Technisch Ontwerp

Diving for Treasure: The Sequel

**2 April 2021**

**Studenten**

Laurens van Brecht (640101)

Alex Cheng (634967)

**Docent**

Mark van der Maas

# Inhoudsopgave

[**Inhoudsopgave**](#_m5k5kghq3woo) **2**

[**Inleiding**](#_1fob9te) **3**

[**Klassendiagram**](#_3znysh7) **4**

[**Klassen**](#_2et92p0) **5**

[DivingForTreasure](#_c4zb9154srnh) 5

[Player](#_1ksv4uv) 5

[Bomb](#_3dy6vkm) 5

[Coin](#_2s8eyo1) 5

[OxygenTank](#_lnxbz9) 6

[BombSet](#_1t3h5sf) 6

[CoinSet](#_17dp8vu) 6

[OxygenTankSet](#_35nkun2) 6

[Shark](#_itg04karcygp) 7

[Clock](#_4d34og8) 7

[PowerUp](#_44sinio) 7

[ResistanceField](#_2ntfdglzfm65) 7

[Boat](#_x2kkobdusqwx) 8

[EndScreen](#_a8lc9ohp84um) 8

[GameObject](#_4y6b24utkok9) 8

[**Interfaces**](#_1y810tw) **9**

[Object Interaction](#_9zz6h8dnb6sb) 9

[Object Score](#_gysxkyqdgae3) 9

[ObjectSet](#_alxal6wrpsmb) 9

[OneOfAKingObject](#_ycov0ll5r6bi) 9

[**Gebruikte bronnen**](#_2xcytpi) **10**

# Inleiding

Zoals al bij het functioneel ontwerp is gezegd, is dit spel, Diving for Treasure: The Sequel, gebaseerd op het spel Diving for Treasure dat ik heb moeten maken tijden het vak Structured Programming Development (SPD).

Naast het functionele ontwerp met een uitleg en analyse van ons gewenste spel, waarin u overtuigd werd wat voor spel ik wou gaan programmeren, is er ook een technisch ontwerp opgesteld. In dit document zullen is te zien hoe het spel is ontwikkelt door middel van een klassendiagram en daarbij een korte uitleg wat er precies in de klassendiagram weergeven is, met daarbij de verbanden tussen elke klasse / interface.

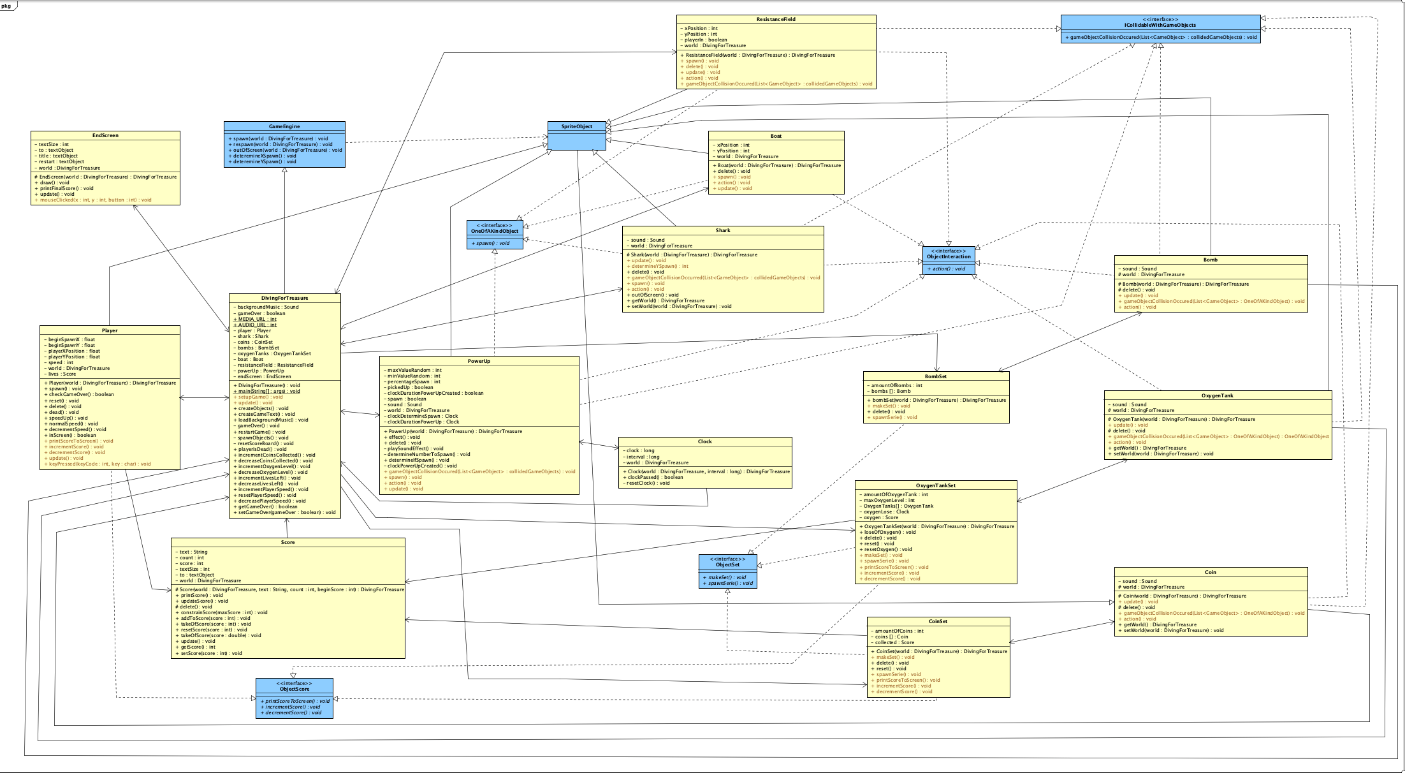
# Klassendiagram

Het gebruik van een klassendiagram in de UML (Unified Modeling Language) is zeer belangrijk voordat je wilt beginnen met het programmeren en het daadwerkelijk realiseren van jouw spel. Door middel van een klassendiagram laat je de structuur zien van je klassen en interfaces, en daarin de attributen, operatoren en de relaties tussen de objecten.

Omdat je een klassendiagram hebt gemaakt en hierin alle klassen met de betreffende methoden op een rijtje hebt gezet in een diagram, zul je tijdens het implementeren een veel beter beeld hebben van wat er is gebeurd, en wat er nog moet gebeuren. Hiermee ga je dus niet zomaar blind ergens beginnen met programmeren, maar heb je een strak plan hoe jij de aanpak hiervan gaat doen.

Ons klassendiagram voor het spel, Diving for Treasure: The Sequel ziet er als volgt uit:

*(zie afbeelding 1 of .asta bestand)*



*Afbeelding 1: De volledige klassendiagram van ons spel.*

*(Voor een duidelijker beeld bekijk de bijlagen onderaan waarin de diagram verdeeld is in 4 grote afbeeldingen)*

# Klassen

## DivingForTreasure

Deze klasse zal het hart van heel het spel zijn. In deze klasse zal ook de main staan van het spel. Deze klasse zal extend worden door de klasse GameEngine. Vanuit deze klasse zullen de functies setupGame() en update() worden overgenomen en overschreven. Er staan ook veel functies in deze klasse. Dit zijn alle functies die essentieel zijn tot het spel en in meerdere klassen gebruikt moeten kunnen worden. Hierbij wordt gedacht aan het doodgaan van de speler of het moet verhogen van de score.

## Player

De klasse Player zal alle functionaliteiten bevatten die van toepassing zijn bij de speler van het spel. Dit is de klasse die voor de speler het belangrijkst zal zijn, omdat dit het karakter is dat er bestuurd zal worden door de speler. De klasse wordt extend door de klasse SpriteObject. Vanuit deze klasse worden de update() en keyPressed() functies overgenomen en overschreven. De klasse krijgt als implement de interface ObjectScore. Hierdoor worden de functies printScoreToScreen(), incrementScore() en decrementScore() overgenomen en overschreven.

De constructor krijgt een DivingForTreasure instantie mee. De variable speed zal de beweeg snelheid van de speler bijhouden. De playerXPosition en playerYPosition zal de huidige x en y positie van de speler bevatten en lives zal de score bijhouden van de levens die de speler nog overheeft op dat moment.

## Bomb

De klasse Bomb bevat alle functionaliteiten voor het bom object. Deze zou samen met de klasse BombSet functioneren. In deze klasse worden de interfaces ICollidableWithGameObjects en ObjectInteraction geïmplementeerd waarin de Bomb klasse het bijhorende geluid zou initaliseren. Ook heeft deze klasse een extend naar de SpriteObject klasse Deze wordt dan gekoppeld aan het correcte .mp3 bestand in de constructor. Behalve het initaliseren van het geluid wordt de bewegingssnelheid van dit object ook bepaald in de update() functie. Naast de update functie() bevat deze klasse ook nog een delete() functie die het verwijderen van deze sprite zal uitvoeren wanneer de bomb buiten het speelscherm is of wanneer een speler in aanraking komt met de bom. Ten slotte worden al deze functies in een gezamenlijke functie genaamd action() uitgevoerd.

## Coin

De klasse Coin bevat alle functionaliteiten voor het munt object. Net zoals bij de Bomb klasse loopt deze klas in samenwerking met een andere klas, in dit geval is het de CoinSet klasse. Ook deze klasse heeft de interfaces: ICollideableWithGameObjects en ObjectInteraction en zou weer extended zijn naar de SpriteObject klasse. In de Coin klasse worden de coins aangemaakt en vergelijkbaar met de Bomb klasse wordt hierbij het geluid geïnitialiseerd door middel van de constructor.

## OxygenTank

De OxygenTank klasse zou doormiddel van de ICollidableWithGameObjects & ObjectInteraction interfaces en extend naar de SpriteObject functie alle functionaliteiten rondom de zuurstoftank bevatten. Hier wordt als eerst weer de geluid instantie van de zuurstoftank gemaakt en daarna in de constructor geïnitialiseerd. Ook wordt hier de bewegingssnelheid bepaald in de update() functie en met de delete() functie alle objecten verwijderd. Daarna worden weer al deze functies toegepast in de action() functie die met een @Override te werking gaan met de interface.

## BombSet

Zoals boven al werd gezegd werkt de BombSet klasse gezamenlijk met de Bomb klasse. Nadat de bom in de Bomb klasse is gemaakt zal hij in het spel worden geplaatst met de BombSet klasse. Hierbij wordt de ObjectSet interface gebruikt. In de klasse zelf staan dan ook de volgende functies, ten eerste de makeSet() functie die met een @Override een bepaald aantal nieuwe bommen creëert en in een array zet. Daarna hebben we de spawnSerie() functie die deze gemaakte bommen van makeSet() plaatsen in het spel. En ten slotte hebben wij nog de functie delete() die het verwijderen van deze bommen regelt.

## CoinSet

Deze functie is nogmaals erg vergelijkbaar met de BombSet klasse die hierboven is genoemd. Hij is aangestuurd door de ObjectSet interface, maar heeft daarbij ook nog de ObjectScore interface. Deze klasse heeft als bedoeling om het maken en plaatsen van de munten in het spel te besturen. Daarnaast wordt de huidige score in het spel bijgehouden gebaseerd op het aantal opgepakte munten. Deze worden dan met de printScoreToScreen() functie weergegeven op het speelscherm.

## OxygenTankSet

De klasse OxygenTankSet bevat de functionaliteiten om de zuurstoftanks van de OxygenTank klasse te plaatsen in het speelscherm. Deze klasse heeft de interfaces ObjectSet & ObjectScore. Vergelijkbaar als de CoinSet & BombSet maakt hij de zuurstoftanks aan in een for loop in een array en zet deze daarna 1 voor 1 in het speelscherm. Daarnaast wordt het zuurstofgehalte van de Player bijgehouden en net zoals de Coin score als bijgehouden en op het speelscherm getoond.

## 

## Shark

De Shark klasse heeft alles rondom het Shark object. Dit object wordt weer met een extend naar de SpriteObject functie en reference naar de ICollidableWithGameObjects, ObjectInteraction en OneOfAKindObject bestuurd. In de constructor worden alle .mp3’s voor de sound effects ingeladen. Verder zijn er een deteremineYSpawn() en een spawn() functie om willekeurige een spawn hoogte op het speelscherm te kiezen waarmee hij de haai dan plaatst met de bewegingssnelheid die in de update() functie geregeld wordt. Als deze haai uit het speelscherm gaat wordt dat gedetecteerd door de OutOfScreen() functie en dan wordt de oude haai vervangen voor een nieuwe haai. Deze worden dan allemaal in de action() uitgevoerd zodat de haai zonder problemen geïmplementeerd wordt.

## Clock

De klasse Clock zal alle functionaliteiten rondom een klok bevatten. Een klok zal benodigd zijn om te bepalen wanneer een speler zuurstof verliest of wanneer een powerup gespawnd zal moeten worden. De constructor van een klok moet een world en een interval mee krijgen. De wereld zal de wereld DivingForTreasure zijn en het interval zal de gewenste tijd moeten zijn dat de er iets moet gebeuren.

## PowerUp

Deze klasse bevat alle functionaliteiten rondom het powerup object. Met een extend naar de SpriteObject functie en als interface de ICollidableWithGameObjects, ObjectInteraction en OneOfAKingObject wordt deze bestuurd. Deze PowerUp wordt door middel van de determineNumberToSpawn() functie op een willekeurige positie op het speelscherm geplaatst wanneer de klok gelijk is aan de tijd van de interval in de Clock klasse. Als de powerup object geplaatst wordt zou er een geluid worden afgespeeld die in de constructor mee is gegeven. Verder wordt de bewegingssnelheid hiervan geregeld in de update() functie.

## ResistanceField

De klasse ResistanceField bevat alle functionaliteiten die van toepassing zijn aan het hoge druk gebied. De constructor krijgt een DivingForTreasure instantie mee. ResistanceField wordt extend door de klasse SpriteObject. Vanuit deze klasse wordt de update() functie overgenomen en overschreven. De klasse krijgt als implements de interfaces ICollidableWithGameObjects, ObjectInteraction en OneOfAKindObject mee. Vanuit ICollidableWithGameObjects wordt de functie gameObjectCollisionOccurred overgenomen en overschreven. Van ObjectInteraction wordt de functie action() overgenomen en overschreven. En tot slot wordt vanuit de OneOfAKindObject interface de functie spawn() overgenomen en overschreven.

## 

## Boat

De klasse Boat zal alle functionaliteit bevatten rondom het boot object. De boot zal functioneren als het startpunt van het spel voor de speler. Deze klasse naar wordt extend door de klasse SpriteObject. Vanuit hier worden wordt de functie update() overschreven. De interfaces ObjectInteraction en OneOfAKindObject worden ge implementeert. Uit ObjectInteraction wordt de functie action() toegevoegd als abstracte functie. Uit OneOfAKindObject wordt de functie spawn() toegevoegd als abstracte functie.

## EndScreen

De klasse EndScreen zal alle functionaliteiten bevatten omtrent het eindscherm. De constructor van het eindscherm zal een wereld moeten mee krijgen. Deze wereld zal een DivingForTreasure object zijn. De klasse EndScreen wordt ge extend door SpriteObject. Vanuit deze klassen moet de abstracte functie update() overgenomen worden. De klasse implementeert ook de interface IMouseInput vanuit de aangewezen game engine. Vanuit deze interface wordt de functie mouseClicked(int x, int y, int button) overgenomen. Het eindscherm zal getekend worden als de speler al zijn levens is verloren.Score

In de Score klasse worden alle functionaliteiten toegepast rondom het behouden van de huidige score. Hierbij wordt weer naar de SpriteObject functie extend en worden de ICollidableWithGameObjects, ObjectInteraction & OneOfAKindObject. Hier worden alle textObjecten geïnitialiseerd in de constructor. en daarna met de printScore() functie op het speelscherm aangetoont. Verder worden er met de addToScore(), takeOfScore() en resetScore() de score aangepast en in de update() up to date gehouden.

## GameObject

Wij zullen enige aanpassing aan de game engine maken. Deze aanpassingen zijn besloten na zorgvuldig na te denken of het op een andere manier ook mogelijk is, maar dit was niet het geval. Veel objecten zoals OxygenTank, Coin en Bomb zullen allemaal op dezelfde manier gespawnd worden. Vanwege dit is het efficiënt om deze functies toe te voegen aan de klasse GameObject. De functies determineXSpawn() en determineYSpawn() zijn hiermee toevoegt aan de klasse GameObject.

Naast dat het bepalen van de X en Y spawn locaties van deze objecten allemaal hetzelfde zijn ook de Spawn(), respawn() en outOfScreen() functies allemaal hetzelfde.

SpriteObject erft van GameObject en de klassen OxygenTank, Coin, Bomb en Powerup erven allemaal van SpriteObject. Hierdoor zullen al deze klassen toegang hebben tot deze functies.

# 

# Interfaces

## Object Interaction

Deze interface zal implementeert worden door een klasse als het belangrijk is dat er een actie verricht moet worden. Voor bijvoorbeeld een bom staat het vast dat er een actie verricht moet worden als deze in aanraking komt een speler. De functie action() zal dan overgenomen moeten worden.

## Object Score

Als het zeker staat dat een object zelf een score bijhoudt en dat deze score moet kunnen worden aangepast moet deze interface implementeert worden. De functies printScoreToScreen(), incrementScore() en decrementScore() zullen dan overgenomen moeten worden en verder uitgewerkt moeten worden.

## ObjectSet

Als een object in groepen moeten spawnen zal er een set van gemaakt moeten worden. Er moet dan zeker weten een set gemaakt worden en deze set moet gespawnd worden. Omdat dit een essentieel onderdeel zullen deze zeker uitgewerkt moeten worden. Hierom zal de interface ObjectSet implementeert worden. Deze interface forceert dan het overschrijven en uitwerken van de functies makeSet() en spawnSerie().

## OneOfAKingObject

Als het vaststaat dat een object maar één keer moet voorkomen in het spel zal dit object een aparte spawn functie hebben. Hierom zal de interface OneOfAKindObject implementeert moeten worden. Dan zal er zeker een spawn() functie worden overgenomen en overschreven.

# Gebruikte bronnen

LucidChart. (z.d.). Tutorial UML-klassendiagram. Geraadpleegd op 18 mei 2020, van <https://www.lucidchart.com/pages/nl/tutorial-klassendiagram>

Wikipedia-bijdragers. (2020, 11 mei). Unified Modeling Language. Geraadpleegd op 18 mei 2020, van <https://nl.wikipedia.org/wiki/Unified_Modeling_Language>