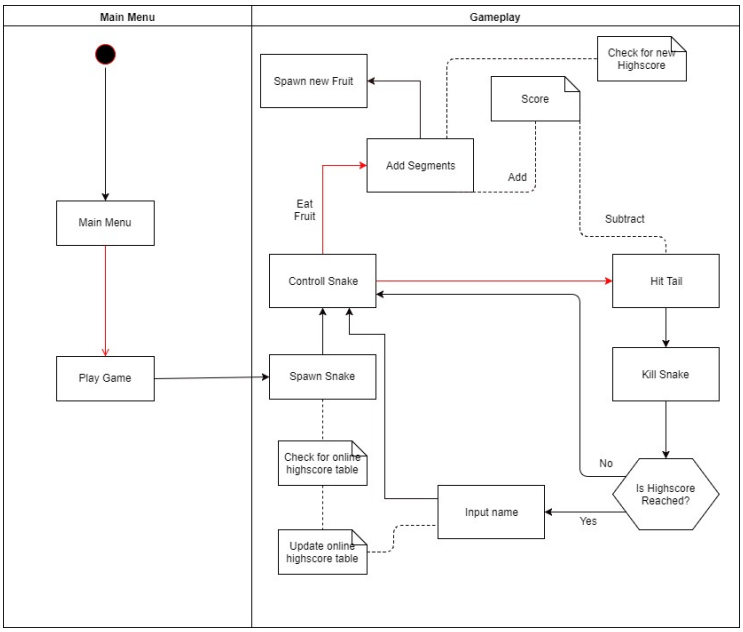
Super snake

Gameplay

Super Snake is een twist op de klassieke video game “Snake”. In Super Snake moet de speler proberen een zo hoog mogelijke score te krijgen door de slang langer te maken, dit is mogelijk door bolletjes te verzamelen die op de bolvormige map spawnen. Wanneer de speler een bolletje verzameld word de slang hier 5 segmenten langer van en beweegt hij vanaf dat punt sneller.



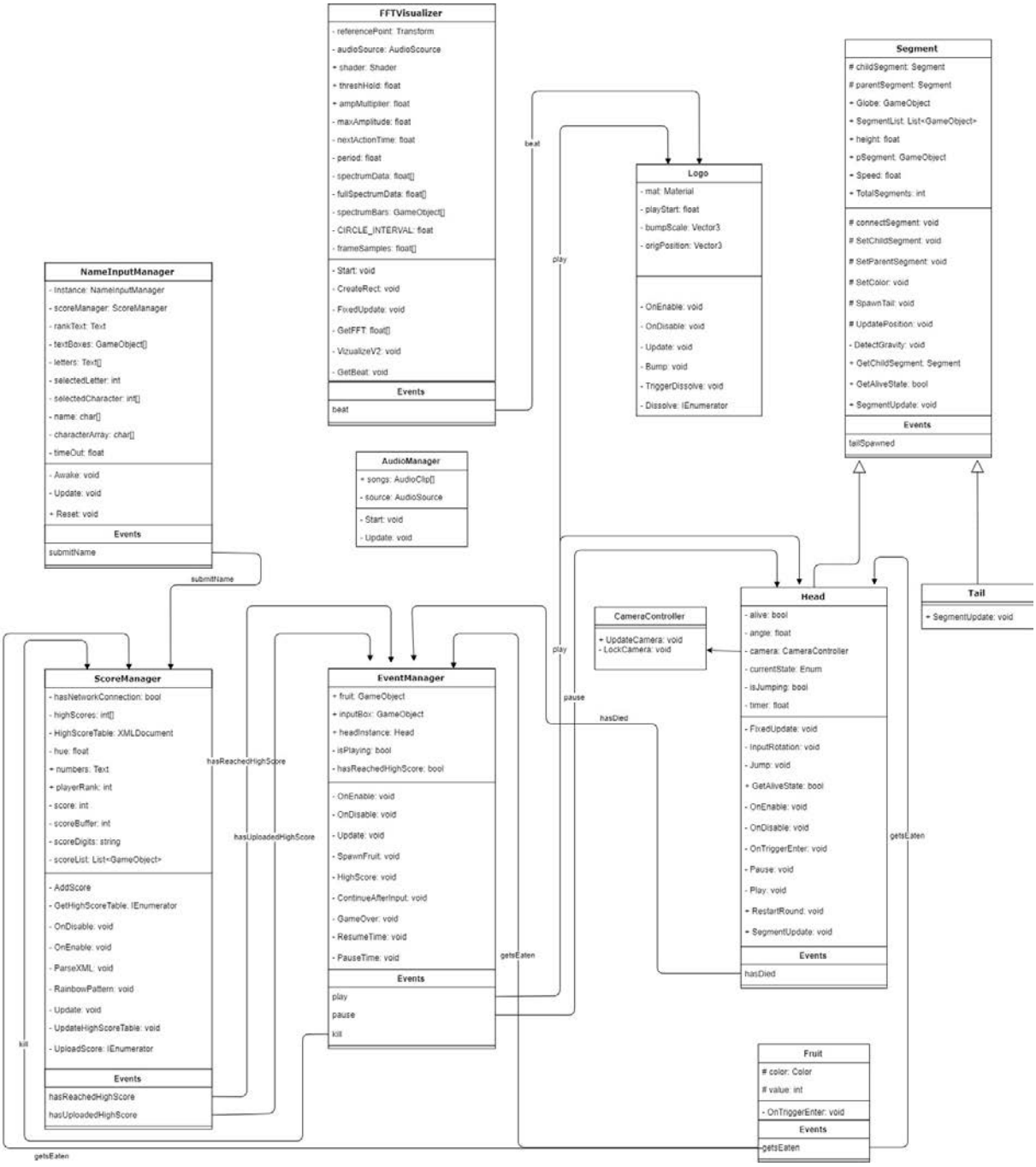
De game is speelbaar met een toetsenbord alleen werkt het best met een gamepad (xbox controller/ joystick). De speler kan met A/D, </>, links/ rechts heen en weer bewegen en kan met de spatiebalk/a-knop springen. Op deze manier kan de speler zijn eigen staart proberen te ontwijken om langer te kunnen blijven spelen.

Wanneer de speler met het hoofd van de slang zijn eigen staart raakt is hij af. Als de speler 1 van de 6 hoogste scores tot dat moment toe heeft behaald(en een netwerk verbinding heeft) krijgt hij de kans om zijn naam in te voeren en deze naar de server te laten sturen met score.

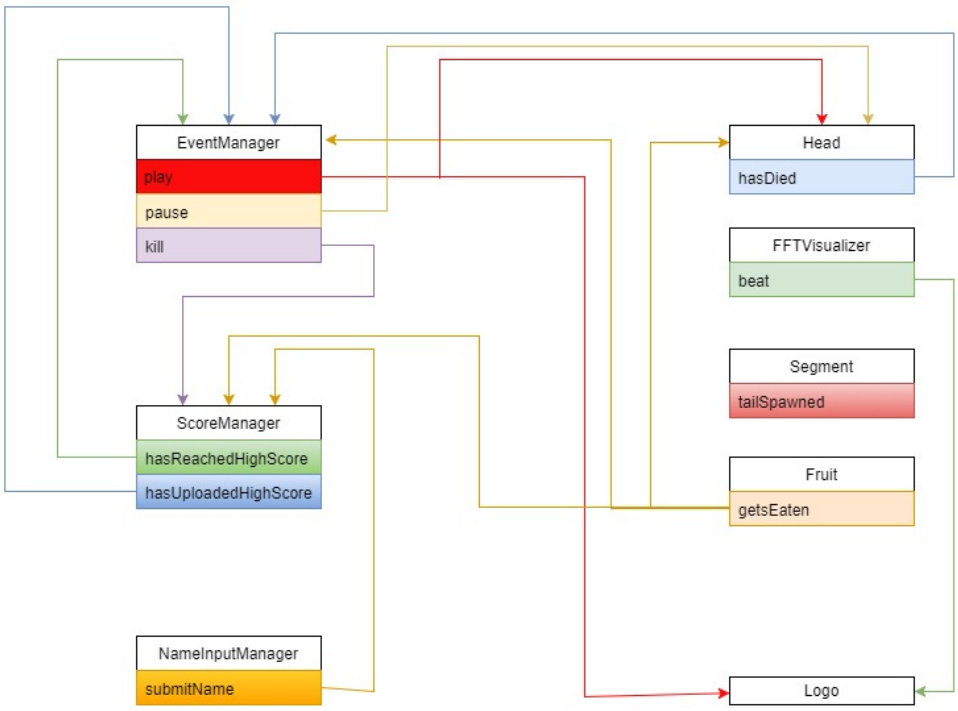
Op de server worden de 6 beste spelers bij gehouden met hun naam en score, deze highscore table is te vinden door de QR-Code die links onderin het scherm te zien is met de telefoon te scannen en zo naar de website te gaan.

UML

Ik heb veel gebruik gemaakt van events binnen mijn game, dit is goed te zien in de uml diagram.



Om ze wat simpeler en leesbaarder te maken beeld ik ze hier onder nogmaals uit, maar dan simpeler.



Op deze manier is het een stuk makkelijker om te zien hoe alle scripts met elkaar communiceren. Buiten deze events om staan de meeste classes namelijk los van elkaar in de scene.

Deze events heb ik gebruikt om onderlinge communicatie tussen de verschillende scripts te vereenvoudigen en efficiënter te maken. Ze hebben me ook enorm geholpen in het uitbreiden van de game aangezien ik aan het begin dan dit project nog niet wist hoe ik het er op het einde exact uit wou laten zien, zo heb ik de keuze om een online highscoretable toe te voegen pas in de laatste 1.5 week gemaakt. Door events te gebruiken bleef het alleen makkelijk om snel nieuwe features toe te voegen zonder veel references te hoeven maken.

Design patterns

Ik heb tijdens het maken van deze game gebruik gemaakt dan de volgend design patterns:

- Observer pattern
- chain of responsibility (soort van)
- Singleton

Observer pattern

Om de game overzichtelijk te houden en communicatie tussen verschillende managers goed te laten verlopen maak ik veel gebruik van events. Voor iedere gebeurtenis waar 1 of meer andere scripts van op de hoogte moeten zijn gebruik ik events, zo kan ik de state van de game bijhouden zonder al te veel references tussen verschillende managers. Deze events hebben me tijdens development enorm geholpen aangezien ik geen concreet idee had van hoe de game er op het einde uit zou zien en graag wou blijven uitbreiden, de events hebben het mogelijk gemaakt om snel en makkelijk op ingame gebeurtenissen in te kunnen springen zonder enorme hoeveelheden references toe te hoeven voegen voor iedere nieuwe toegevoegde feature.

Chain of responsibility

Hoewel ik niet echt het chain of responsibility pattern heb gebruikt heb ik wel een belangrijke design keuze gemaakt naar aanleiding van dit pattern. Om de slang correct te laten bewegen heb ik er voor gekozen om de kop van de slang bestuurbaar te maken en de rest van de slang de kop te laten volgen. Ik ontdekte alleen dat wanneer de framerate dropte de slang onvoorspelbaar werd, ook was het lastig om een geordende nette lijst van alle segmenten bij te houden en waar ze waren. Naar aanleiding hiervan heb ik een eigen take op het chain of responsibility pattern bedacht. Zowel de kop van de slang als de andere segmenten inheriten van de Segment class. De kop heeft hier zijn eigen childclass Head voor en de staart segmenent hebben de subclass Tail. De Head is de enige plek in de slang waar een FixedUpdate vandaan word gecalled, vanaf hier update hij zichzelf. Nadat de kop zichzelf heeft geupdate called hij de segmentUpdate method van zijn child, die doet vervolgens hetzelfde met diens childSegment en zo voort. Dit resulteert in een ketting van segmenten die van kop naar staart 1 voor 1 ieders locatie update. Hierdoor is het nagenoeg 100% voorspelbaar waar ieder segment op ieder moment kan zijn, de scripts worden namelijk in een constante volgorde uitgevoerd van voor naar achteren. Deze update ketting word ook gebruikt tijdens het spawnen van nieuwe segmenten waar het nieuwste segment achteraan komt te staan in de ketting.

Singleton

Ik heb voor de NameInputManager(het script dat de onscreen-UI managed wanneer je je highscore invoert) een singleton gebruikt. Ik moet namelijk op een bepaald moment een functie uit deze class callen en een variabele opvragen en vond deze operatie niet binnen mijn andere events passen qua gebruik. Hierom besloot ik een reference te gebruiken, alleen aangezien ik ten alle tijden maar 1 NameInputManager nodig zal hebben heb ik hier een singleton van gemaakt om hem op een makkelijke manier via script te kunnen vinden.

Object pool

Ik heb toen we les kregen over de object pool heel even geprobeerd om dit toe te passen voor het spawnen van de aparte staart segmenten. Na testen besloot ik alleen toch om liever geen object pool te gebruiken, het bleek overbodig te zijn aangezien de segmenten net zo lang moesten blijven als de slang leefde en ze vervolgend met enkele frames interval moesten despawnen. Hierdoor bleek het

uiteindelijk efficiënter te zijn om de segmenten gewoon pas te instantiaten wanneer ze nodig waren in plaats van de uit de pool te halen. Om deze reden heb ik dit pattern uiteindelijk niet gebruikt.

Veranderingen in development

In development is er enorm veel veranderd. Een aantal van deze dingen waren: het toevoegen van de muziek visualizatie, het toevoegen van een online highscore table, de 3d score, de mogelijkheid om te kunnen springen en controller support.

Hoewel controllersupport verreweg het makkelijkste was om te implementeren heeft dit wel de grootste impact gehad, het bleek namelijk dat de game veel leuker en makkelijker is met een controller dan met een toetsenbord. De controller input bracht alleen wel een ander probleem met zich mee, en dat was de textinput voor de highscore table.

De highscore table was het lastigst om te implementeren, gelukkig kende ik al wel een manier om het werkend te krijgen (ik heb namelijk al eens eerder met unity in combinatie met xml en php gewerkt voor project context van vorig jaar). Op mijn hku server staat een php script dat ik heb geschreven om de xml highscoretable aan te kunnen passen en downloaden. Wanneer er een naam en een score via WWW FORMS vanuit unity naar het php adres word verstuurd zorgt het php script ervoor dat de xml file met de 6 beste spelers word geüpdatet en stuurt hij deze xml file mee terug naar unity. Deze highscore table is te zijn door de QR-Code links onderin het scherm te scannen met de telefoon en zo mijn site te bezoeken met daarop een live score table.

De controller naam input was eigenlijk nog best wel een uitdaging (in ieder geval meer dan ik origineel verwacht had). Ik heb namelijk zo efficiënt mogelijk geprobeerd te werken met zo min mogelijk logics. Hierdoor ontstond een (tijdens development in ieder geval) nogal verwarrend web aan array indexes en relaties tussen variabele. Uiteindelijk is het wel gelukt om het semi elegant op te lossen.

De muziek visualisatie was gelukkig niet al te moeilijk om werkend te krijgen, ik heb gelukkig afgelopen zomervakantie zelf een arduino muziek visualisatie gemaakt met FFT dus was ik al wel een beetje bekend met audiovisualisatie.

Al deze features waren origineel niet gepland maar ik vond het leuk om aan dit project te werken dus besloot er maar zo veel mogelijk coole features in te zetten.

Hoe zou ik het anders doen?

Ik denk dat ik het de volgende keer niet veel anders dan dit op zou lossen. Ik ben zo vroeg mogelijk begonnen met development en heb mijn code zo geschreven dat ik op ieder moment veranderingen kon maken wanneer ik er achter kwam dat iets beter/ efficiënter kon.

Al met al ben ik best tevreden met het uiteindelijke resultaat.