CS2: THE KISS OF DEATH

Tom Kaal, Sander Koenders en Frank Blom

02/12/2015

HAN ICA – OOSE OOAD

Harrie van Seters

Inhoudsopgave

1. Inleiding

Welkom bij de kiss of Death. Dit is een spel waarbij een monster, strijder of elf kan zijn en het doel is om het monster te vinden die de gedaante heeft aangenomen van een strijder, een elf of een dwerg. Samen met je team moet je proberen dit monster te vinden, maar hij kan natuurlijk ook bij jou in het team zitten. Val wezens aan van de tegenpartij of van je eigen team. Weet je iemand te verslaan en is dit het monster, dan verandert hij in zijn ware 'gore' gedaante. De enige manier om het monster te verslaan, is door hem te kussen. Wanneer je het monster hebt gevonden heeft jou team gewonnen. Weet het monster iedereen te verslaan, dan wint het monster. Kus je iemand van de tegenpartij en is het geen monster? Dan verander je in een kikker en moet je maar hopen dat een medespeler jou kust, want als kikker zijnde kun je niks beginnen.

In dit document is beschreven wat het spel bevat, hoe het spel werkt en hoe het er in werkende code zou moeten opleveren. Door gebruik te hebben gemaakt van verschillende UML diagrammen hebben wij het systeem zo goed mogelijk ontworpen.

2. Actoren en NPI

2.1. Actoren

Als actoren hebben wij:

- Speler
- Systeem

2.2. NPI

Spel	Spel	Spel	
Detail			
Overzicht			
Startwaarden			
Variabelen			
Waarden			
File			
Database	Database		
Kamers	Kamers	Kamers	
Grot	Grot	Grot	
Deuren	Deuren	Deuren	
Elf	Elf	Elf	
Dwerg	Dwerg	Dwerg	
Monster	Monster	Monster	Gedaante
Strijder	Strijder	Strijder	
Wezens	Wezens	Wezens	Levenspunten
			Energie
			Kracht
			Geslacht
Boodschappen	Boodschappen	Boodschappen	
Vorm			
Schematisch			
tegel	tegel	tegel	
speelveld	speelveld	speelveld	
Muurdelen	Muurdelen	Muurdelen	
Vakje	Vakje	Vakje	
Gedaante	Gedaante		
Kussen			
Speler	Speler	Speler	

Medespeler			
Type			
Stategie			
Kikker	Kikker	Kikker	
Levenspunten	Levenspunten		
Energie	Energie		
Kracht	Kracht		
Geslacht	Geslacht		
Attribuut	Attribuut	Attribuut	
Zwaard	Zwaard	Zwaard	
Harnas	Harnas	Harnas	
Knuppel	Knuppel	Knuppel	
Gevecht			
Gaten	Gaten	Gaten	
Vloer			
Bezemsteel	Bezemsteel	Bezemsteel	
Vitamines	Vitamines	Vitamines	
Winnaar			
Intranet			
Server			
Client			
Clientvenster			
Spelerskarakter			
Object	Object	Object	

3. Furps

Functionality

- Een spel
- Items oppakken.
- Door deuren heen lopen.
- Over gaten heen springen.
- Boodschappen sturen.
- De speler kan op een vakje staan.
- De grot (oftewel speelveld) kan meerdere vormen aannemen.
- Elkaar aanvallen waardoor de levens punten gewijzigd worden.
- Een speler kan zich verdedigen.
- Een speler kan zich bewegen in het spel maar alleen wanneer de speler genoeg levenspunten heeft.
- Een speler kan zich ook sneller dan normaal lopen, dit kost wel het dubbele aantal punten.
- Een speler kan een andere speler kussen.
- Vliegen met een bezem om naar een willekeurige plaats te vliegen.
- Een speler kan vitamines opeten om levenspunten aan te vullen.
- Elk type wezen kan een bijbehorend attribuut dragen.
- Een speler kan een attribuut oppakken en neerleggen
- Het bijhouden van een attribuut kost extra levenskracht.
- Een speler kan zich aanmelden in systeem om mee te doen.
- Het spel start pas wanneer er 9 personen in zitten.
- Het spel genereert 3 teams met daarin 3 wezens van dezelfde types.
- Eén van de gegenereerde wezens wordt automatisch een monster.
- Het spel wordt afgesloten wanneer het monster verslagen is of wanneer alle spelers dood zijn.
- Het stel wordt gespeeld in 2D.
- Speler kunnen chatten met andere speler die van dezelfde types zijn.
- Bij 0 levenspunten is het wezen dood.
- Het spel kan altijd gestopt worden.

Usability

- Het spel moet gemakkelijk speelbaar zijn

Reliability

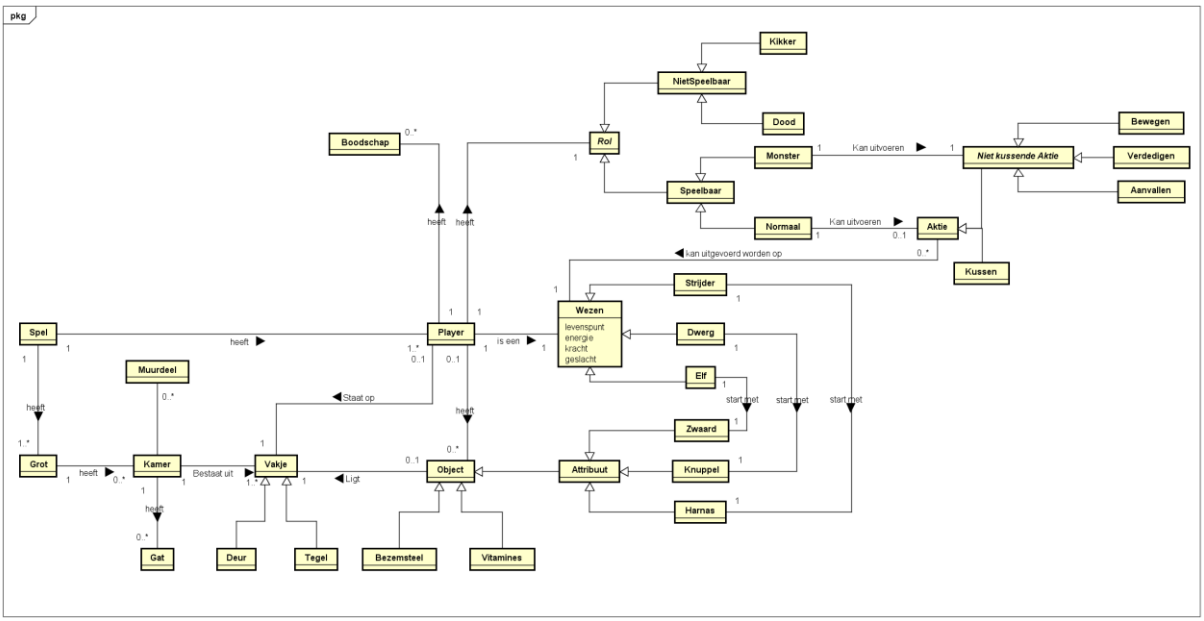
Performace

- Realtime communicatie

Supportability

- Het spel moet op het intranet werken
- Het draait op een server

4. Use Case Diagram



5. Use Cases

5.1. UC1: Verdedigen

Primary Actor: <Speler>	
Stakeholders: <Tegenspeler>	
Brief Description: Een speler verdedigt zichzelf tegen een aanval van een <Tegenspeler>. Het verdedigen kost altijd energie. Of en hoeveel levenspunten er af gaan is afhankelijk van of de aanval lukt en hoe sterk de <Speler> en de <Tegenspeler>is.	
Preconditions: De <Speler> heeft al zijn levens nog en staat naast een <Tegenspeler>. De <Speler> word succesvol aangevallen door de <Tegenspeler>maar verdedigd op tijd.	
Postconditions: De <Speler> heeft minder levenspunten wanneer het gevecht is afgelopen.	
Main Success Scenario:	
1. De <Speler> geeft aan te willen verdedigen.	2. Het systeem haalt van de <Speler> energie af.
.	3. Het systeem berekent op basis van kans of er schade aan de <Speler> wordt toegebracht.
	4. Het systeem berekent op basis van de hoeveelheid kracht en attributen van zowel de <Speler> als <Tegenspeler>, hoeveel levenspunten er af gehaald moeten worden.
Extensions (Alternative Flows):	
	3a1 [Systeem berekend dat er geen schade aan de <Speler> word toegebracht] 3a2 Er worden levenspunten afgehaald van de <Tegenspeler>.

5.2. UC2: Kussen

Primary Actor: <Speler>	
Stakeholders: <Tegenspeler>, <Medespeler>	
Brief Description: Een <Speler> kust een <Tegenspeler>.	
Preconditions: De <Speler> staat naast een <Tegenspeler>en heeft voldoende levenspunten om te kussen.	
Postconditions: De <Tegenspeler> veranderd in een kikker.	
Main Success Scenario:	
1. De <Speler> geeft aan te willen kussen	2. Het systeem haalt de helft van de <Speler> zijn levenspunten er af.
.	3. Het systeem veranderd de <Tegenspeler> in een kikker.
Extensions (Alternative Flows):	
	3a1 [<Tegenspeler> is het monster] 3a2 Het systeem geeft aan dat het monster dood is en het team waar de speler in zit gewonnen heeft.
	3b1 [<Tegenspeler> is een kikker van een ander type] 3b2 [Use Case eindigt]
	3c1 [<Medespeler> is een kikker van een hetzelfde type] 3c2 Het systeem verander de kikker terug naar zijn originele gedaante.

5.3. UC3: Laten vallen <Attribuut>

Primary Actor: <Speler>	
Stakeholders: -	
Brief Description: Een <Speler> laat een attribuut vallen.	
Preconditions: De <Speler> is in bezit van een attribuut.	
Postconditions: De <Speler> is niet meer in het bezit van een attribuut.	
Main Success Scenario:	
1. De <Speler> geeft aan zijn attribuut te willen laten vallen	2. Het systeem haalt het attribuut weg bij de <Speler>.
.	3. Het systeem legt het attribuut neer op het vakje waar de <Speler> zich bevind.

5.4. UC4: Aanvallen

Primary Actor: <Speler>	
Stakeholders: <Tegenspeler>	
Brief Description: Een <Speler> valt een andere speler aan	
Preconditions: Het karakter van de <Speler> is geen kikker en de <Speler> heeft genoeg energie	
Postconditions: Levenspunten zijn afgeschreven	
Main Success Scenario:	
1. De <Speler> geeft aan een <Tegenspeler> te willen aanvallen.	2. Systeem vraagt op wie de aanval uitgevoerd moet worden
3. De <Speler> geeft aan op welke <Tegenspeler> de aanval uitgevoerd moet worden	4. Het systeem checkt of er een <Tegenspeler> aanwezig is op de aan te vallen locatie.
	5. Het systeem valt aan op de aan te vallen locatie met <<init>> vertraging.
	6. Het systeem controleert of er verdedigd word door de <Tegenspeler>.
	7. Het systeem berekent de slagingskans van de aanval.
	8. Het systeem bepaald m.b.v. de slagingskans de winnaar en de verliezer van de aanval.
	9. Het systeem verminderd de levens punten van de verliezer.
	10. Het systeem verminderd de energie van de <Speler>.
Extensions (Alternative Flows):	
	9a1 [<Speler> heeft te weinig levenspunten na de aanval van de <Tegenspeler>] 9a2 De <Speler> wordt uit het spel verwijderd.
	9b1. [<Tegenspeler>heeft te weinig levenspunten na de aanval van de <Speler> en is het monster] 9b2 De <Tegenspeler> verandert in zijn werkelijke ‘gore’ gedaante.

5.5. UC5: Chatten

Primary Actor: <Speler>	
Stakeholders: <Medespeler>	
Brief Description: <Speler> van hetzelfde type kunnen elkaar boodschappen sturen.	
Preconditions: De zendende <Speler> en de ontvangende <Medespeler> zijn van hetzelfde type, de <Speler> is geen kikker.	
Postconditions: Er is een boodschap verzonden naar de <Medespeler>.	
Main Success Scenario:	
1. De <Speler> geeft aan een boodschap te willen sturen.	2. Het systeem vraagt om naar welke <Medespeler> het bericht gestuurd moet worden
3. De <speler> geeft aan naar welke <Meldespeler> het bericht moet	2. Het systeem vraagt naar het bericht wat verstuurd moet worden.
3. De <Speler> geeft het bericht wat verstuurd moet worden.	4. Systeem geeft de op optie om het bericht te sturen.
4. De <Speler> geeft aan het bericht definitief te willen versturen.	5. Systeem verstuurd bericht.

5.6. UC6: Bewegen

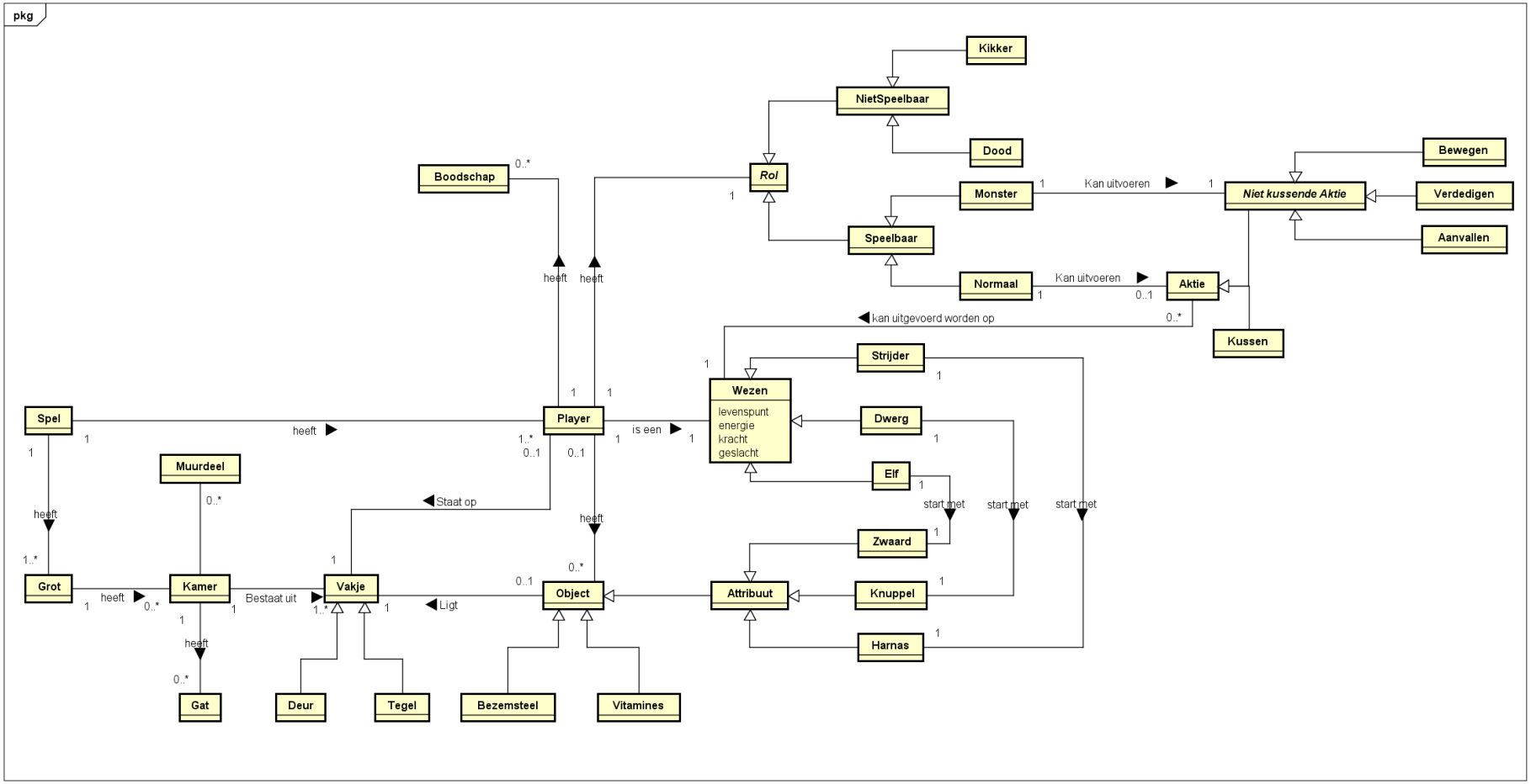
Primary actor: <Speler>	
Stakeholders:	
Brief description: De <Speler> kiest een vakje uit om zich naar toe te bewegen. Op het uitgekozen vakje kan een object liggen die de <Speler> automatisch op pakt wanneer hij op dat vakje terecht komt. Het kost 2 tot 6 energie(afhankelijk van de situatie) om zich te verplaatsen. Voor een aangrenzende vakje 2 energie, een gat over springen kost 4 energieën een transitie van kamers kost 6 energie.	
Preconditions: <Speler> heeft meer dan 10 energie	
Postconditions (Success Guarantee): <Speler> staat op een ander vakje	
Main Success Scenario (Basic Flow):	
Actor Action	**System Responsibility**
1 <Speler> kiest om te bewegen.	2 Systeem vraagt naar welk vakje de <Speler> wilt.
3 <Speler> kiest een vakje.	4 Systeem vraag <Speler> welke snelheid hij zich wil verplaatsen
5 <speler> geeft snelheid aan	6 Systeem verlaagt <Speler> zijn energie op basis van de snelheid.
	7. Systeem verplaatst <Speler> naar gekozen vakje.
	8 Systeem geeft object aan <Speler>.
Extensions (Alternative Flow):	
	2a1 [<Speler> geeft aan met de bezem te willen bewegen] 2a2 Systeem hervat bij stap 7.
	4a [<Speler> kiest vakje buiten bereik] >> [Use Case eindigt]
	4b [Tussen gekozen vakje en <Speler> is een gat] >> [Systeem verlaagt energie van <Speler> met 4]

	4c [Gekozen vakje is een deur] >> [Systeem verlaagt energie van <Speler> met 6]

5.7. UC7: Aanmelden

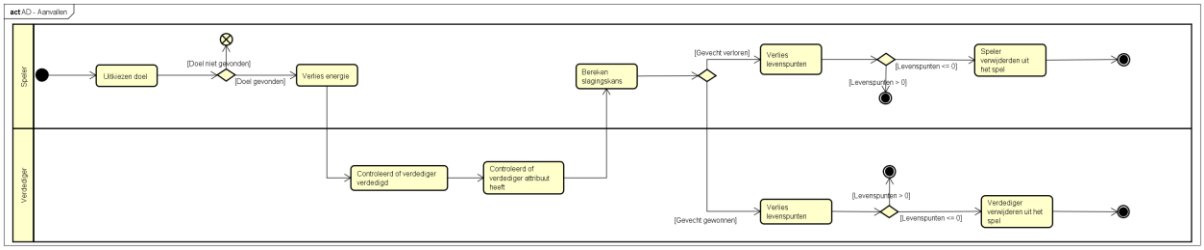
Primary actor: <Speler>	
Stakeholders: <Medespelers>	
Brief description: De <Speler> meldt zich aan bij een spel. Waarna het spel kan starten.	
Preconditions: Server is opgestart en er is nog minimaal één plaats beschikbaar.	
Postconditions (Success Guarantee): <Speler> heeft zich aangemeld voor een spel en de teams zijn gevormd	
Main Success Scenario (Basic Flow):	
Actor Action	**System Responsibility**
1. <Speler> kiest voor aanmelden	2.Systeem meldt <Speler> aan bij het spel.
	3.Systeem geeft aan genoeg <Medespelers> te hebben in de teams.
	4.Systeem formeert teams.
	5.Systeem start het spel

6. Domain model

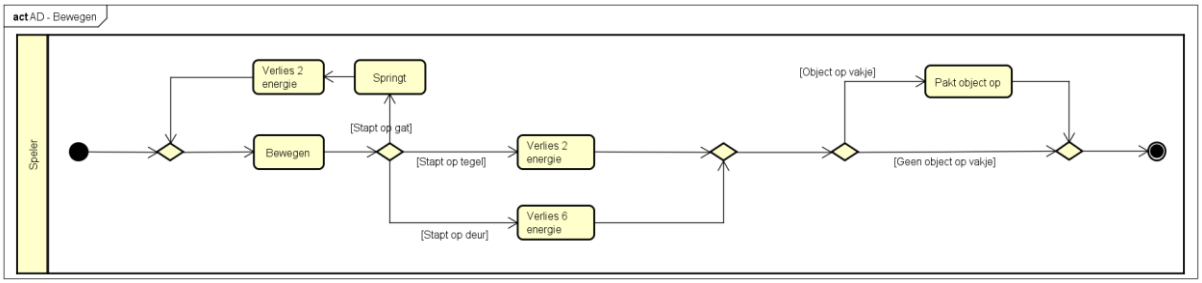

pkg
Kikker
NietSpeelbaar
Dood
Boodschap
0..*
Rol
1
Monster
1
Kan uitvoeren
1
Niet kussende Aktie
Bewegen
Verdedigen
Aanvallen
Speelbaar
heeft
heeft
Normaal
1
Kan uitvoeren
0..1
Aktie
0..*
kan uitgevoerd worden op
Kussen
Strijder
1
1
Wezen
levenspunt
energie
kracht
geslacht
Spel
1
1
heeft
Player
1..*
0..1
0..1
1
1
is een
1
Dwerg
1
Elf
1
Muurdeel
0..*
Staat op
heeft
start met
start met
start met
heeft
Zwaard
1
1..*
1
0..*
Grot
1
heeft
0..*
Kamer
1
Bestaat uit
1..*
Vakje
1
Ligt
0..1
Object
Attribuut
Knuppel
1
1
heeft
Harnas
1
0..*
Gat
Deur
Tegel
Bezemsteel
Vitamines

7. Activity Diagram

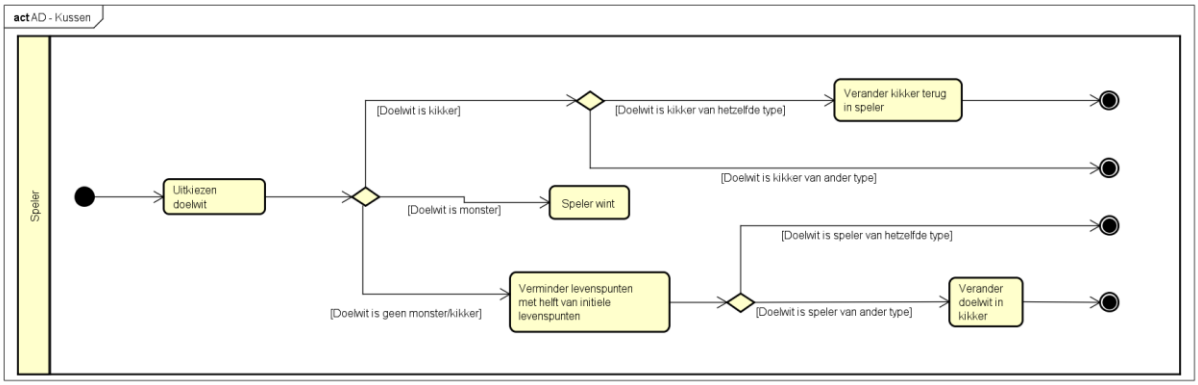
7.1 AD Aanvallen



7.2 AD Bewegen

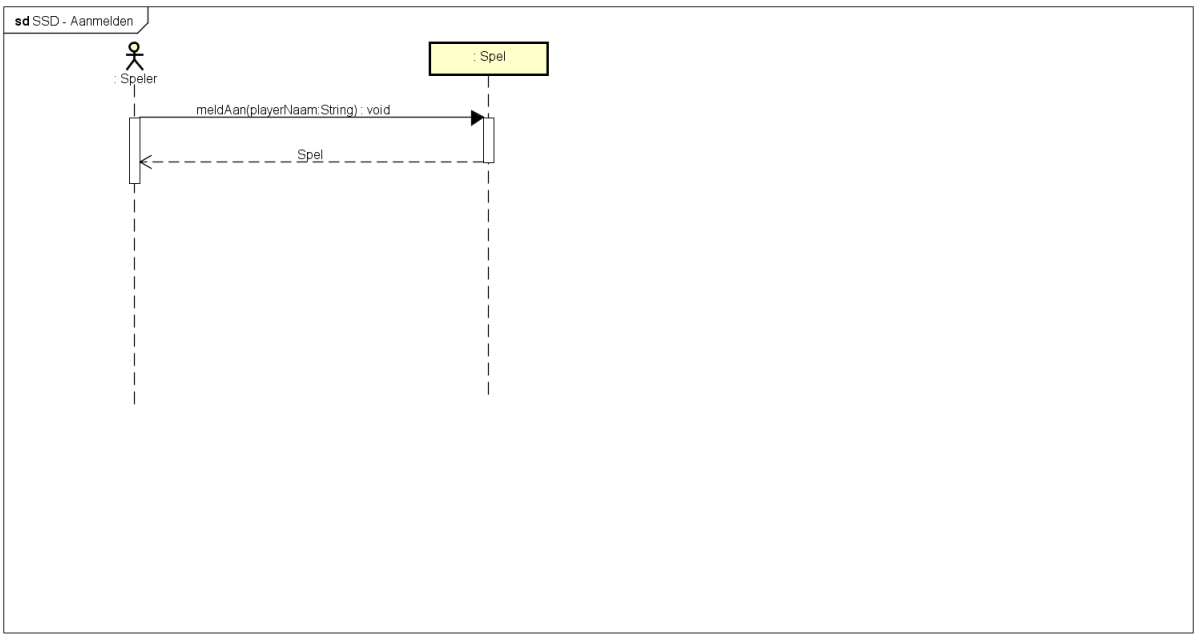


7.3 AD Kussen

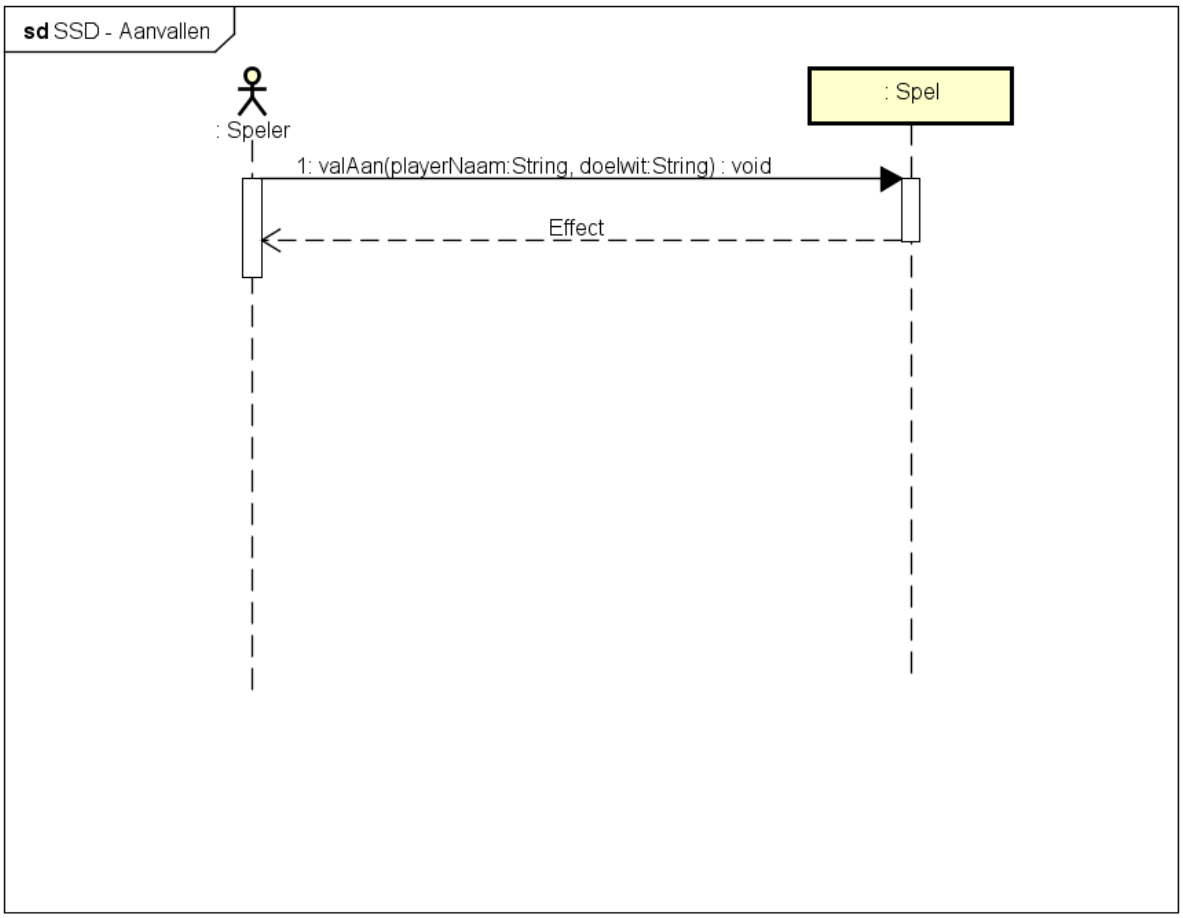


8. System Diagrams & System Sequence Diagrams

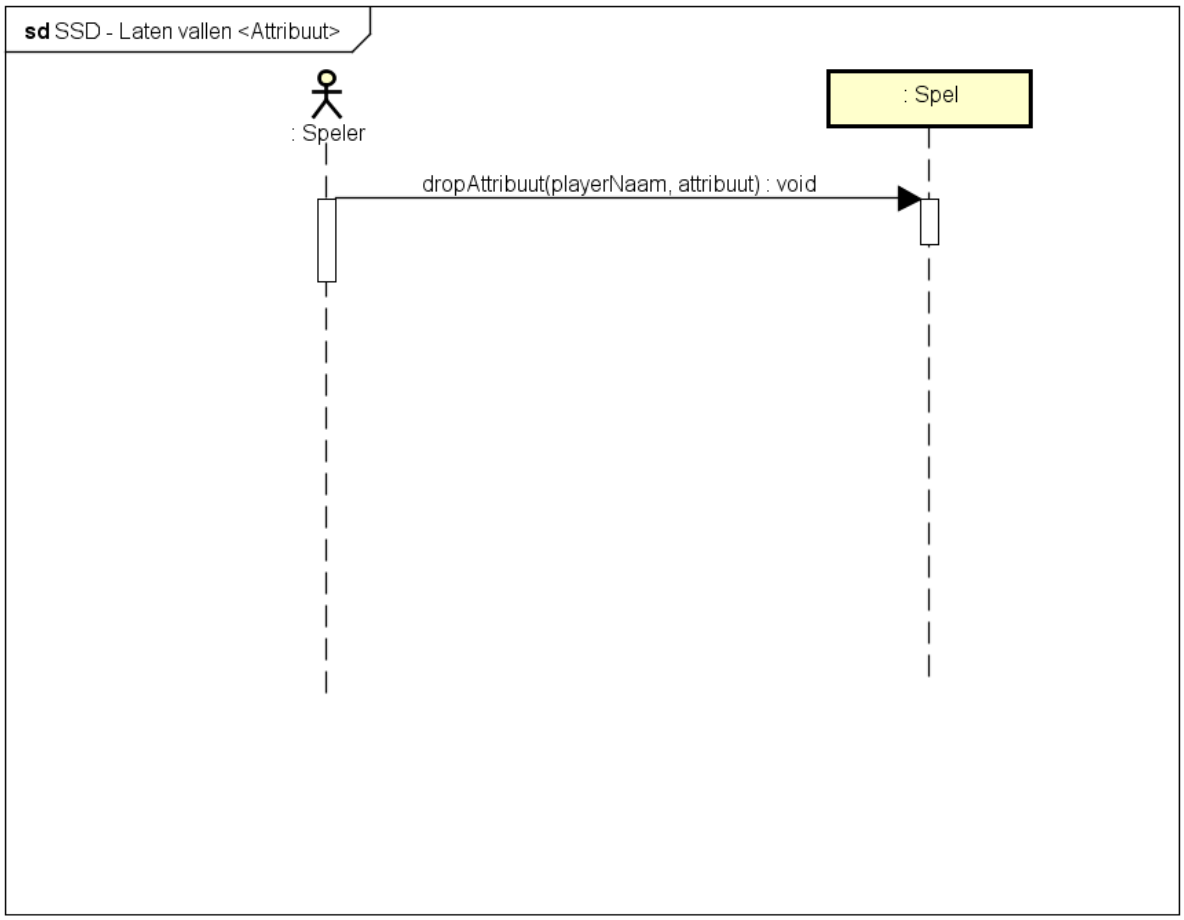
8.1. SSD Aanmelden



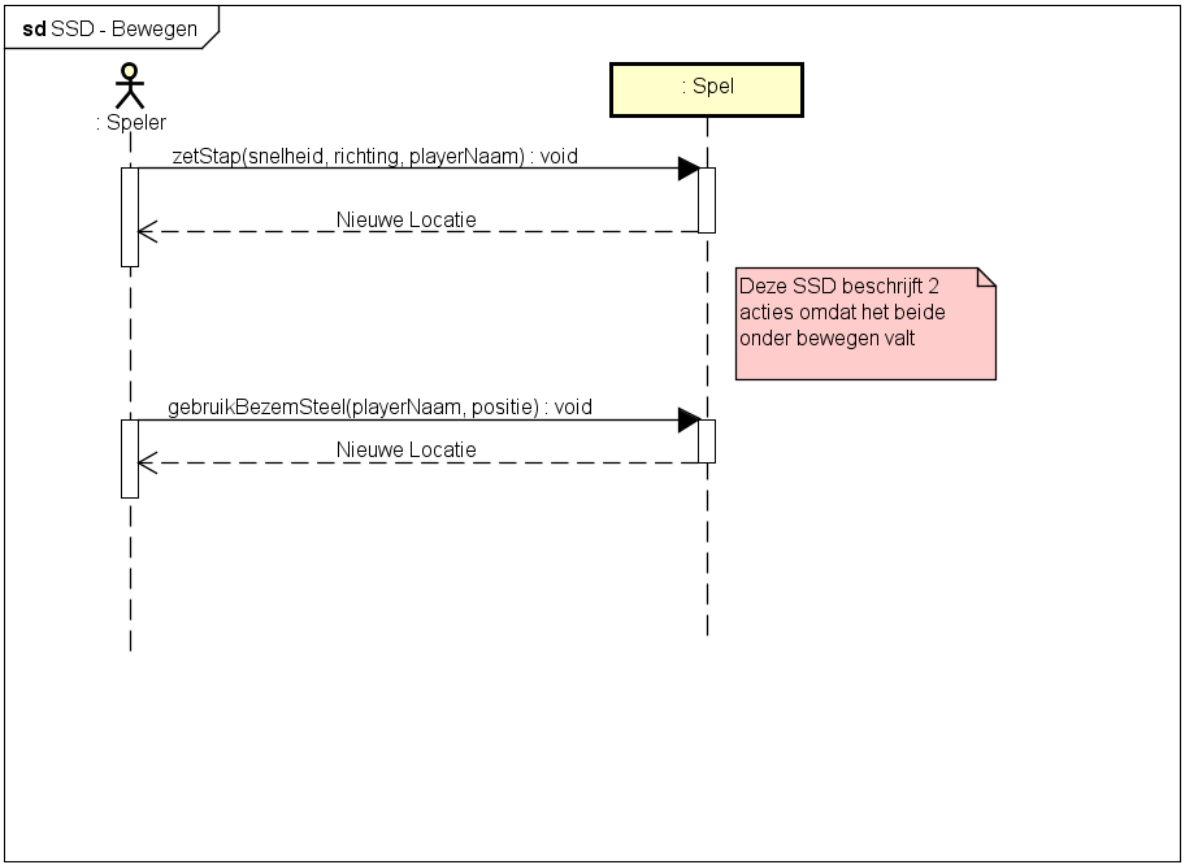
8.2. SSD Aanvallen



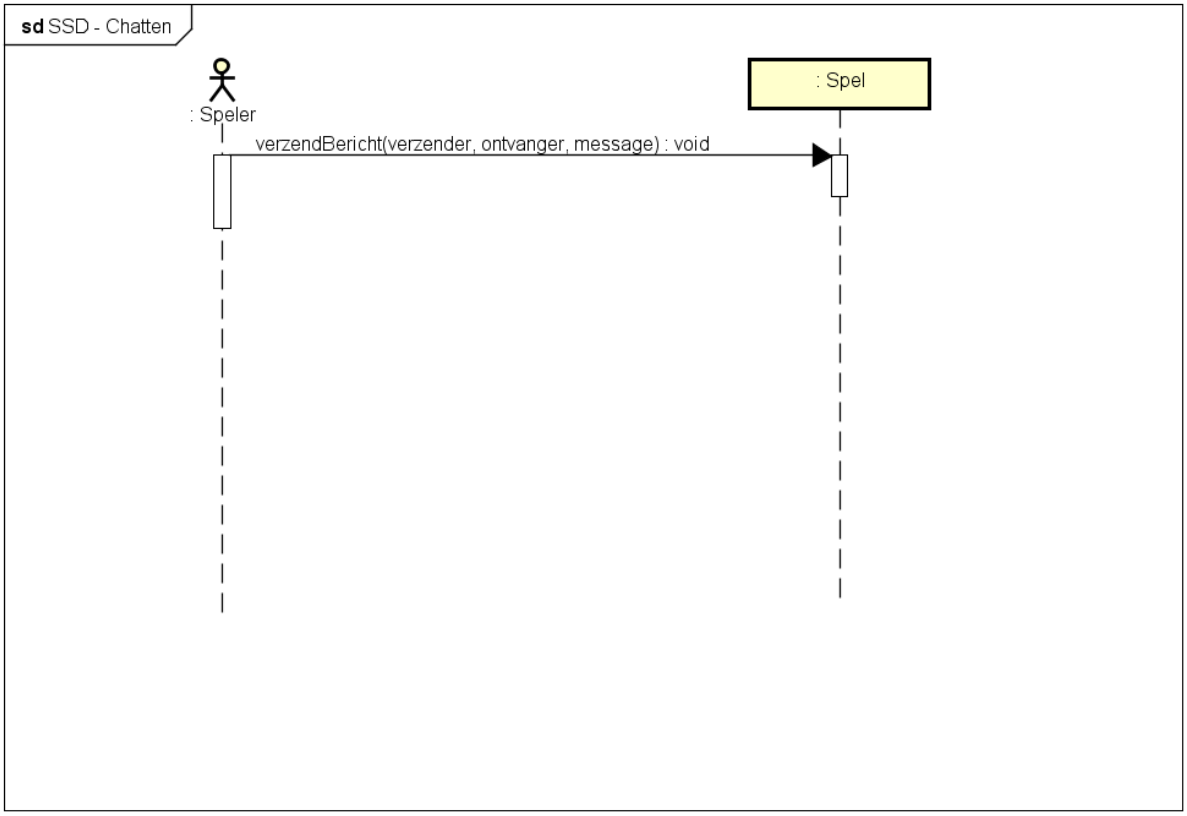
8.3. SSD Laten vallen <Attribuut>



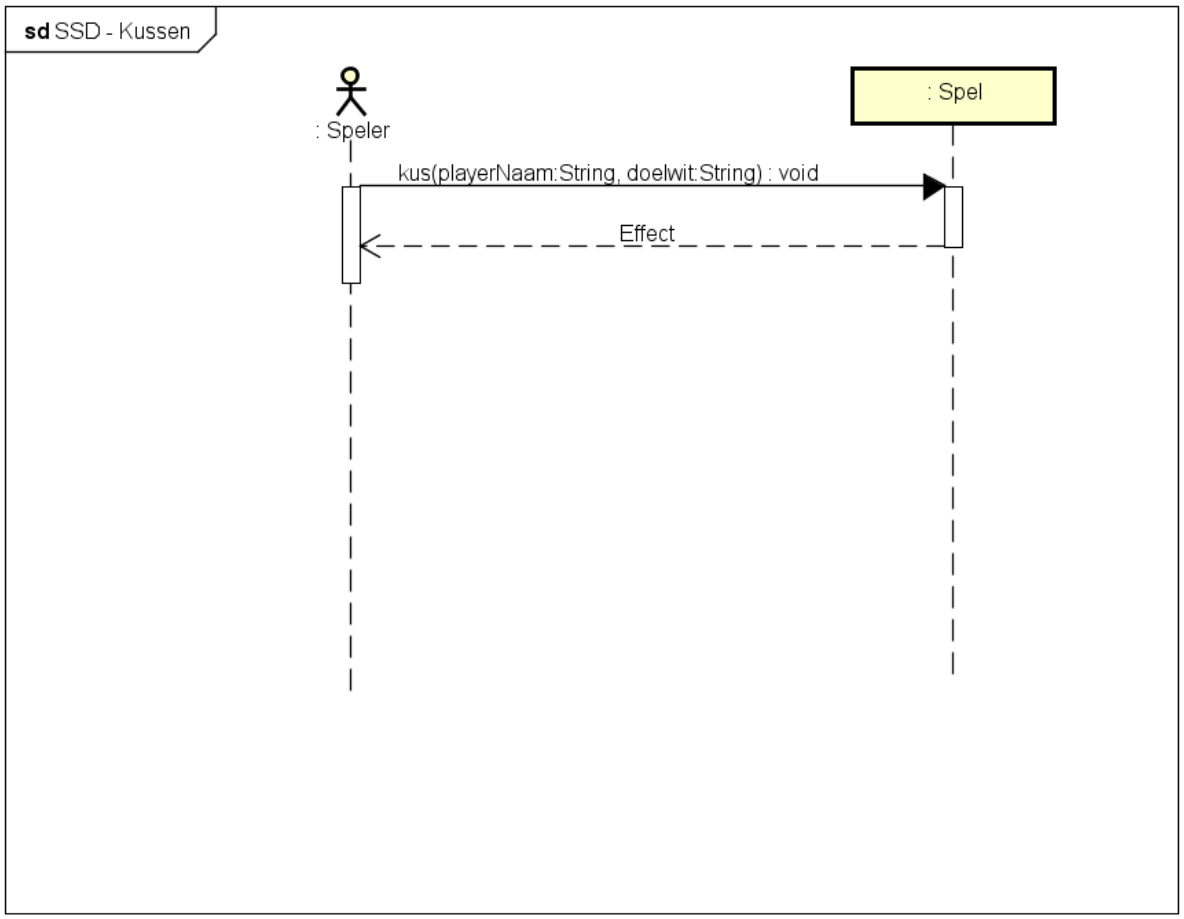
8.4. SSD Bewegen



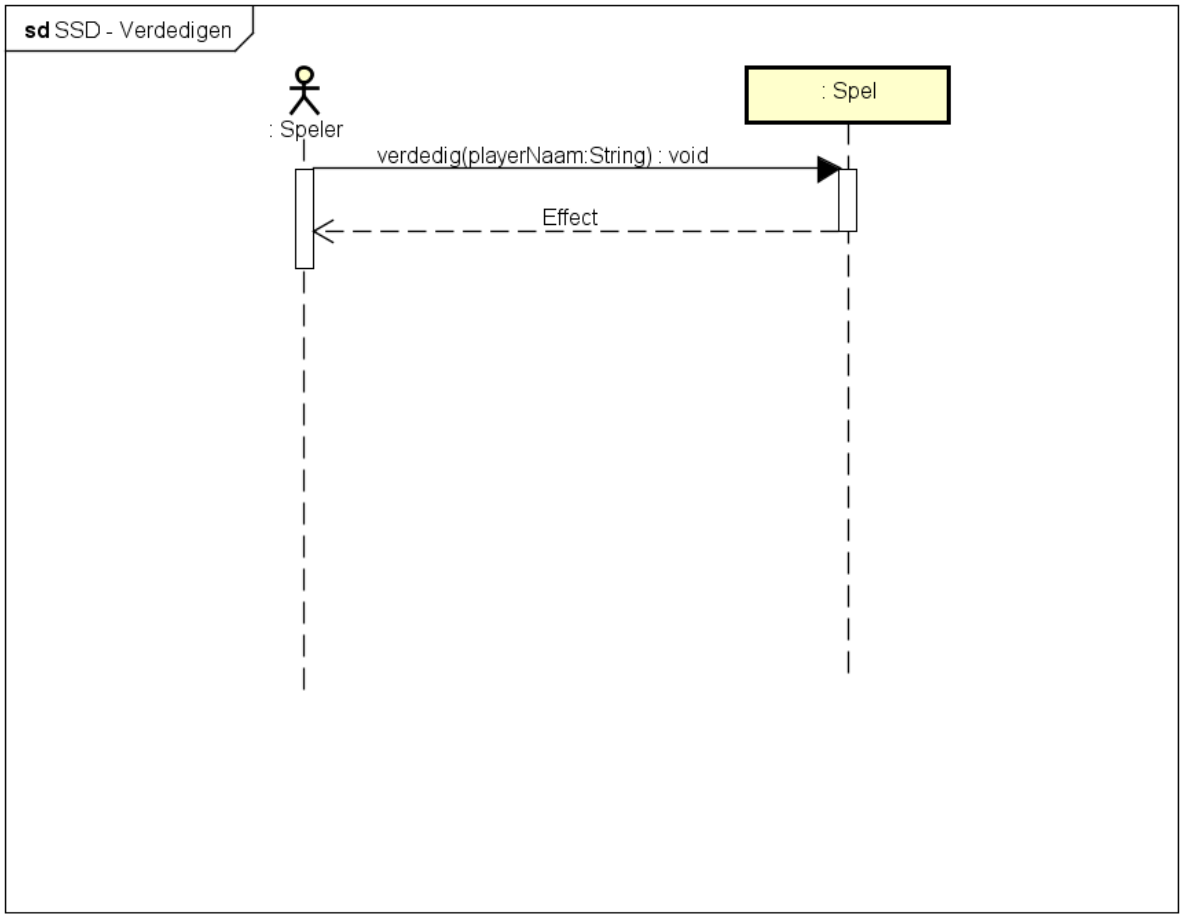
8.5. SSD Chatten



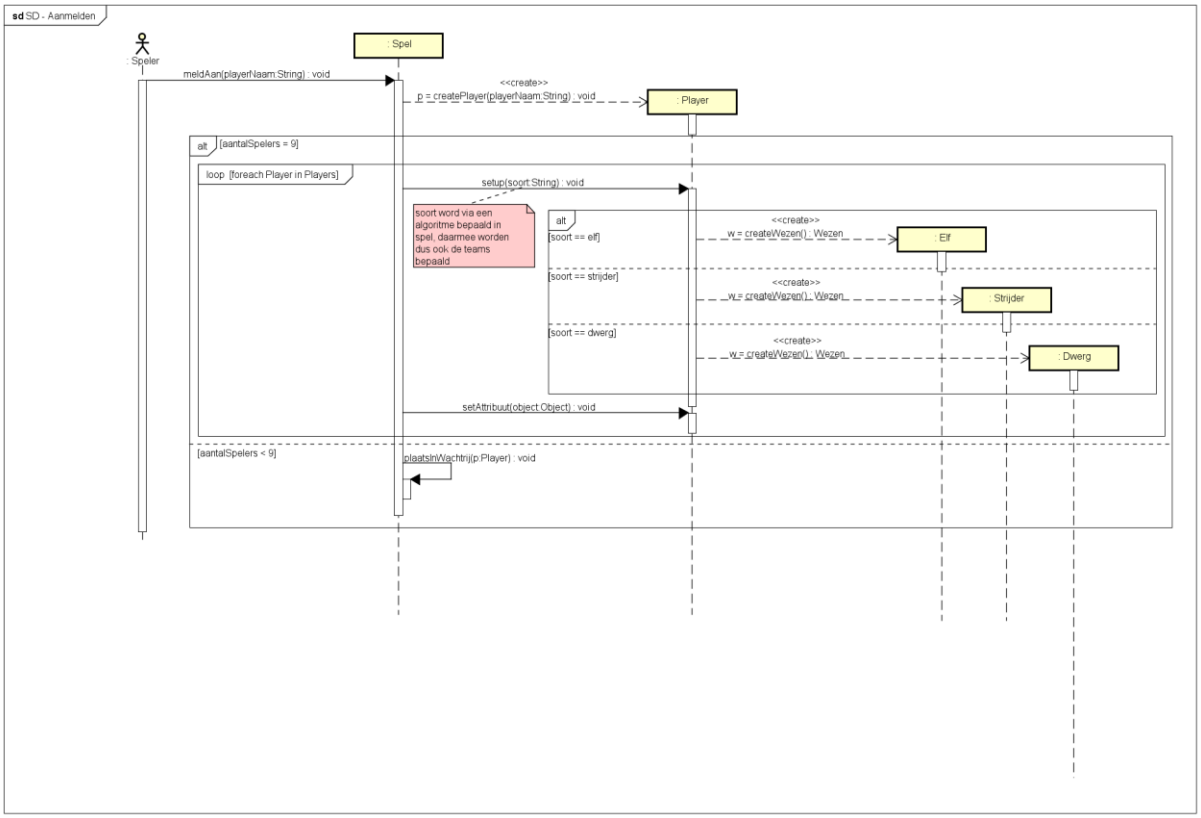
8.6. SSD Kussen



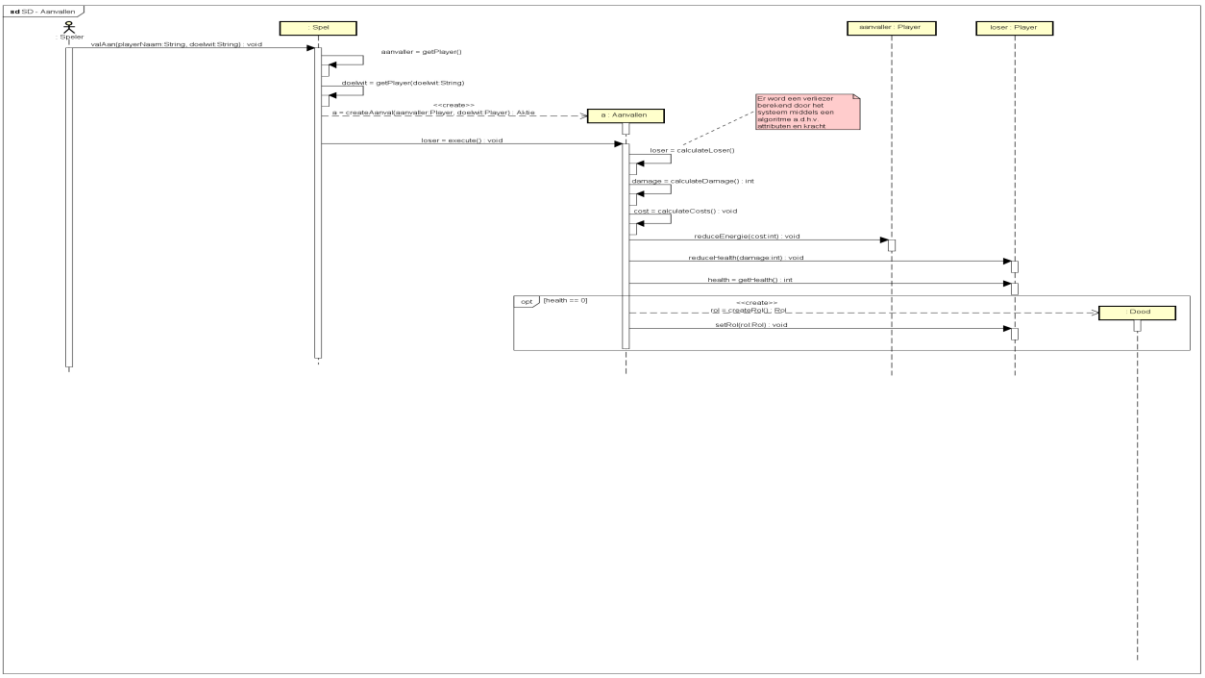
8.7. SSD Verdedigen



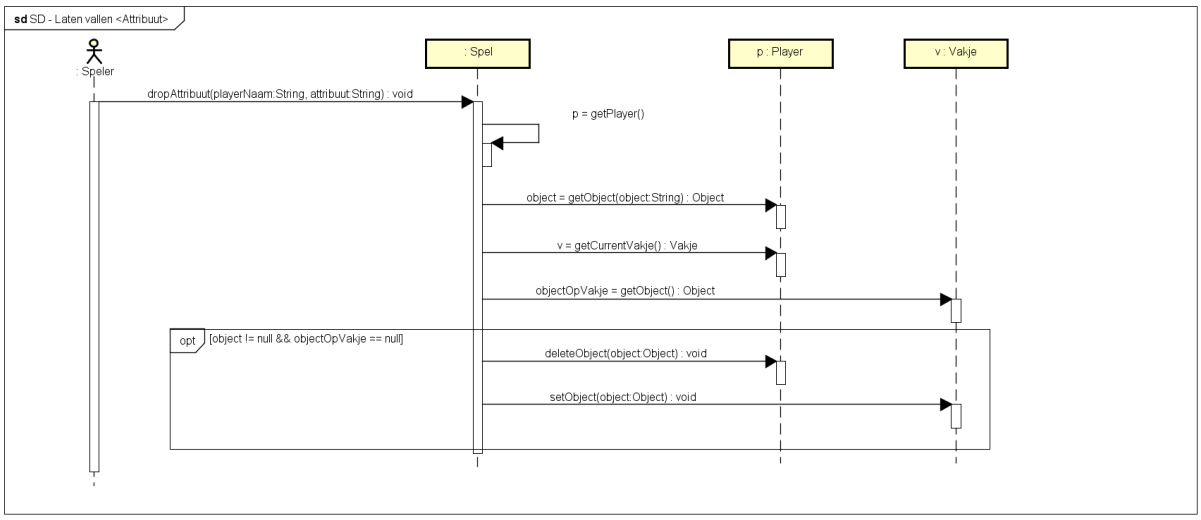
8.8. SD Aanmelden



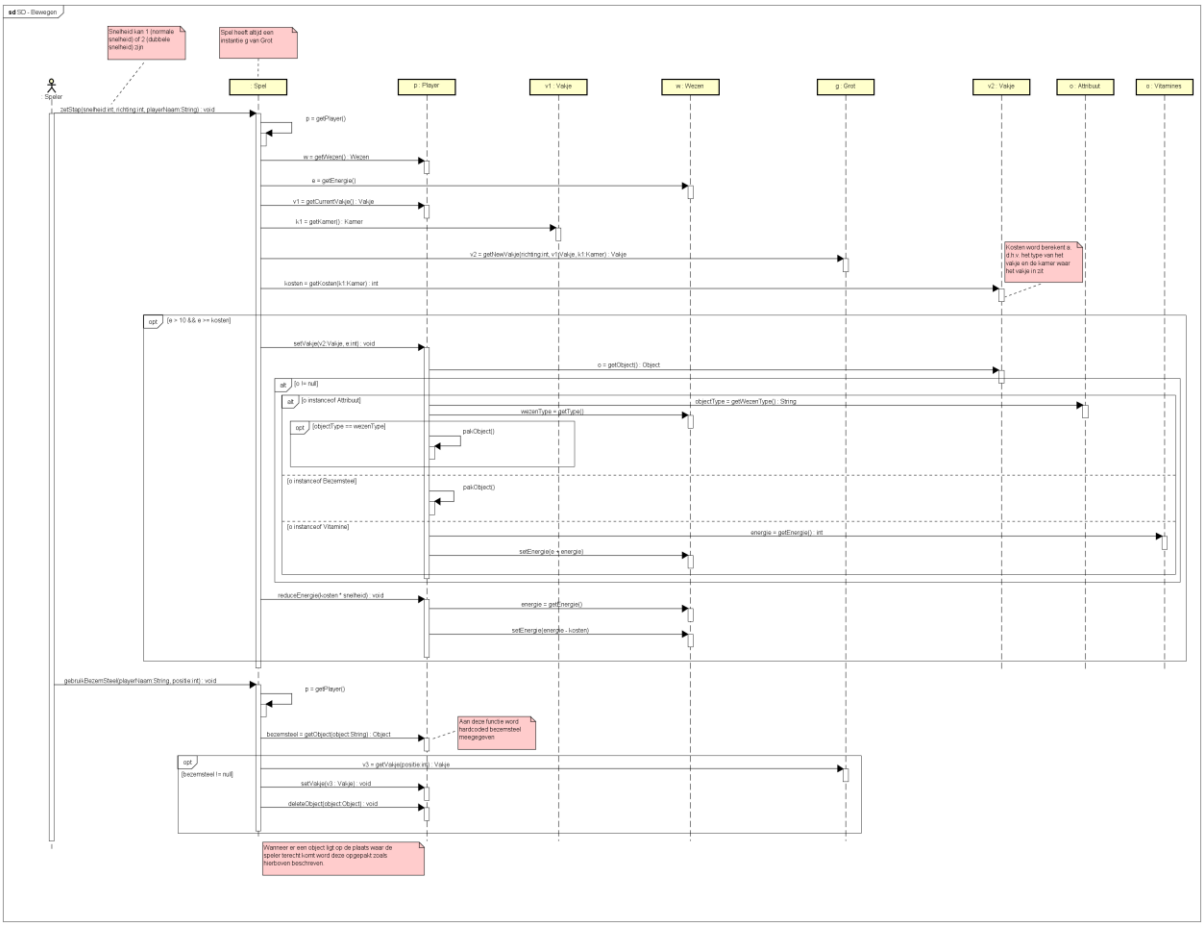
8.9. SD Aanvallen



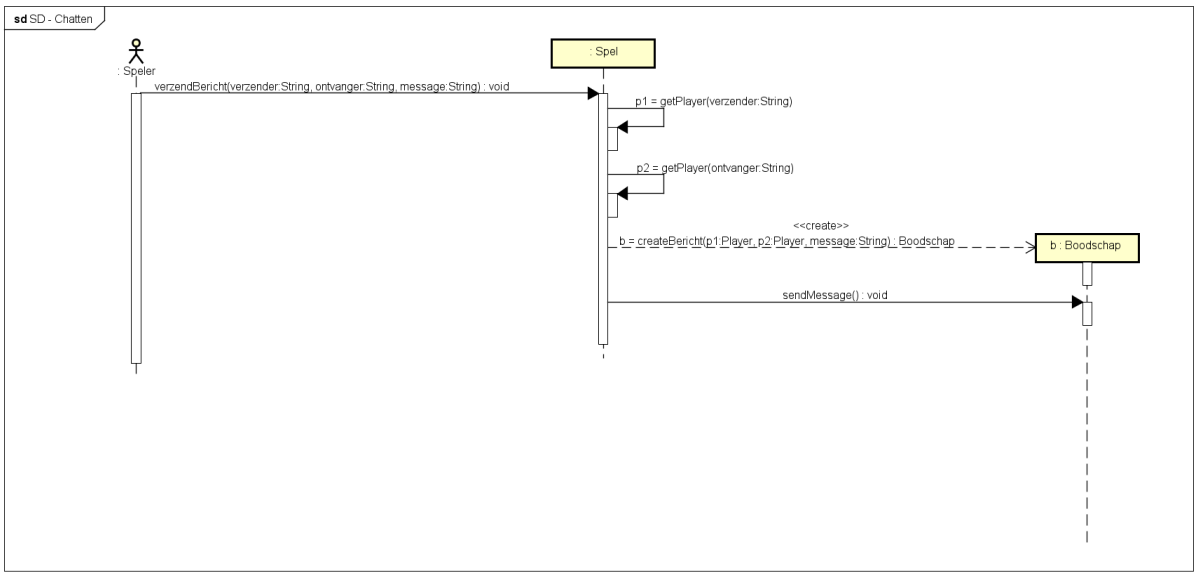
8.10. SD Laten vallen <Attribuut>



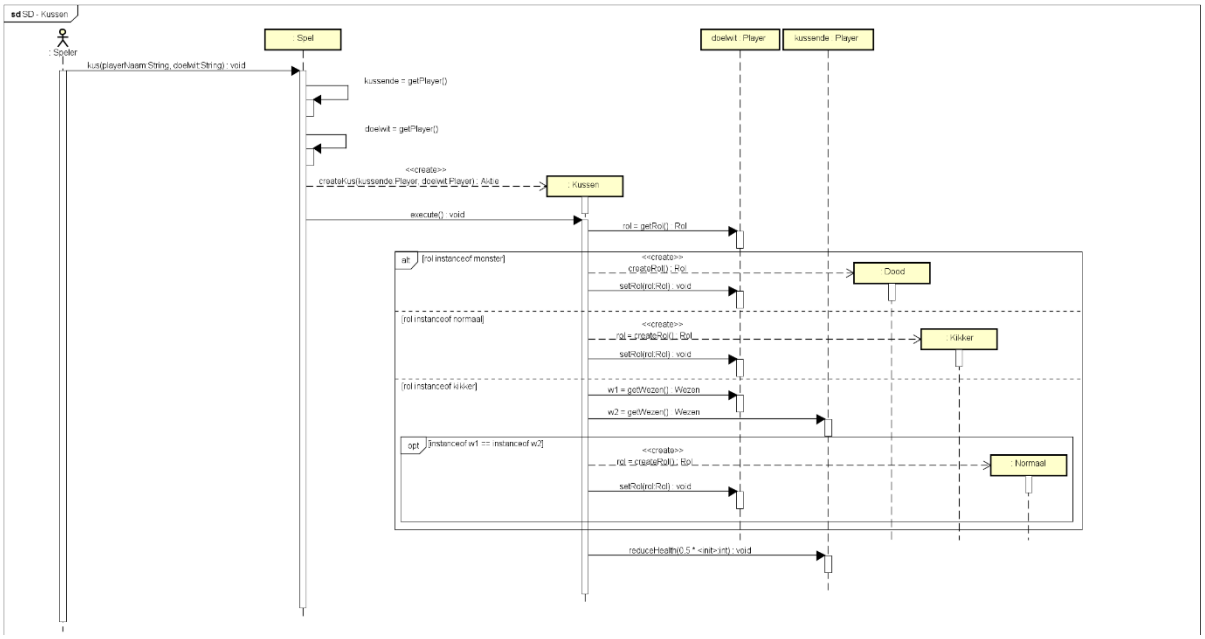
8.11. SD Bewegen



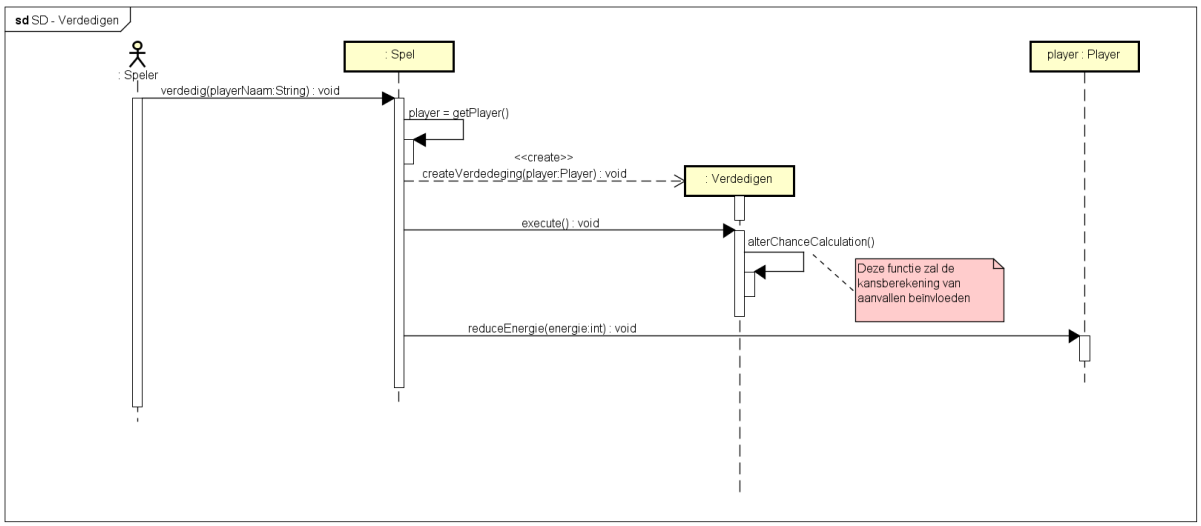
8.12. SD Chatten



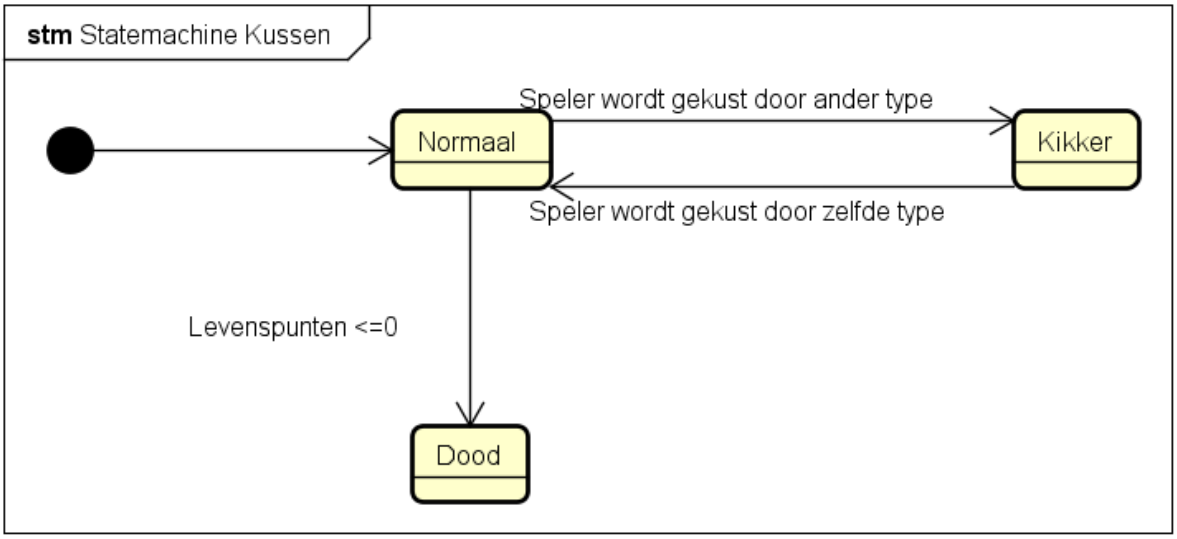
8.13. SD Kussen



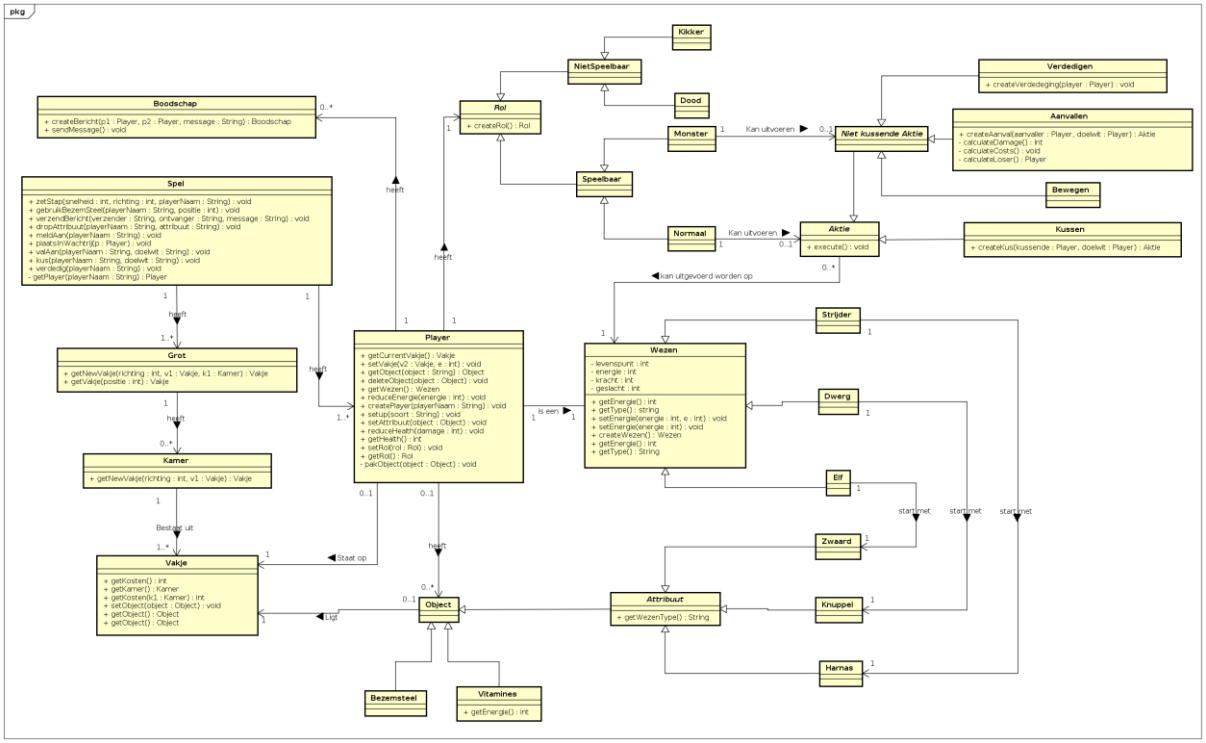
8.14. SD Verdedigen



9. State Machine Diagram



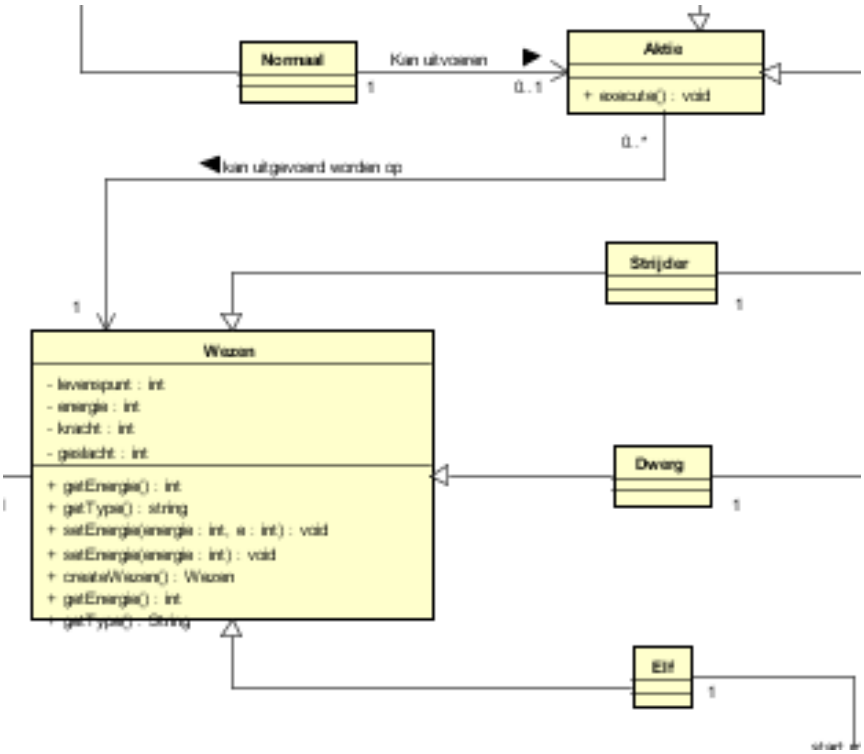
10. Design Class Diagram



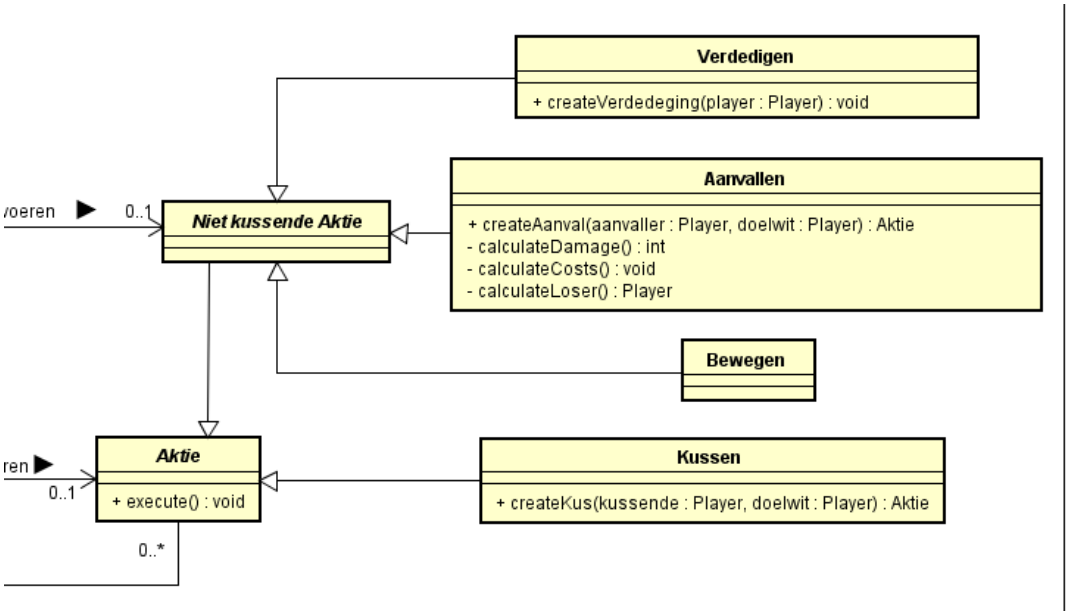
11. OO-ontwerpprincipes

Polymorfisme

Dit OO-ontwerpprincipe hebben wij onder anderen toegepast voor de klasse Wezen. Wezen kan verschillende dingen zijn namelijk, een Strijder, Dwerg of een Elf. Deze drie zijn allemaal een wezen. Dit heeft als voordeel dat er geen dubbele code wordt geschreven. Dus dan krijg je zoiets als:
Wezen w = new strijder();
w.getEnergie();
Dit is mogelijk aangezien een strijder een wezen is. Zo pakt hij altijd de goede mthode In het Design Class Diagram ziet dat er zo uit:



Abstracte klassen
Wij gebruiken een abstracte klasse Aktie. Het voordeel hiervan is dat je de klasse Aktie niet meer kan initialiseren. Aangezien je hier niks zinnigs mee kan. De subklassen hiervan kan je wel initialiseren, zoals bijvoorbeeld aanvallen. Op deze functie roep je bijvoorbeeld execute aan. Alle subklassen hebben deze methode maar elke heeft zijn eigen implementatie, en toch blijft het een Aktie. In het Design Class Diagram ziet dat er zo uit:



12. Architectuurmodel