00 19-9-2019

Spel concept

De retro game

Ik heb de core mechanics van Asteroids (1979 voor Game Boy) genomen en hierop een twist toegepast. De core mechanics blijven hetzelfde, je krijgt punten voor het vernietigen van objecten die over het scherm zweven en als ze het scherm aan de ene kant uit gaan komen ze er andere kant weer in. Er is één ding anders en dat is dat je niet kan schieten.

De Twist

De twist is dat je speelt als een andere asteroid.
Je bestuurt een asteroid de andere asteroïden moet raken om punten te verdienen.
Ik kwam op dit idee om iets met deze game te doen aangezien ik vroeger veel heb gespeeld. De twist kwam van mijn natuurlijke nieuwsgierigheid over wat er kan gebeuren als je dingen omkeert, wat zich in dit geval uite in de speler geen schip geven waarmee je moet schieten en niet tegen de asteroïden mag stoten, maar de speler juist een asteroïde te maken waarmee ze juist tegen dingen aan moeten stoten.

Toelichting

Gebruikte software

Unity 2019.2.3f1
Microsoft Visual studio 2017

Verantwoording

Finite State Machine:

Ik wil een Finite State Machine gaan gebruiken voor de menu's om ervoor te zorgen dat ik niet meerdere scènes hoef in te laden tijdens het navigeren van de menu's. Dit is zorgt voor minder scènes die bijgehouden moeten worden aangezien ik één scene kan gebruiken voor alle menu's.

Observer Pattern:

Ik wil een Observer Pattern gebruiken voor de input van de speler tijdens het spelen. Dit maakt de input van de speler meer modulair en makkelijker om aan te passen mocht het nodig zijn.

Classes, Interfaces

InputManager:

In de input manager worden events aangemaakt voor andere classes om op te subscriben.

Player:

De speler stuurt de Player class aan met de WASD toetsen van het toetsenbord.
Deze class regelt alleen de beweging van de speler.

GameManager:

Subscribed op de events van InputManager en roept PlayerMovement in Player class aan via invokes.

UiHandler:

Regelt de score counter voor tijdens het spelen. Het eerste wat het doet is op basis van een tag een text object vinden en die de benodigde waardes geven. Heeft ook een publieke functie ScoreUp die kan worden aangeroepen om de score met een random waarde tussen de 1 en 5 omhoog te laten gaan.

Enemy:

De enemy class is verantwoordelijk voor al het gedrag van de Enemy waaronder het checken of hij tegen een speler is aangebotst in welk geval het de UiHandler zoekt op basis van tag, de functie ScoreUp aanroept en zichzelf vernietigt.

Actor:

De parent class van Player en Enemy, dit zorgt ervoor dat zowel de players als de enemy's niet het scherm uit kunnen gaan en in plaats daarvan er aan de andere kant van het scherm weer inkomen.

Spawner:

Zorgt ervoor dat er altijd 5 enemy's gespawnd zijn.

MenuData:

Een ScriptableObject die de data heeft van de prefabs voor de menu's.

StateMachine:

Zorgt ervoor dat er de states worden aangeroepen samen met de benodigde MenuData.
Dit wordt gefaciliteerd door een SwitchState functie die aangeroepen kan worden vanuit de states. Deze heeft een type State nodig zodat het niet elke keer een nieuwe state hoeft aan te maken.

MenuState:

De state voor het hoofdmenu. Als deze state als eerst wordt aangeroepen initialiseert het eerst een prefab van het hoofdmenu op basis van wat het doorgegeven krijgt van StateMachine. Als de state verandert zet het dit geïnitialiseerde object uit.
Als er weer wordt terug geswitched naar deze state zet het dit object weer aan. Dit zorgt ervoor dat er niet de hele tijd nieuwe Game Objecten geïnitialiseerd hoeven te worden.

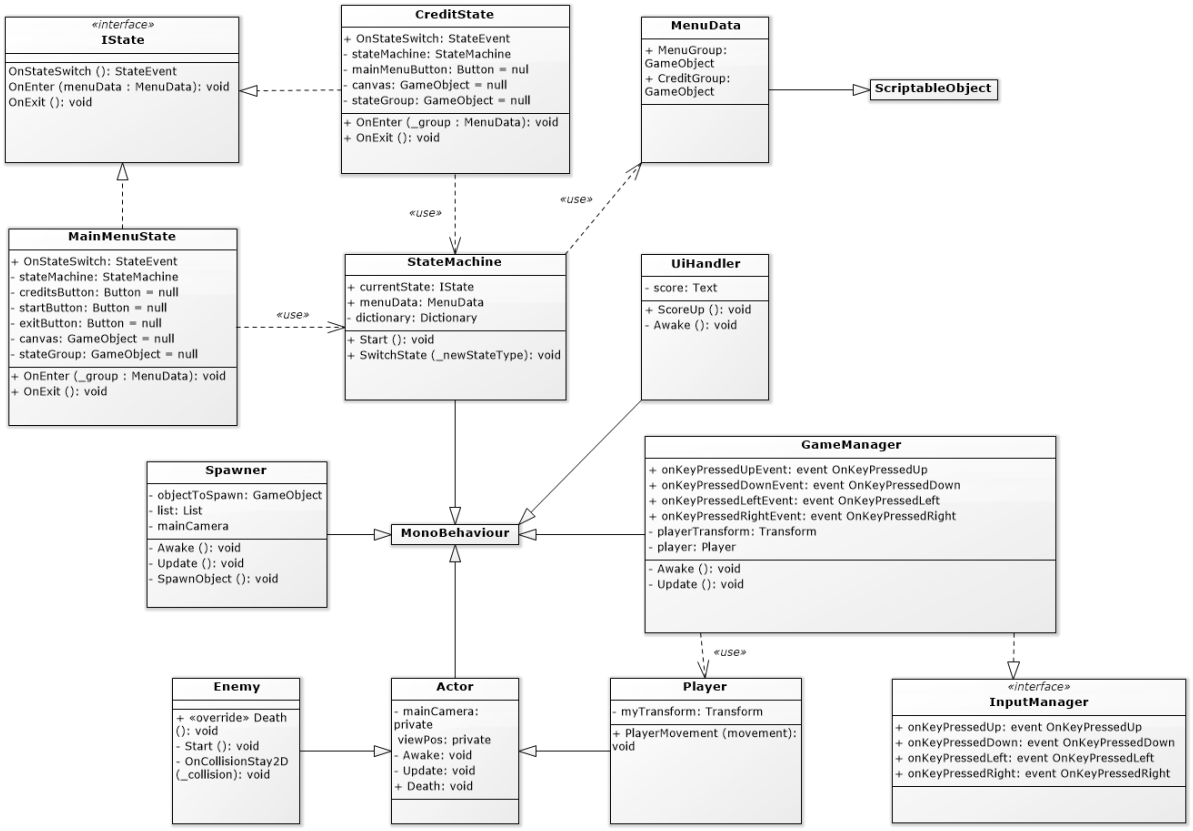
CreditState:

De state voor het credits. Als deze state als eerst wordt aangeroepen initialiseert het eerst een prefab van het credit menu op basis van wat het doorgegeven krijgt van StateMachine. Als de state verandert zet het dit geïnitialiseerde object uit.
Als er weer wordt terug geswitched naar deze state zet het dit object weer aan. Dit zorgt ervoor dat er niet de hele tijd nieuwe Game Objecten geïnitialiseerd hoeven te worden.

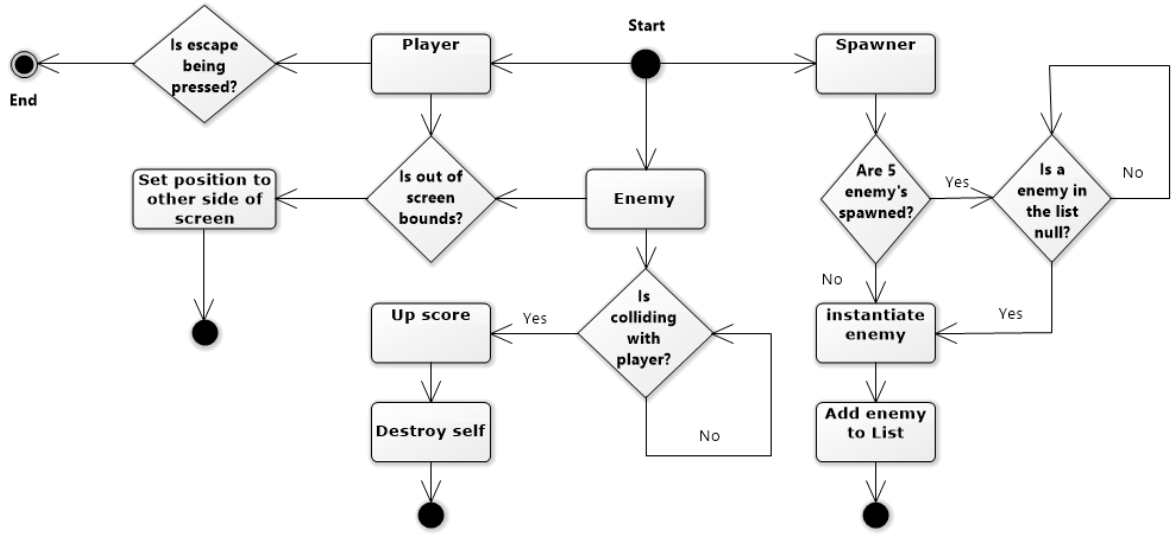
IState:

Een interface die als basis voor de States dient.

UML Class Diagram



UML Activity Diagram



Reflectie

Ik vond dit project erg lastig, van bijna alles wat is behandeld in de les had ik nog nooit mee gewerkt of zelfs nog nooit van gehoord. Hierom ben ik nog redelijk tevreden met wat ik uiteindelijk heb kunnen maken aangezien ik tijdens dit project moeite had met dingen te begrijpen en werkend te krijgen. Ik had daarentegen zeker een paar dingen anders willen en of kunnen doen die er niet van zijn gekomen door een tekort aan tijd.

Ik had graag de scène transitions beter willen doen. Mogelijk een aparte class ervoor in plaats van het te doen in de MainMenuState en Player classes.
Wat ik ook nog graag anders had willen doen is dat UiHandler en Enemy classes alsnog op een tag moet zoeken willen ze een bepaalde functie uitvoeren. Ik heb het gevoel dat dit beter kon maar zoals eerder genoemd, ik had hier jammer genoeg niet de tijd voor.

Ook had ik graag nog een death voor de player willen maken. Waarbij er soms ships zouden spawnen die zouden schiet op de speler en als de speler dan geraakt zou worden zou het spel eindigen. Ook dit is er niet ingekomen door een tekort aan tijd.

Verder ben ik niet helemaal zeker van mijn Activity Diagram aangezien ik deze nog nooit eerder heb gemaakt en maar één referentie had en geen andere kon vinden in de context die ik nodig had.

Ik ben redelijk tevreden met dat ik na wat moeite de FSM werkend heb kunnen krijgen en nu ook redelijk lijk te begrijpen. Hetzelfde geld voor dat ik heb kunnen werken met een ScriptableObject waar ik voor dit project nog niet van wist dat het bestond.
Hiernaast was dit project ook de eerste keer dat ik een UML maakte en dat ik met events, delegates en invokes werkte.