When pigs swim

Mark Donker

Mart van den Berg



Inhoud

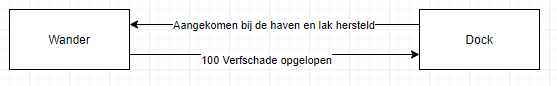
# Implementatie A\*

1. Om de afstand te meten hebben we steeds tussen elke node gekeken wat de zwaarte van de edge is en wat de kosten zijn om bij de huidige node te komen, en deze gebruikt om te berekenen hoever de afstand is naar de volgende node. Wanneer de zwaarte van de edge + de kosste om bij het startpunt van de edge minder is dan de daarvoor opgeslagen kosten naar die node, wordt de afstand vervangen met de nieuwe kostten. Ook slaan we dan op dast de route naar de bepaalde node via de node aan het begin van de edge is.
2. De gekozen heuristiek maakt gebruik van de locatie van een huidige node en de uiteindelijke locatie van de doel node, voor de afstand wordt de manhattan distance gebruikt. Na dat de afstand is berekent wordt deze gedeelt door 32, dit is namelijk de afstand in pixels tusse n2 nodes en hierdoor bereiken we een admissable heuristic. Deze afstand wordt samen met de huidige kosten naar de node gebruikt om te bepalen wat de volgende node in de queue is. Zo bekijken we niet meer nodes dan nodig.
3. ---
4. Knabbel die dwaalt wat rond van node naar node totdat hij in de straal van een varkentje komt en deze ruikt (100px). Als knabbel een varkentje ruikt dan gaat hij naar de dichtstbijzijnde node waar het varkentje zich op dat moment bevindt en gaat daar d.m.v. A\* op de snelst mogelijke manier naar toe. Als hij op dezelfde manier in de buurt van de Porcus Vincit komt dan gaat hij wat dwalen over de graaf en eet geen varkens. Wanneer Knabbel zich 100 keer heeft verplaatst gaat zij met hetzelfde algoritme en dus de snelste route terug naar haar slaapplek.

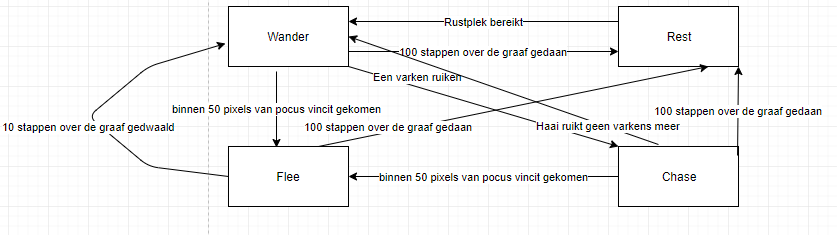
Porcus Vincit werkt op dezelfde manier als Knabbel echter dwaalt deze rond totdat zij haar maximale schadepunten heeft bereikt, daarna selecteer deze een random node waar ze zichzelf kan laten repareren en gaat op de snelst mogelijk manier hierheen.

# **Finite state machine**

1. Toestandsdiagram porcus vincit:



Toestandsdiagram Knabbel:



Transitiekansen zijn hier niet toepasselijk omdat verschillende transities prioriteit hebben over de ander, Wanneer knabbel bijvoorbeeld 100 stappen heeft gelopen, stops hij overal mee en zoekt hij zijn rustplaats. De andere transities maken dan dus ook niet meer uit.

1. Toestanden en het gedrag:

**Porcus vincit:**

* Wander:

De boot beweegt zich willekeurig over de graaf, bij elke stap wordt er één verfschade toegevoegd en wanneer een varken met de boot collide, klimt deze aan boord en is hij veilig voor de haai.

* Dock:

Bij de ingang van de dockstate wordt er random één van de drie havens gekozen, het gedrag van deze state is het bewegen dmv A\* over de graaf naar de gekozen haven, en wanneer deze haven is bereikt de verfschade weer te herstellen.

**Knabbel:**

* Wander:

Knabbel verplaatst zich naar een willekeurig aanliggende knoop op de graaf.

* Chase:

Knabbel beweegt zich dmv A\* naar de node die het dichts bij het dichtstbijzijnde varken ligt. Wanneer Knabbel met een varken botst eet hij deze op.

* Flee:

Knabbel dwaalt 10 stappen willekeurig over de graaf.

* Rest:

Knabbel beweegt zich dmv A\* naar zijn rustplek en eenmaal daar aangekomen hersteld hij zijn energie.

In elke staar behalve Rest zal knabbel ook elke stap optellen om bij te houden of dat er naar de reststate geswitched moet worden.

1. Aanpassingsstrategien voor transitiekansen:

Omdat er in deze situatie dus altijd een prioriteit is zijn er geen transisitekansen nodig.

Voor knabbel zijn de prioriteiten als volgt:

1. Rest
2. Flee
3. Chase
4. Wander

En voor porcus vincit zijn er maar twee States en is er dus altijd maar een mogelijke transitie.

# **Force-driven entities**

1. Op de varkentjes werken altijd de volgende krachten : Wander, Alignment, Cohesion, Separtion, Wall Avoidance

Op de varkentjes werken soms de volgende krachten : Flee, Seek

Bij Wander wordt er een cirkel voor het varkentje getekend en binnen deze cirkel wordt een punt gekozen waar een vector naar toe wordt getrokken, deze wordt vervolgens vermenigvuldigd met een wander radius. Daarna wordt een lokale doel bepaald om deze vervolgens te transformeren naar een doel op wereld.

Bij Alignment wordt er gekeken naar alle varkentjes die in de buurt zijn van het varkentje. Alle headings van de andere varkentjes worden bekeken en opgeteld bij de gemiddelde heading. Aan het einde wordt de heading gedeeld door het aantal varkentjes en daarna nog de eigen heading er vanaf gehaald. Dit resulteert in een kracht die het varkentje dezelfde richting geeft als het gemiddeld van de varkentjes om hem heen.

Bij Cohesion wordt er gekeken naar het middelpunt van alle varkentjes om het varkentje heen en deze wordt gebruikt om Seek aan te roepen. Hiermee krijg je een vector die naar het midden van alle varkentjes wijst.

Bij Seperation wordt er wederom naar alle varkentjes gekeken echter is het doel hier om van de varkentjes weg te sturen. De afstand tussen 2 varkentjes wordt berekend en deze wordt gedeeld door de afstand van de vector die hieruit komt. Dit wordt voor alle varkentjes bij elkaar opgeteld en leidt uiteindelijk tot een richting die van de varkentjes weg stuurt. Hoe dichterbij een varken hoe harder hij ervan weg zal sturen.

Bij Wall Avoidance worden eerst 3 Feelers gemaakt dit zijn vectoren die het varkentje gebruikt om te detecteren of er een muur dichtbij is. De eerst vector steek recht voor de richting waar het varkentje heen gaat uit, deze is ook het langste. De andere 2 liggen in een hoek van 90 graden aan de eerst vector de ene naar links de andere naar rechts. Verder zijn er virtuele niet zichtbare muren gemaakt om te bepalen waar een varkentje niet verder kan. Door middel van deze feelers gaat het varkentje kijken of deze met een muur kruist met behulp van de intersect functie. Hij kijkt voor elke feeler hoever hij over een muur gaat en onthoudt alleen de feeler die het verste over een muur gaat. Als hij over een muur gaat dan berekent hij hoever de feeler over de muur gaat en gebruikt deze kracht om te berekenen hoe hard er naar de andere kant moet worden gestuurd. De lengte van het begin van een muur minus het einde van een muur wordt genormaliseerd om een richtingsvector te krijgen. Deze wordt daar vermenigvuldigd met de lengte van de overschoot van de feelervector. Daarna wordt de perp van deze vector gebruikt om van de muur af te sturen.

Flee, een relatief simpele kracht om toe te passen. Je gebruikt het varkentje waar je de kracht van wil berekenen in combinatie met waar je van wil vluchten. Je laat de locaties van elkaar af, je normaliseert deze en vermenigvuldigt deze vector met de maximale snelheid van je varkentje. De is de wenselijke snelheid van het varkentje en dus de kracht die gebruikt wordt.

Seek, werkt precies hetzelfde als flee alleen pak je de locatie van het varkentje en haal je de locatie van datgene waar je van vlucht eraf i.p.v. andersom.

1. Omdat we bij de opdracht de sterkten van hoe zwaar de vectoren meetellen hebben gekregen hebben wij gekozen om de methode Weighted Turncated Sum toe te passen.

Hierbij wordt er gekeken naar elke kracht de er op het varkentje werkt en deze vermenigvuldigt met hoe sterk die kracht mee mag tellen. Uiteindelijk wordt de resulterende kracht getruncate, zodat de maximale kracht niet overschreden wordt.

De varkentjes hebben voor de aantrekkingskracht tot zowel Knabbel als Porcus Vincit een vermenigvuldiging tussen -1 en 1 (Flee en Seek).

Voor Cohesion, Separtion, Alignment hebben zij een kracht tussen de 0 en 1.

Wander staat standaard op 1, zodat de varkentje überhaupt met een realistische snelheid van hun plek afkomen en, zodat flocking niet overheerst waardoor erg raar bewegingsgedrag ontstaat op ten duur.

Voor wall avoidance hebben wij een hoger getal gekregen, omdat we niet willen dat een varkentje door een muur heen gaat of op een eindland terecht komt wat niet de bedoeling is, tevens zou het ook onlogisch zijn dat deze kracht minder sterk telt dan welke kracht dan ook, wie wil er nou liever tegen een muur aan lopen/rennen/zwemmen? Als een varken dus te dicht bij een muur komt kijkt zij alleen naar het ontwijken van deze muur.