# BOMBERMAN 3D

Dennis Borst

Studenten nummer: 3030815

Kernmodule Game Development 1

19-09-2019

# Spel concept

Mijn spel concept in het kort is bomberman in 3D (in first person view). Je speelt dit met z'n vieren waarvan jij één speler bent. De anderen drie zijn (computergestuurde) tegenstanders die je moet verslaan. Alle vier de karakters beginnen in één van de vier hoeken van de map. Je doel is om alle tegenstanders uit te schakelen met je bommen. Zodra ze allemaal dood zijn dan win je het spel. Maar opgepast, als je je in de buurt van een tegenstander bevind weten ze automatisch waar je bent en zullen ze proberen bommen te plaatsen die jou kunnen uitschakelen (je kan ook door je eigen bommen uitgeschakeld worden).

Normaal is bomberman in 2D, dus deels van de twist is dat het in 3D is. Maar ik hoor mensen al denken, een spel van 2D naar 3D is toch geen twist? Dat klopt, alleen wat dit spel zo interessant maakt is dat het in first person is. Normaal kan je in 2D nog je tegenstanders op het beeldscherm zien waardoor je precies weet waar ze lopen. In mijn spel moet je op het geluid afgaan. De bom maakt een explosiegeluid zodra ze exploderen dus door middel van het geluid weet je hoe ver de tegenstanders van je af staan.

De grote uitdaging die ik mijzelf geef is de AI (de tegenstanders). Ik heb nooit eerder met een FSM gewerkt dus leek het mij interessant om dat te proberen.

# Verantwoording

**Factory Method Pattern:**
Ik wou in dit project ook gebruik maken van interfaces. Dit zorgt ervoor dat mijn code een stuk overzichtelijker is en dat ik niet in de herhaling val in scripts. Zo moeten bijvoorbeeld een muur en een karakter allebei schade krijgen van een bom. Dit is dus ideaal voor een interface.

**Singleton Pattern:**
Ik wou graag een aantal singletons gebruiken om het zo makkelijker te maken om bij algemene scripts te komen in mijn game. Zo is de GameManager vrij algemeen waardoor het goed kan zijn dat er meerdere scripts dat script moeten kunnen aanroepen. Natuurlijk zorgt dit er ook voor dat je er maar één van mag hebben in je scene, wat bij managers vaak voorkomt. Ik zal ook geen twee GameManager nodig hebben.

**State Pattern:**
De state pattern wil ik graag toepassen in mijn AI. Ik heb hiervoor nog niet gewerkt met de Finite State Machine en ik weet dat het erg handig kan zijn voor wat complexere AI. Dus ik heb gebruik gemaakt van de FSM. Ik wou graag leren hoe ik de FSM ook met verschillende classes kan maken waardoor het overzichtelijker is dan één script vol met code.

# The Classes

**GameManager:**
De GameManager regelt de grote lijnen van het spel. Het zorgt ervoor dat als je gewonnen hebt dat je naar de volgende scene gaat of als je dood bent de scene weer word gereset. De GameManager is een singleton omdat ik het handig vind om dit vanaf elke script aan te kunnen roepen. Zo ben je niet gelimiteerd van waar uit je alle functies uit de GameManager kunt oproepen.

**ActorManager:**
De ActorManager heeft maar één doel en dat is het bijhouden van hoeveel spelers er nog levend zijn. Zodra er maar één speler over is dan heb je gewonnen (dit gebeurd alleen wanneer jij niet dood bent, anders zou de scene instant herstarten). Ik heb de ActorManager een singleton gemaakt omdat je dan makkelijk vanuit het Enemy script kan communiceren dat er een tegenstander dood is. Daarnaast komen er ook geen meerdere ActorManagers.

**UIManager:**
In de UIManger worden de levens van de karakters bijgehouden en geupdate naar het scherm.

**Actor:**
Het Actor script regelt alles wat de Player en de Enemy gemeen hebben. Zo communiceert dit script naar de bom of er een bom geplaatst kan worden, zo ja dan word de bom geplaatst en gaat het script daar verder. Met E kan de bom geplaatst worden. Dit script houd ook bij hoeveel health elk karakter nog heeft.

**Player:**
De Player is een inheritance van de Actor. Dit script doet niet veel extra’s behalve dat het een singleton bevat die er voor zorgt dat het Enemy script precies weet waar de Player zich op dat moment bevind. Daarnaast override de Player de Die functie van Actor. Want als de Player doodgaat dan moet het spel restarten. Dus die roept de GameManager dan aan.

**Bomb (& IDamagable):**
De bom voert zelf een hoop functies uit. Zo zal de bom checken of het mogelijk is om geplaatst te worden. Ik heb hier ook gebruik gemaakt van een delegate en events. Volgens mij was het hier niet heel erg nodig maar ik had hiermee er voor kunnen zorgen dat er meerdere functies werden uitgevoerd door alleen het event aan te roepen. Wanneer de Explode functie word aangeroepen zullen er vier raycast checken of er een karakter of een muur geraakt word. Dit checkt ie door middel van de interface IDamagable. Als de interface toegevoegd zit op een script dan kan het dus geraakt worden door de bom. Die vervolgens de Damage functie zal aanroepen in dat script.

**DestructableWall:**
Dit script zorgt er alleen voor dat de muur een IDamagable interface bevat waardoor ie opgeblazen kan worden door de bom.

**Enemy:**
Het Enemy script heeft zelf niet heel veel functies. Het Enemy script maakt een nieuwe FSM (Finite State Machine) script aan die al het gedrag van de Enemy aanstuurd. Op het moment staan er vooral variable in het script die doorgegeven worden aan de states door middel van de interface IUser. Ik denk zelf dat dit niet de beste oplossing is maar voor nu werkte het en de volgende keer ga ik naar een betere manier opzoek.

**IUser & ITarget:**
De IUser zorgt ervoor dat alle variablen van de Enemy ook bij de states bruikbaar zijn. ITarget doet dit ook maar alleen maar voor de positie van de speler. Zodat de Enemy altijd bij kan houden of ie in de buurt is van de Player.

**FSM:**
In dit script worden alle states in een dictionary gezet. In dit script word ook het switchen van de states gehandhaafd. Hier word ook de IUser en de ITarget doorgegeven aan het State script.

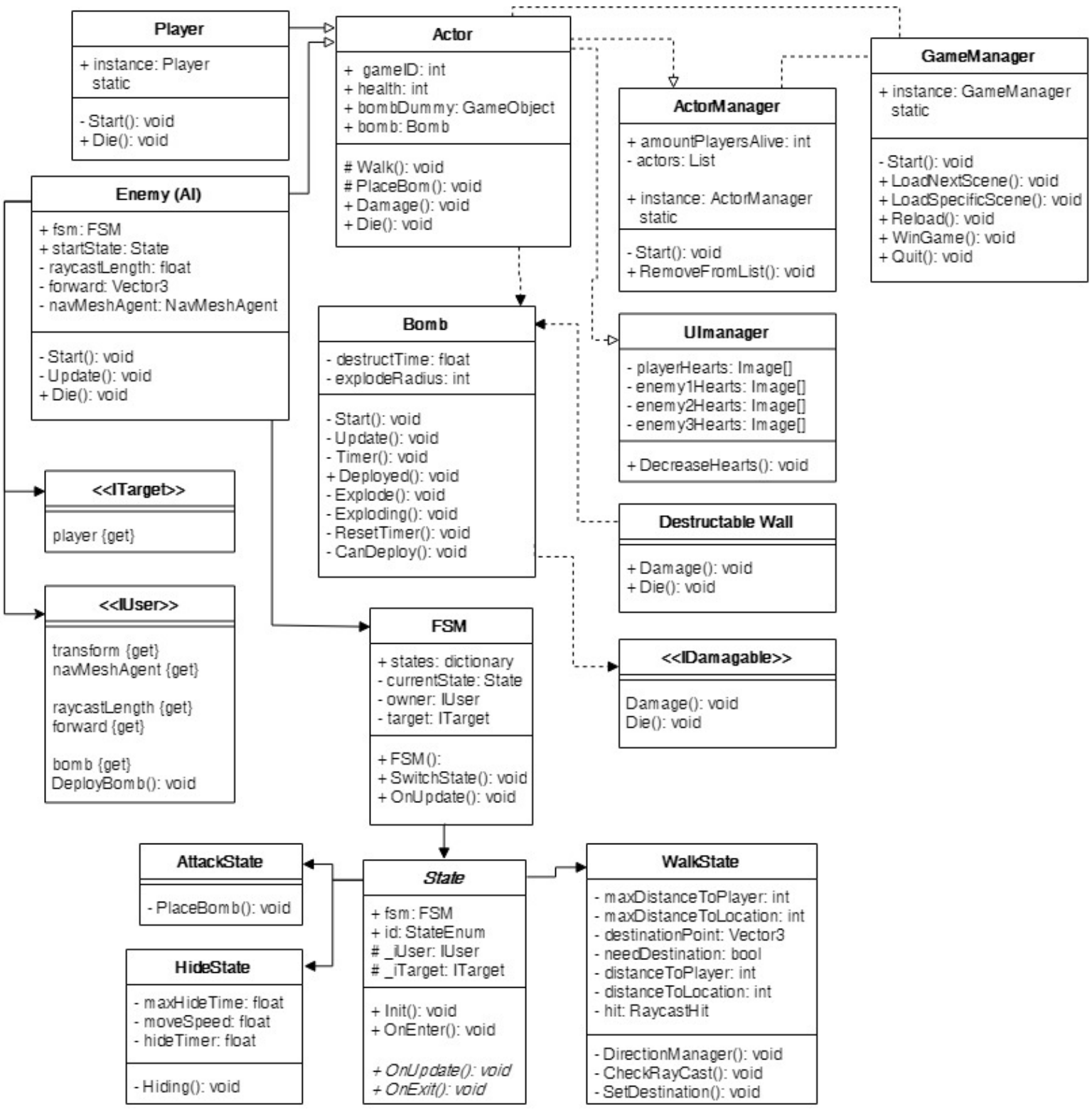
**State:**
De State class heeft als doel dat het een soort blueprint is voor alle andere states. Zo implementeer ik een aantal functies in de State class die door iedere inheritance van States gebruikt moet worden.

**WalkState:**
Dit is de state waar de Enemy een random locatie op de map zoekt. Hier word ook gecheckt of de Enemy in de buurt is van de Player, zo ja dan negeert de Enemy het uitgekozen punt en gaat ie vervolgens een pad richting de Player maken. De Enemy checkt door middel van een raycast naar voren of ie tegen een muur of een speler aanloopt. Zo ja dan zal de state switchen naar de AttackState.

**AttackState:**
In deze state word aangeroepen dat de Enemy een bom moet neerleggen. Als de bom is neergelegd veranderd de state naar de HideState.

**HideState:**
In de HideState word gezegd dat het karakter naar achteren moet rennen. Ik wou eerst nog maken dat de Enemy zich ook nog naar de zijkant kon bewegen als de Enemy tegen een muur achter zich aan kwam. Alleen hier kwam ik niet uit. Dus voor nu rent de Enemy alleen naar achter. Zodra de bom geëxplodeerd is dan veranderd de state weer naar de WalkState.

# UML Class Diagram:



## UML Activity Diagram: