OOpD Beroepsproduct

Functioneel en technisch ontwerp

**Student**: Klaas van der Linden 599644

**Student**: Bart Brendeke 598674

**Docent**: Fritz van Deventer

**Vak**: OOPD

**Datum**: 05-04-2018

Inhoud

[Inleiding 2](#_Toc509844829)

[Functioneel ontwerp 3](#_Toc509844830)

[Technisch ontwerp 5](#_Toc509844831)

# Inleiding

Wij zijn 2 studenten die voor het vak Object Oriented Program Development moeten aantonen dat ze een game kunnen maken in een aangeleverde GameEngine met behulp van de programmeertaal Java. De game die wij gaan maken is een Mario-spinoff waarbij de besturing een twist heeft. Informatie m.b.t de game wordt in dit document verder uitgelicht.

# Functioneel ontwerp

Hieronder volgt het functioneel ontwerp waarin wij ons spel beschrijven.

De naam van ons spel is Super Mario Ultraland. Super Mario Ultraland is onze eigen versie van de welbekende Mario platform games van Nintendo. De speler kan Mario besturen om zo prinses Bowser uit de handen van de kwaadaardige Peach te redden door Mario aan het einde van het level te brengen en Peach te verslaan.

Het spel speelt zich af in de eerste wereld van de game Super Mario World. Het perspectief waarin de speler het spel speelt is van de zijkant. De wereld zelf scrolt vooruit als de speler naar links en rechts loopt.

De speler kan Mario besturen met de toetsen W, A, S en D. Hierbij is W: springen, A: naar links, D: naar rechts en S: bukken. Voor de besturing hebben wij echter een twist bedacht. Zodra Mario een vijand dood door op deze te springen, wordt de besturing omgedraaid: W is dan bukken, A: naar rechts, D: naar links en S: springen. Als Mario wederom een vijand dood wordt besturing weer normaal en zodra Mario weer een vijand dood wordt de besturing weer omgedraaid enzovoorts.

Uiteraard hebben de objecten in ons spel interactie tussen elkaar. Als de onderkant van Mario de bovenkant van een vijand raakt, dan wordt de vijand gedood. Als Mario van de zijkant door een vijand wordt geraakt, verliest hij een leven. Mario heeft 3 levens in totaal. Bij 0 levens is Mario af en wordt deze geplaats aan het begin van het level wat dus betekent dat de speler opnieuw moet beginnen. Ook kan Mario interactie hebben met tiles in het spel. Met behulp van tiles willen wij platformen bouwen waar Mario interactie mee heeft. Als Mario hier tegen aan loopt kan hij niet verder dus moet de speler met Mario op/over het platform springen om verder te kunnen gaan.

Ook willen wij powerups toevoegen in de vorm van mushrooms die ergens in de wereld verstopt liggen. Als Mario deze powerups aanraakt krijgt hij bijvoorbeeld een leven erbij of een speedboost.

De speler begint aan het begin van het level met 3 levens. Aan het einde van het spel moet de speler de eindbaas Peach verslaan om het spel uit te spelen. In tegenstelling tot een normale vijand, verliest Mario alle zijn levens als hij Peach aanraakt. Daarom zal er aan het einde van de wereld zich de welbekende bijl bevinden waar Mario op moet springen om Peach te verslaan.

De overige elementen die wij willen toevoegen mochten wij tijd over hebben, zijn een startmenu en een pauzemenu.

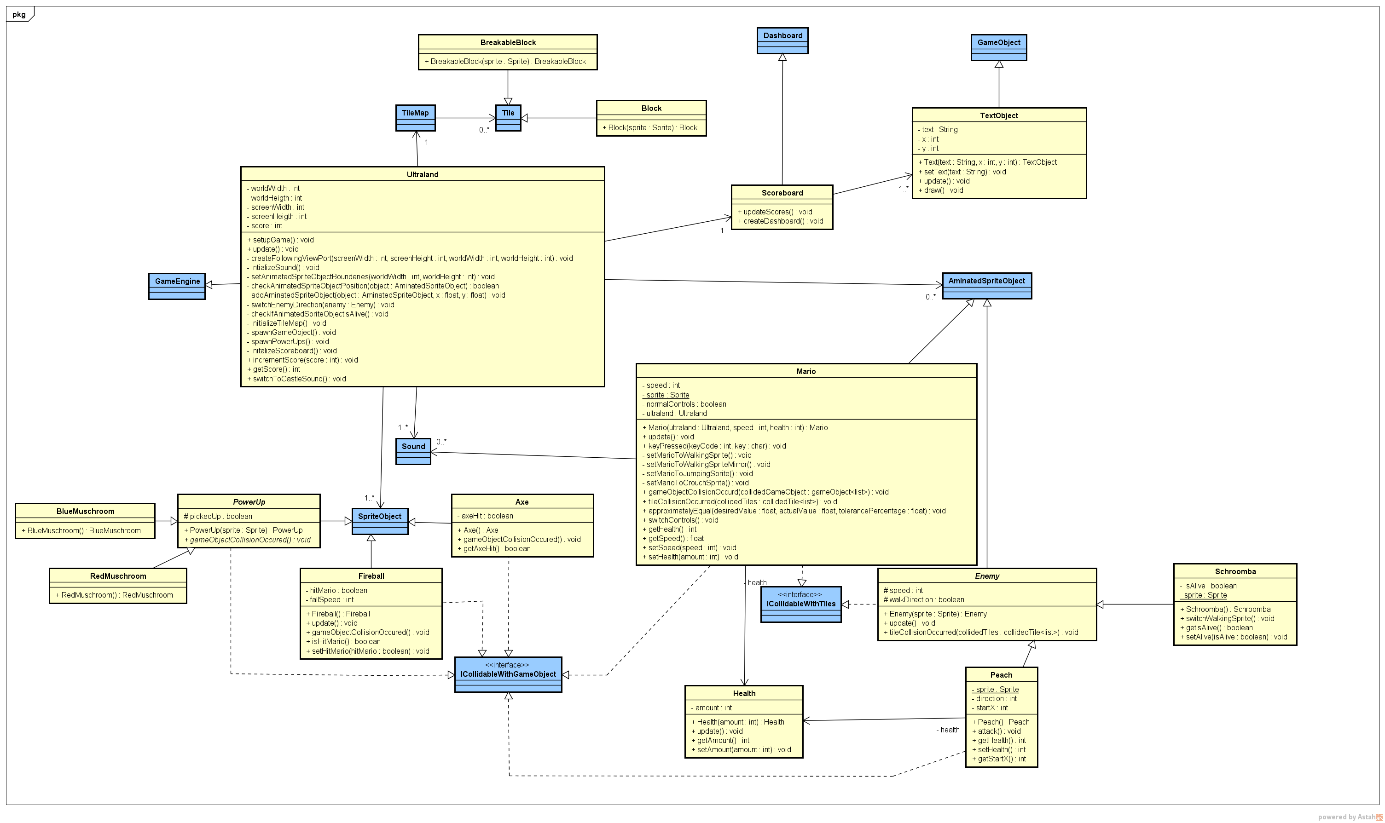
In de Bijlage is een schets te vinden waarin de hoofdelementen uit het spel zijn weergeven.

Hieronder volgt een tabel met daarin de requirements op basis van het MoSCoW principe.

|  |  |  |  |
| --- | --- | --- | --- |
| Nummer | Omschrijving | Prioriteit | Testresultaat |
| 1 | De speler kan Mario besturen met WASD | M | Werkt |
| 2 | Mario kan naar links lopen | M | Werkt |
| 3 | Mario kan naar rechts lopen | M | Werkt |
| 4 | Mario kan springen | M | Werkt |
| 5 | Mario kan bukken | M | Werkt |
| 6 | De sprite van Mario kan veranderen | M | Werkt |
| 7 | Mario heeft levens | M | Werkt |
| 8 | Mario kan levens verliezen | M | Werkt |
| 9 | Er zijn schroomba’s (vijanden) | M | Werkt |
| 10 | Schroomba’s lopen heen en weer | M | Werkt |
| 11 | De sprite van een schroomba kan veranderen | M | Werkt |
| 12 | Een schroomba wordt gedood als Mario erop springt | M | Werkt |
| 13 | Er is een wereld | M | Werkt |
| 14 | De wereld beweegt met Mario mee | M | Werkt |
| 15 | Er zijn tiles(blokken in de wereld) | M | Werkt |
| 16 | Er zijn blokken die Mario kan breken | M | Werkt |
| 17 | Mario heeft collision met blokken | M | Werkt |
| 18 | Er zijn powerups | S | Werkt |
| 19 | Powerups geven Mario een bepaald effect | S | Werkt |
| 20 | De controls worden omgedraaid als Mario een vijand doodt. | S | Werkt |
| 21 | Aan het einde van de wereld is een bossfight met Peach | S | Werkt |
| 22 | Er wordt achtergrondmuziek afgespeeld | S | Werkt |
| 23 | De achtergrondmuziek verandert tijdens de bossfight | C | Werkt |
| 24 | Er is een startmenu | C | Ontbreekt |
| 25 | Er is een pauzemenu | C | Ontbreekt |
| 26 | Er is een eindmenu | C | Ontbreekt |

# Technisch ontwerp

Hieronder staat het klassendiagram van Super Mario Ultraland. De blauwe klassen zijn klassen uit de GameEngine, de gele klassen zijn klassen die wij zelf gaan maken en de rode klassen zijn klassen die wij alleen maken mochten wij tijd over hebben.



Hier volgt een korte toelichting van elke klasse:

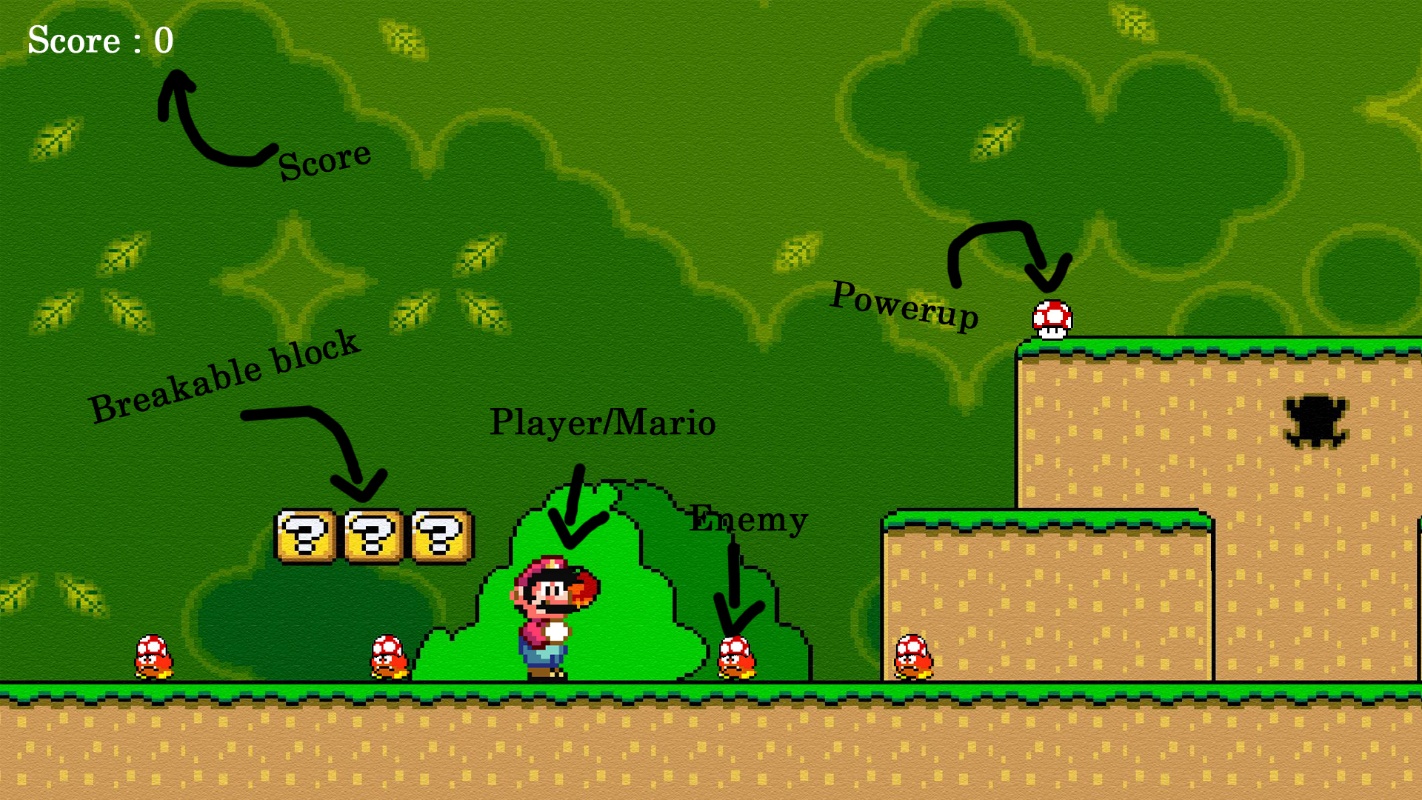
* **Ultraland**: Dit is de Main klasse van onze game. Deze klasse extends GameEngine en zorgt er dus voor dat alle GameObjecten en de wereld worden getekend. Ook maakt deze klasse gebruik van de Sound klasse om zo achtergrond muziek af te spelen tijdens de Game. Tevens maakt deze klasse gebruik van de klasse Tilemap zodat we de tiles kunnen teken in onze wereld. In de klasse bevinden zich ook de AnimatedSpriteObject zoals Mario en de Enemies.

In de klasse maken we gebruik van verschillende zelfgemaakte functies. Als eerst maken we gebruikt van een functie die een viewport maakt. Deze viewport zorgt ervoor dat er een wereld wordt weergeven en dat deze wereld verder scrollt als de gebruiker met Mario heen en weer loopt. De functie setAnimatedSpriteObjectBoundaries zorgt ervoor dat de SpriteObject niet uit de wereld kunnen lopen, maar binnen de wereld blijven.

De functie checkAnimatiedSpriteObjectPosition kijk of het SpriteObject zich nog binnen de wereld bevindt. Met de functie addAnimatedSpriteObject kunnen er SpriteObject worden toegevoegd wat handig is als je een grote hoeveelheid enemies wil toevoegen door bijvoorbeeld gebruik te maken van deze functie met een for-lus, dit gebeurt dan ook in spawnGameObject. Met de functie checkIfAnimatedSpriteObjectIsAlive wordt gekeken of er animatedSpriteObjecten zijn die niet meer moeten worden getekend, en verwijdert deze ook. De functie switchEnemyDirection zorgt ervoor dat enemies de andere kant op lopen mochten zij tegen andere GameObject of tiles oplopen. In spawnPowerUps worden alle power ups gespawned in de map. Met initalizeTileMap worden de tiles in de wereld getekend. Met initalizeSound wordt er achtergrond muziek gedeclareerd. InitalizeScoreboard tekent het scoreboard in beeld met de behaalde punten. De methode switchToCastleSound veranderd de muziek naar de bossfight muziek. En als laatste returned getScore de huidige score van de speler.

* **Mario**: Dit is de klasse waarin als met betrekking tot de speler/Mario wordt geregeld. Als variabelen hebben wij een snelheid die bepaald hoe snel Mario loopt, een sprite die bepaalt wat de huidige sprite is voor Mario bijv: lopen of bukken. Deze sprite wordt meegeven aan de super klasse animatedSpriteObject. Daarnaast is er een boolean normalControls die op true of false staat. Deze wisselt van true naar false als Mario collision heeft met een enemy wat wordt gecontroleerd gameObjectCollisionOccured. Dat is ook de reden dat Mario de interface ICollidableWithGameObject implementeert. De functie keyPressed handelt de besturing van Mario af. Ook zijn er verschillende functies om Mario van sprite te wisselen. Mario implementeert tevens de klasse ICollidableWithTiles wat de mogelijkheid geeft om platformen te bouwen m.b.v. tiles waar Mario dan over heen moet springen. De methode approximatelyEqual maakt een tolerantie aan voor afwijking tussen Mario en een enemy. Met switchControls wordt de boolean normaleControls true of false gezet. En dan hebben we nog getters en setters voor health en speed.
* **Health**: De klasse health bepaalt de hoeveelheid levens voor een gameObject. Zowel Mario (de speler) als Peach (de eindbaas) maken gebruik van deze klasse. De klasse bevat een int amount zodat wij kunnen instellen hoeveel health een gameObject krijgt. Verder heeft hij nog een getter en een setter voor amount (hoeveelheid van levens).
* **Enemy**: Dit is een abstracte klasse die de eigenschappen voor enemies bepaald. Zo heeft elke enemy een bepaalde beweegsnelheid en een sprite. Ook heeft elke enemy een update functie waarin het bewegen van de enemy kan worden geregeld. Enemy extends AnimatedSpriteObject omdat alle enemy’s AnimatedSpriteObjecten zijn, ook implementeert hij ICollidableWithTiles zodat de enemy niet door een blok kan lopen.
* **Schroomba:** Dit is een normale enemy die door de map loopt, hij extend ook de klassen Enemy. Schroomba heeft een boolean isAlive die aangeeft of de schroomba nog getekend moet worden. Daarnaast heeft hij nog een Sprite die omgedraaid wordt met switchWalkingSprite voor het geval dat Schroomba ergens tegenaan loopt, uiteraard loopt de Schroomba dan ook de andere kant op. Ook heeft de Schroomba nog een getter en een setter voor isAlive.
* **Peach:** Peach is de klassen van de eindbaas, deze extends Enemy. Peach heeft een Sprite, een startX die aangeeft waar Peach moet spawnen en een direction waaruit kan afgeleid worden welke kant Peach moet oplopen. De methode attack spawned de attack van Peach, namelijk 5 FireBalls. Verder heeft Peach nog getter en een setter voor Health.
* **Axe:** De klassen Axe is bedoelt om Peach mee te kunnen verslaan, Axe implementeert ICollidableWithGameObjects en extends SpriteObject. GameObjectCollisionOccured kijkt of Mario hem heeft geraakt en zet axeHit op true. Daarnaast returned getAxeHit de boolean axeHit.
* **Fireball:** Fireball is de klassen die de aanval van Peach opbouwt, Fireball extends SpriteObject en implementeert ICollidableWithGameObjects. Fireball heeft een boolean die aangeeft of hij Mario heeft geraakt en een fallSpeed voor zijn valsnelheid. De methode gameObjectCollisionOccured kijkt of hij Mario heeft geraakt, geeft de speler ook damage. Verder is er nog een getter en setter voor hitMario.
* **PowerUp:** PowerUp is een abstracte classe die de eigenschappen bepaald voor de powerups, PowerUp extend SpriteObject en implementeert ICollidableWithGameObjects. PowerUp heeft een boolean pickedUp die aangeeft of hij is opgepakt. En een abstracte methode gameObjectCollisonOccured.
* **RedMuschroom:** RedMuschroom extend PowerUp, als Mario RedMuschroom aanraakt krijgt hij een leven erbij.
* **BlueMuschroom:** BlueMuschroom extend PowerUp, zodra Mario BlueMuschroom aanraakt krijgt hij verhoogde snelheid.
* **Block:** Block extends Tile en heeft een Sprite. Block wordt getekend in de Tilemap in Ultraland.
* **BreakableBlock:** BreakableBlock extends Tile en heeft een Sprite. BreakableBlock wordt getekend in de Tilemap in Ultraland.
* **Scoreboard:** Scoreboard heeft een of meerdere TextObjecten, Scoreboard extends Dashboard. De methode updateScores werkt de scores bij. En createDashboard maakt een Dashboard aan om de scores op weer te geven.
* **TextObject:** TextObject extends GameObject, hij kan een tekst weergeven voor het Scoreboard. TextObject heeft een text wat je wilt weergeven, een x en een y positie. Verder heb je een methode setText voor het setten van text en een draw die het textobject tekent.

# Bijlage



Figuur Schets