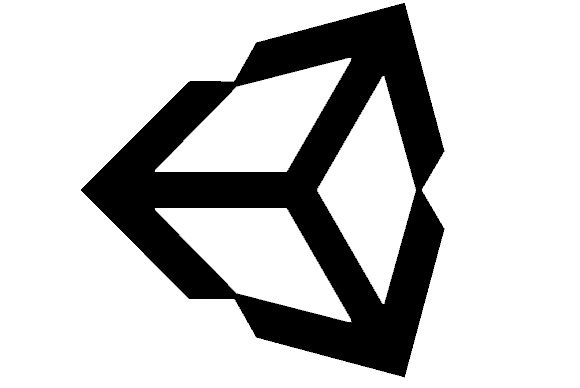
Einddocumentatie GTO7





**Naam:** Frank Severijns  
**StudentNr:** 2383764  
**Klas:** GD7

Datum: 14/01/2016

# Multiplatform

# API

Ik heb gekozen om de GameJolt API te gebruiken (<http://gamejolt.com/games/unity-api/15887>). De GameJolt API stelt me in staat om scores op te slaan en trophies te maken die de speler kan unlocken.

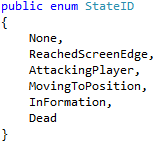
Om de API te gebruiken heb ik een Unity plugin geimporteerd met daarin een prefab die ik in de menu scene van mijn spel heb geplaatst. De plugin is makkelijk te gebruiken en heeft ook een ingebouwd UI voor o.a. het inloggen van de gebruiker en het bekijken van trophies.

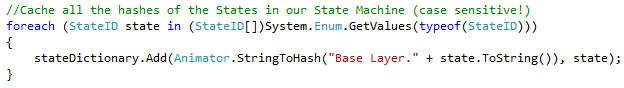
Om de API aan te roepen heb ik een singleton klasse (APIManager) geschreven waarin ik een aantal publieke methodes beschikbaar maak om zo makkelijk vanuit andere scripts de benodigde functionaliteit van de API te gebruiken.   
Om bijvoorbeeld een trophy te unlocken moet er eerst gekeken worden of de trophy niet al unlocked is (anders zal de pop-up van de trophy iedere keer opnieuw tevoorschijn komen), en kan ik daarna pas de trophy daadwerkelijk unlocken. Door dit een enkele keer in mijn APIManager klasse als publieke methode UnlockTrophy(int trophyId) te implementeren hoef ik vervolgens in andere klassen alleen maar APIManager.instance.UnlockTrophy(id) aan te roepen.

De APIManager beheert naast de trophies ook het opslaan en inladen van scores in de DataStore en het inloggen en uitloggen.

# AI

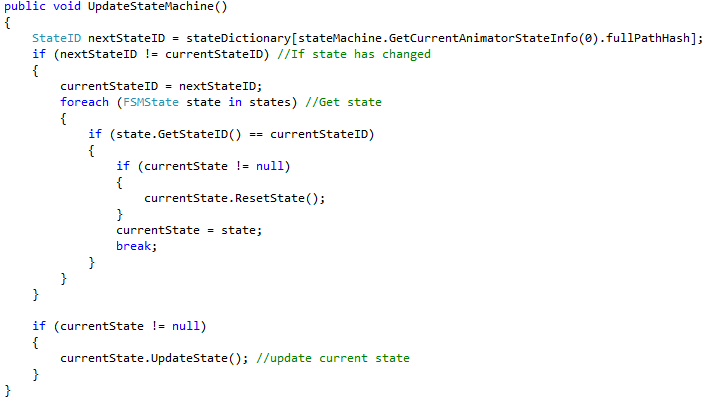
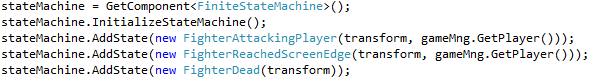
Om de FSM te maken heb ik gebruik gemaakt van het Advanced FSM voorbeeld uit hoofdstuk 2 van GAI7, in combinatie met een tutorial waarin Mecanim gebruikt wordt als Finite State Machine (<http://darkgenesis.zenithmoon.com/mastering-unity-2d-game-developmentai-and-state-machines/>).

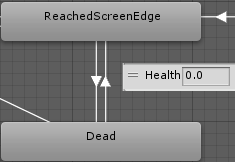
Om te beginnen heb ik een FiniteStateMachine script. In dit script wordt bijgehouden welke state op dit moment actief is. In dit script staat ook een enum waar alle gebruikte states in staan. Het is belangrijk dat deze naamgeving ook precies zo terugkomt in de Mecanim states. Deze StateIDs worden aan een Mecanim state verbonden worden middels een Dictionary.

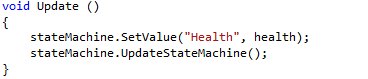
Deze Dictionary heeft een key (int) en een value (StateID). De key van een dictionary item is de hash van de mecanim state naam, de value is de bijbehorende StateID.

Vervolgens heb ik een abstracte klasse FSMState gemaakt. Deze klasse wordt overerft door alle state scripts. Hierin staan een aantal methodes gedefineerd, waaronder de GetStateID() methode, waarin een StateID teruggegeven wordt (in bijvoorbeeld een DeadState script word dan StateID.Dead teruggegeven).

De State scripts worden door iedere agent toegevoegd aan een List in hun FiniteStateMachine component.

In iedere update wordt bijgehouden welke state in Mecanim actief is. Door de actieve state uit te lezen en de hash ervan te gebruiken om de bijbehorende StateID uit de eerder genoemde Dictionary te halen, kan bepaald worden welke State script op dat moment gebruikt moet worden.

Om in mecanim het transitioneren van state naar state bij te houden maak ik gebruik van de mecanim parameters en transition condities. Door bijvoorbeeld de waarde van Health constant up-to-date te houden zorgt mecanim er zelf voor dat wanneer health onder 0 komt, de actieve state transitioneert naar Dead (zolang er in mecanim een transition tussen de states bestaat).

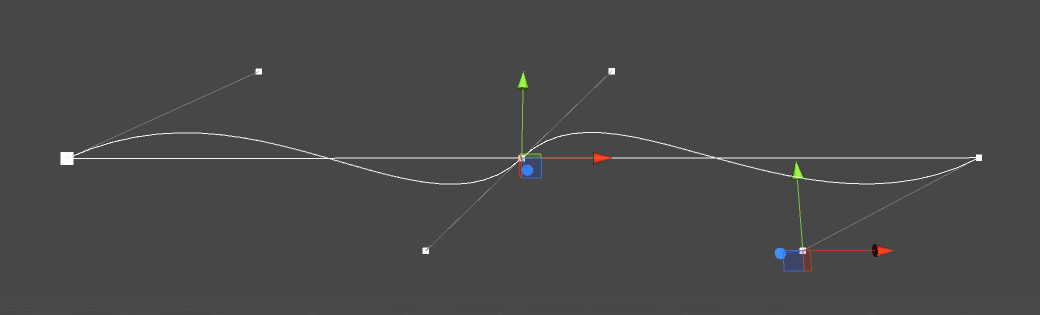


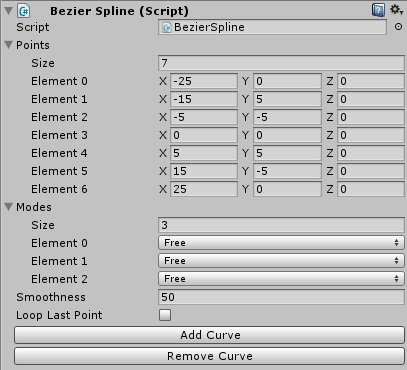
Dit helpt om de states en de transities te visualiseren. Ook zorgt dit ervoor dat ik niet in code hoef te bepalen wanneer ik van state naar state moet transitioneren, dit wordt allemaal automatisch gedaan in mecanim, zolang de parameters en condities goed ingesteld zijn en bijhouden worden.

# Efficient Level Creation

Een van de vijanden in het spel is een “drone”, deze vijand doet niets anders dan in grote groepen in een bepaald patroon over het speelveld bewegen. Om deze patronen te maken heb ik een editor script gebruikt dat ik eerder gebruikt heb voor een ander spel. (<http://catlikecoding.com/unity/tutorials/curves-and-splines/>)

Het editor script zorgt ervoor dat ik vanuit de inspector en in de Unity editor bezier splines kan maken met versleepbare punten. Deze bezier splines kan ik gebruiken als pad die de drone vijanden volgen door telkens naar volgende punt in de spline te bewegen.



//nog iets over wat de editor script precies doet

Om het instantieren van objecten zoals projectielen efficienter te maken heb ik een instance pooling systeem gemaakt. De instance pools kunnen een dynamische grootte hebben, wat betekent dat er nieuwe instanties bij gemaakt worden als er te weinig in de pool zitten, maar kunnen ook een vast aantal behouden.

Om dit systeem te bouwen heb ik twee scripts gemaakt: ObjectPooler en PooledObjectBehaviour.   
De instances hebben allemaal een PooledObjectBehaviour script (of een script die van dit script overerft).   
In het PooledObjectBehaviour script staan methodes om het gameobject van de instance in een parent te zetten en om het object te disablen.   
In de ObjectPooler wordt een lijst van de instances gemaakt en bijgehouden. Als ik vanuit een ander script een beschikbare instance wil ophalen om te gebruiken heeft de ObjectPooler een GetNextInstance() en een GetNextDynamicInstance() methode met een GameObject returnwaarde.   
De GetNextDynamicInstance() methode instantieert een nieuwe instance als er geen beschikbare instances in de pool zijn.   
De GetNextInstance() methode heeft een boolean parameter, als deze boolean true is zal de methode altijd een GameObject uit de pool returnen, ook al is deze in gebruik. Staat de boolean parameter op false, dan returnt de methode null.  
De GameObject in de returnwaarde kan dan in een ander script (bijvoorbeeld de ProjectileController) geactiveert worden zodat het lijkt alsof er een nieuw projectiel gespawnt is.

# Persistence

# Localization

# Audio Management

# Adaptive Difficulty