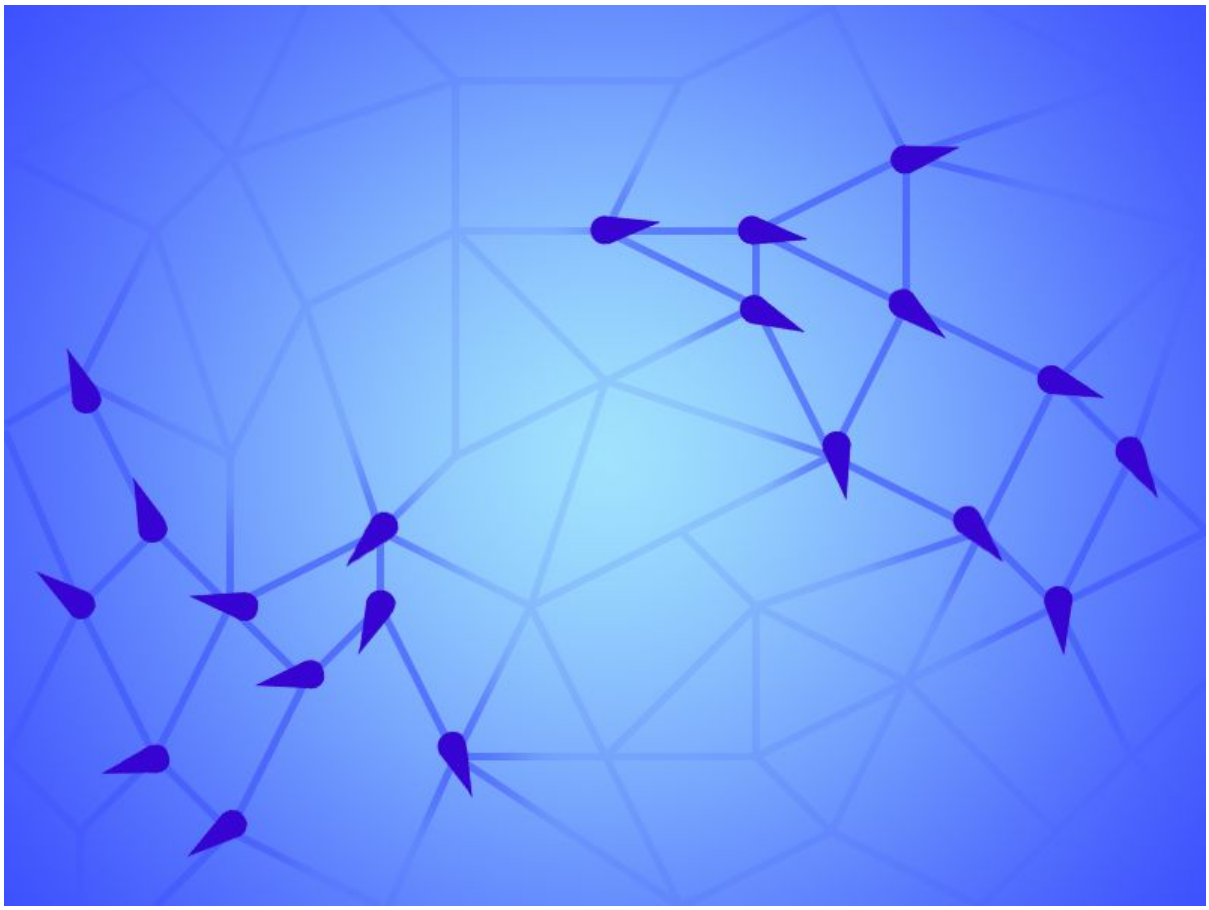


Stimmets mekanik

En undersökning av fiskars flockbildning ur ett mekanistiskt perspektiv

Naturvetenskapligt program
Gymnasiearbete
HT 2015 - VT 2016

Mattias Ulmestrand, Andi Pärn
Handledare: Andreas Logg

Abstract

The aim of this project was to study fish and find out which fundamental laws that govern their movements, and in a mechanistic approach, simulate it. The investigation was based on empirical studies as well as earlier research on the topic, such as the *Boids* algorithm provided by Craig Reynolds. The main subjects of investigation were fish of the species *Notemigonus crysoleucas*.

The investigation demonstrates that complex systems can be described by a set of relatively simple laws. To back up this claim, two organic simulations were run to show how chaotic a system can get when given a few simple rules. The algorithms were written in such a way that they were to be as close to reality as possible, thereby being as strictly scientific and natural as they could.

The results were surprisingly satisfying. Throughout the project, much was learned about how fish orientate themselves, and the simulation was a success.

Innehållsförteckning

1. Inledning

Naturliga händelser har genom alla tider lyckats häpna människor i deras komplexitet. En sådan händelse är fiskars bildning av stim. Man har försökt avkoda denna gåta genom att till och med tala om att fiskar kan ha något sorts kollektivt samvete. Naturvetenskapen ger än inget stöd för ett kollektivt samvete, och en undersökning bör till störst del förlita sig på vetenskapliga teorier.

För att förstå hur detta fenomen utspelar sig är det lämpligt att försöka simulera det. Genom att göra detta erhålls större förståelse för vilka fördelar fiskar kan ha av att vara del av ett stim.

Inom många områden av dagens samhälle kan det vara av intresse att på en utförlig nivå förstå hur fiskar rör sig. En sådan undersökning kan vara viktig för att till exempel föra spelutveckling och designteknik framåt, då moderna simuleringar kräver allt mer realism. Vissa teorier föreslår även att detta kan appliceras vid flöden, såsom tunnelbaneträngsel. Ett annat område i vilket man kan tillämpa denna typen av undersökning är automationingenjörskap. I det moderna samhället framkommer ett ökat intresse av självstyrande bilar. Sådana bilar måste ta hänsyn till sin omgivning, likt flockdjur.

1:1. Bakgrund och teori

År 1986 utvecklade och presenterade ingenjören Craig Reynolds en algoritm för flockbeteende, vilken han kallade för "Boids-algoritmen" (eller förkortat, 'Boids'). Denna algoritm är komponerad av tre relativt enkla lagar, vilka styr rörelserna av flockdjur (Reynolds, 2001). Den presenterar flockdjur som en grupp av individer, styrda av rent mekanistiska regler. I Boids appliceras reglerna på följande vis:

1. En flockmedlem styr in mot medelvärdet av närliggande flockmedlemmars positioner.
2. En flockmedlem anpassar sig efter närliggande flockmedlemmars hastigheter.
3. En flockmedlem accelererar bort ifrån för intilliggande flockmedlemmar.

När dessa lagar sammanslås, erhålls ett realistiskt sätt att förstå sig på flockbildningens mekanik. Enligt Tunström (2015), doktor i fysik och assisterande professor på Chalmers, är många algoritmer som beskriver flockbeteende varianter av Boids. Algoritmen beskriver flockdjur på en individuell nivå, men har en påverkan för hela gruppen av individer.

Medan Reynolds algoritm presenterar flockdjur på ett elegant simpelt sätt, lämnar den utrymme för modifikation för att erhålla än mer övertygande resultat.

1:1.1. Liv baserat på enkla regler

De främsta vetenskapliga teorier som bygger på flockbeteende framför ett regelsystem, med andra ord ett antal regler som styr flocken individ för individ vilket i sin tur påverkar hela flockens beteende. Detta tankesätt har använts genom historien för att skapa olika typer av simulationer.

Conway's Game of Life, ofta förkortat Game of Life, som utvecklades av John Conway på 1970-talet, simulerar liv baserat på befolkning. Ett matris bestående av kvadrater ritas upp och levande celler läggs in i dessa. Dessa levande celler utmärker sig eftersom de är ifyllda, till skillnad från döda som inte är det. Varje cell har åtta grannar som ligger i en ring runt den. Man applicerar sedan ett fåtal regler på varje cell och dess grannar:

1. Har en cell färre än två grannar eller fler än tre kommer den dö via underbefolkning alternativt överbefolkning.
2. Har död cell exakt tre grannar kommer den födas. Alternativt överlever en levande cell enbart när antalet grannar är tre.

(Conway, 2014)

Dessa regler är tämligen enkla och man skulle kunna tro att effekterna av dem är lika simpla. Detta är dock inte fallet. Faktum är att det är omöjligt att förutspå hur populationen ser ut vid en viss tidpunkt. Anledningen är att det finns oändligt många konfigurationer av själv-dödande individer som existerar i en ruta men som försvinner i nästa. I hopp om att ge en förklaring kan paralleller dras mellan Game of life och derivator. Har man en konstant i ekvationen kommer denna nollställas vid derivering och data försvinna. Ekvationen man har kvar kan omvandlas till ett oändligt antal primitiva funktioner, fast det är omöjligt att helt säkert veta vilket värde originalkonstanten före integrering hade från början eftersom en konstantterm läggs till vid integrering. Funktionen kan på detta vis, genom upprepad integrering, göras extremt komplex eller mycket simpel, likt vad som händer i Game of life. Det är därmed möjligt att få fram hur en population kan se ut för att få önskad effekt på en viss tid, fast omöjligt att veta exakt vilken uppsättning individer gav resultatet. Game of life pekar på att komplexitet finns även i det simpla, bara det finns ett flertal regler som styr.

1:1.2. Kaosteori

Kaos skapas från det lilla. Små förändringar leder till andra förändringar som i sin tur utvecklas till en grad att resultatet blir helt annorlunda än vad som var planerat. Vad Game of Life och Boids har gemensamt är deras kaotiska natur, att populationer respektive rörelsemönster förändras till synes okontrollerat, och till råga på allt extremt annorlunda vid även små förändringar. Detta fenomen kallas ofta för fjärilseffekten (Eaves, Fromhold, 2011). Det sägs att när en fjäril flaxar med vingarna skapas obalans i systemet (luften, etc.) i dess närvaro, vilket sedan stör sättet andra system fungerar och en våg av förändring sprider sig från system till system. Till slut har denna graduella förändring pågått så länge att en enorm konsekvens uppstått från fjärilens flaxande och bildat t.ex. en orkan, hypotetiskt sett. I

verkligheten finns givetvis mycket större krafter som påverkar mer på kortare tid än en fjärils lugna flaxande, men teorin stämmer in i teoretiska förhållanden.

Dessa enkla exempel visar på att det kan vara möjligt att skapa komplexa system med enbart enkla regler och att det troligtvis även är fallet baklänges.

1:1.3. Programmeringsteori

Genom denna rapports gång kommer det att demonstreras en del källkod i språket Java. Därför är viktigt att ha lite grundkoll på hur programspråket fungerar.

- **Klasser**
  Java är ett klassbaserat programmeringsspåk. En klass är en bit kod som kan köras. Flera klasser (ung. dokument) kan anslutas till varandra och användas när de behövs.
- **Variabler**
  `int` - reella heltal
  `double` - reella tal, decimaltal
  `variabeltyp namn[n]` - **fält** med många variabler vars värden hittas och sätts genom att ange variabelns plats (index) i fältet.
  Exempel: `int name[]=new int[5]; //fält med fem heltal.`
  Exempel2: `name[0]=42; //heltal 1 ges värdet 42`
  Exempel3: `int life=name[0] //life ges värdet 42`
  `objekt[n].variabel` - objektfält med ett flertal variabler som kan finnas och sättas genom indexet och variabelnamnet på en av variablerna i objektet.
  Exempel: `fish[0].r` = radien runt fisk ett.
- **Objekt**
  Syntaxexempel: `Objekt obj=new Objekt(int x, int y);`
  "Objekt // new Objekt" - separat klass med namn "Objekt" // skapar nytt objekt
  "obj" - objektets namn i den aktuella klassen.
  "int x, int y" - ett flertal variabler som paketeras i objektet.
  I programmet är det fiskar som är i fokus. Dessa kommer behandlas som så kallade objekt. Ett objekt är en klass i vilken flera olika variabler kan lagras. För att programmet skall fungera på ett korrekt sätt behöver objekten innehålla till exempel fiskarnas hastighet i x- och y-led, eller deras positioner.
- **if-satser**
  Syntax: `if (villkor){kod} else if(villkor2){kod2} else{kod3}`
  Dessa är villkorssatser. Om ett villkor uppfylls, sker en del av programmet. Till exempel, om en boll nuddar marken, studsar bollen. Annars rör den sig som vanligt.
- **for-loopar**
  Syntaxexempel: `for (int i=0;i<42;i++){kod}`
  for-loopar är strukturerade på sådant sätt att ett värde på en variabel definieras och sedan ökas eller minskas (i++, i ökas med 1) tills ett villkor inte längre gäller (i detta

fall när i=42). Under denna tid kan programmet gå igenom en bit kod ett flertal gånger
med olika indata (ofta baserade på eller helt beroende av loopvariabeln) för att räkna ut vissa saker.

- **Metoder**
  Syntaxexempel: `private int metodnamn(int n){kod}`
  "public/private" - vilken del av programmet som har tillstånd att anropa den.
  "int" - variabeltyp som returneras, "void" när flera/olika/globala variabler ändras.
  "(int n)" - variabeltyp som krävs av metoden som indata.
  Metoder används inom programmeringen för att lösa problem på ett snyggt sätt. Metoder anropas (syntax: `metodnamn(variabel)`) varvid programmet kör ett par rader kod och skickar tillbaka resultatet. I ett program kan metoder anropas varsomhelst och returnera värden. Alternativs returneras inget och allmänna (globala) variabler ändras.

1:1.4. Vattenmotstånd

Fluidmotstånd är behändigt att använda sig av när rörelser i ett flytande medie skall beskrivas. Formeln för kraften på ett objekt i rörelse lyder:

$$F_D = \tfrac{1}{2}\rho v^2 C_D A$$

F_D är den bromsande kraften,
ρ är vätskans densitet,
v är objektets fart relativt vätskan,
A är tvärsnittsarean för objektet, och
C_D är motståndskoefficienten, ett dimensionslöst tal.

(Wikipedia, 2016)

1:1.5. Local to Global

När flera partiklar interagerar med varandra finns flera olika perspektiv som används för att betrakta och förutspå rörelserna för dem (Sumpter, Mann, Perna, 2012). Dessa teorier kan även inkorporeras vid modeller för kollektiva rörelser hos djur.

När en modell av denna art betraktas är det lämpligt att använda sig av ett lokalt perspektiv, och ta reda på hur detta spelar roll för hela gruppen. Detta går även mycket väl att koppla till kaosteorin, då den omtalade fjärilseffekten grundar sig på hur en individs rörelser kan spela roll i ett större perspektiv.

1:1.6. Stim

När man talar om olika typer av stim brukar man tala om *schooling* resp. *shoaling*. Den förstnämnda gäller för mer koordinerade stim i vilka fiskarna rör sig nästan identiskt i förhållande till varandra. Den sistnämnda gäller för de flesta typerna av fiskar, speciellt för de arter som man finner t.ex i Sverige. Enligt denna beteendemodell rör sig fiskarna något självständigt, men håller fortfarande ihop (Weisheng, 2014). Hos denna gruppering syns ofta de karaktäristiska accelerationstopparna för fiskarna då de ibland gör stora utlopp av hastighet. När man betraktar ett sillstim ser man ofta hur fiskarna gör en snabb rörelse med stjärtfenan för att sedan stanna av.

1:1.7. Fiskars sinnen

Fiskar känner av rörelser och uppfattar omgivningen till stor del med hjälp av deras sidolinjeorgan samt deras ögon. Utav att deras ögon sitter på sidan av huvudet ser fiskar i ungefär 360º (Lundquist, 2012). Sidolinjeorganen, som är en typ av organ för att känna av tryckförändringar från omgivningen (Zug, 2014), gör det möjligt att ytterligare märka av om ett djur är i närheten.

1:2. Syfte

Syftet med denna undersökning är att undersöka fiskars beteenden och med hjälp av ett fåtal regler beskriva deras rörelse i stim.

1:3. Frågeställning

Vilka undersökningar måste utföras för att erhålla tillräckligt med information för att utföra undersökningen med önskat resultat? Vilka akademiker är lämpliga att konsultera med? Är det möjligt att simulera denna typen av beteende med naturligt resultat? I sådana fall, kan man med hjälp av tidigare forskning utöka teorierna om stimbildning?

2. Metod och materiel

2:1. Metod

De första studierna som utfördes var fältstudier vid Universeum. Vid ett antal observationer dokumenterades fiskarnas rörelser för att kunna beskriva deras rörelser med hjälp av simulering. Många bytesfiskar observerades här och de flesta av dem beter sig enligt liknande regler. De mest användbara filmerna som dokumenterades vid Universeum var filmer av ett stim som bestod av karpar som under influens av en stör rörde sig i cirkelformade rörelser runt stören.

Vi konsulterade med professor Tunström (2015) för att se om hans forskning i ämnet kunde bidra med relevant data. Tunström och hans kollaboratörer hade framställt en hemsida, vilken fungerade som ett stöd till vår undersökning. På hemsidan fanns en video med filmade små karpar, *Notemigonus crysoleucas* [bilaga 1]. Dessa fiskar var filmade i en 2,1 x 1,2 m tank. Djupet

var ungefär 4,5 - 5 cm, så stimbildningen var approximativt tvådimensionell. Arten bildar stim naturligt i grunt vatten, således var stimbildningen av dessa fiskar trovärdig i sin karaktär. Dessa fiskar blev till stor del referenspunken för programmet.

Den största prioriteringen som en stimfisk har tycks vara att på något sätt hålla sig inne i stimmet. Ett enkelt sätt att åstadkomma detta på är att just söka sig in mot medelpositionen av fiskar som är nära nog, såsom den första regeln i Boids. I extremfall, då endast två fiskar är i närheten av varandra, skulle detta innebära att dessa accelererar rakt mot varandra. Men först måste det tas reda på huruvida en fisk är inom området av en annan fisk. Om man ser fiskarna uppifrån kan man se det som om de vore i ett tvådimensionellt plan. Enligt en observation [bilaga 2], befinner sig alla fiskar på ungefär samma djup, och det blir således en ungefärligt tvådimensionell växelverkan.

2:1.1. Kollisionsdetektering

För att detektera om en fisk befinner sig inom en annan fisks grannskap i ett tvådimensionellt plan, kan simpel kollisionsdetektering för cirklar utnyttjas [bilaga 3]. Med hjälp av dess syn och sidolinjeorgan kan en fisk känna av rörelser inom ett begränsat område. Vi antar att detta område är cirkulärt. Detta medför enkel uträkning med realistiska förhållanden, med tanke på fiskarnas sinnen. Vad som kommer att refereras till som kollisionsdetektering är egentligen en detektion kring huruvida en fisk är inom en annan fisks område, då en “kollision” sker när en fisk är inom området.

Tänk er att man har två fiskar i ett plan. Om man ser på en av dem, har den uppgiften att detektera om den andra är inom dess område. Rent matematiskt går detta att tas reda på om man jämför radien av dess av grannskap med avståndet till den andra fisken i x- och y-led. Då erhålls ett villkor: om det totala avståndet till fisken är mindre än eller lika med radien för den primära fiskens grannskap, sker en kollision. Nedanför följer en beräkning av en kollisionsdetektering i Java:

```
public void collisionDetection(){
	double rSquare = (fish[a].r)*(fish[a].r);
		//Kvadraten av områdets radie
	double xDist;
	double yDist;
	xDist = fish[a].x - fish[b].x;
	yDist = fish[a].y - fish[b].y;
		//Avstånd i x- och y-led mellan fiskar
	double distSquared = xDist*xDist + yDist*yDist;
	if(distSquared <= rSquare){
	*kod*
	}
}
```

Exemplet visar en kollisionsdetektering, men ingenting som följer. För att applicera den mest viktiga regeln som förr talades om, krävs en utökning av koden. Denna metod kallar vi för *cohesion*, betydande sammanhållning.

2:1.2. Beteendealgoritmerna

När en fisk kommer inom grannskapet av en annan fisk skall den alltså accelerera rakt mot den andra fisken [bilaga 5]. Om den istället befinner sig inom flera fiskars grannskap kan man se det som om den accelererar mot medelvärdet av deras positioner, som en summa av alla accelerationer till närliggande fiskar. Denna acceleration kan tänkas bero på avståndet till medelpositionen av de detekterade fiskarna, då fisken har ett ökat behov av att söka sig in i stimmet om det befinner sig längre ifrån det (givet att den är inom ett område från vilket den kan upptäcka stimmet). Om avståndet till stimmet är stort, men fisken är nära nog för att detektera det, accelererar den mycket. Om avståndet istället är litet, accelererar fisken tämligen lite. Denna acceleration ökar proportionellt med avståndet mellan fiskarna.

```
public void cohesion (){
       double rSquare = (fish[0].rCentre)*(fish[0].rCentre);
       //Alla fiskar har samma radie för deras grannskap, vi väljer bara fisk 0.
       double xDist, yDist;
       for(int a = 0; a < boidNr; a++){
             //En loop som går igenom alla fiskar
            double xMean = 0, yMean = 0;
            int count = 0;
            for(int b = 0; b < boidNr; b++){
                if(a != b){ //Om inte en fisk kollas med sig själv...
                    xDist = fish[a].x - fish[b].x;
                    yDist = fish[a].y - fish[b].y;  //Avstånd mellan fiskar
                    double distSquared = xDist*xDist + yDist*yDist;
                    if(distSquared <= rSquare){
                        count++;
                        xMean += fish[b].x;
                        yMean += fish[b].y; Dessa blir till medelvärdet sedan.
                    }
                }
            }
            if(count > 0){  //Om fler än 0 fiskar är inom området...
                xMean /= count;
                yMean /= count; //Skapa medelvärden
                fish[a].dx += (xMean - fish[a].x)/300;
                fish[a].dy += (yMean - fish[a].y)/300;
                //Dividera med 300 för att skala ned till rätt storlek.
            }
        }
    }
```

En fisk som försöker bli del av ett stim söker sig dock inte bara in i stimmet, utan den anpassar även sin hastighet till de andra fiskarnas hastigheter [bilaga 6]. En enkel slutsats som kan tagas vid betraktning av ett stim är att medlemmarna i stora drag inte simmar med

särskilt skiljande hastigheter. Då är det naturligt att ännu en algoritm krävs för att få programmet att fungera. Denna metod kallar vi för *adaption*, alltså anpassning. Då en fisk har funnit stimmet har det enligt *cohesion* börjat accelerera in mot mittpunkten av det. Inriktningsalgoritmen *adaption* har lägre prioritering, således kan vi göra antagandet att den verkar över ett mindre område. Den första algoritmen, *cohesion*, ser inte till att fiskarna har samma riktning som varandra, därför behöver det problemet åtgärdas med *adaption*.

Algoritmen fungerar mycket likt *cohesion*, men istället för att vara en funktion av närliggande fiskars positioner, är den en funktion av deras hastigheter. När algoritmen körs, hittas ett medelvärde av närliggande fiskars hastigheter, och en acceleration läggs till i den riktningen. Utan denna algoritm börjar enbart fiskarna cirkulera kring varandra, utan struktur. Med dessa två kombinerade algoritmer söker sig istället fiskarna mot varandra, och anpassar sina hastigheter efter varandras.

```
public void adaption(){
        double rSquare = (fish[0].rCentre - 8)*(fish[0].rCentre - 8);
        //Mindre radie än för 'cohesion'.
        double xDist, yDist
        for(int a = 0; a < boidNr; a++){
            dxMean = 0, dyMean = 0;
            int fishCount = 0;
            for(int b = 0; b < boidNr; b++){
                if(a != b){
                    xDist = fish[a].x - fish[b].x;
                    yDist = fish[a].y - fish[b].y;
                    double distSquared = xDist*xDist + yDist*yDist;
                    if(distSquared <= rSquare){
                        fishCount++;
                        dxMean += fish[b].dx;
                        dyMean += fish[b].dy;
                    }
                }
            }
            if(fishCount > 0){
                dxMean /= fishCount;
                dyMean /= fishCount;
                fish[a].dx += dxMean/16;
                fish[a].dy += dyMean/16;
            }  //Ett medelvärde av närliggande fiskars hastigher adderas.
        }
    }
```

Åter igen krävs dock en utökning av koden, då de tidigare två algoritmerna som dikterar rörelse enbart resulterar i att fiskarna lägger sig precis intill varandra, med samma hastigheter. Det behövs en algoritm som får fiskarna att hålla sig på ett fixt avstånd ifrån varandra. Denna algoritm bör ha ännu lägre prioritering än de två ovanstående och verkar således över ett än mindre område. Vi kallar denna algoritm för *repulsion*.

För att bestämma karaktären av accelerationen som erhålls av denna algoritm gjordes ett antal undersökningar. Först undersöktes det hur två magneter repellerar varandra. Anledningen för att just magneter undersöktes är för att deras repulsion ökar kraftigt, ju mindre avståndet till magneterna är, och det är enligt liknande samband som fiskar beter sig. I långa drag kan magnetisk repulsion mellan två stavmagneter placerade ända mot ända anses vara:

$F = 1/d^2$
(Wikipedia, 2016)

$a = 1/(d^2m)$ enligt Newtons andra lag.

Om man skulle anpassa denna funktion för fiskarna skulle *d* symbolisera avståndet mellan fiskarna. Problemet med att beskriva repulsionen enligt denna funktion är att accelerationen går mot oändligheten då avståndet mellan fiskarna går mot 0. Detta skulle resultera i ett NaN-uttryck (Not a Number) för deras hastigheter, eller en annars orimligt hög hastighet. Detta vill vi absolut inte åstadkomma i programmet, så vi undviker problemet genom att göra en funktionssubstitution.

En liknande funktion, som undviker oändlighetsproblemet, erhålls genom att ta radien för repulsions-zonen och subtrahera avståndet, för att sedan kvadrera differensen. Detta medför att accelerationen stiger då fiskarna kommer närmare varandra, och detta kvadratiskt. För stora avstånd blir kvadraten inte särskilt stor, men när fiskarna kommer för nära varandra, accelererar de våldsamt, men begränsat, ifrån varandra. I karaktären liknar detta en kurva för repulsion av magneter. Funktionen ser alltså ut på följande sätt:

$a = (r - d)^2$

Nästa del av resultatet är riktningen av accelerationen. Vi utgår från hypotesen att fiskarna rör sig direkt bort ifrån varandra, alltså direkt motsatt accelerationen som erhålls av *cohesion*. På så vis kan accelerationerna ta ut varandra och fiskarna lägga sig på ett bestämt avstånd till varandra. Vi antar även att *repulsion* gäller för eventuella hot för stimmet, då fiskarna bör ta sig iväg från rovdjur på liknande sätt som de tar sig iväg från för närliggande stimmedlemmar. Fiskar är nämligen relativt primitiva varelser, och i ett utrymme såsom ett tomt, tvådimensionellt plan, finns inte många faktorer att ta hänsyn till när fiskarna ställs inför en hotande situation. Med hjälp av grundläggande trigonometri kan man dock visa att en fisk transporterar sig längre bort från hotet om det accelererar med den inversa vektorn till hotet [bilaga 4].

För att stödja detta antagande undersöktes en video som dokumenterades vid Universeum [bilaga 2]. På denna video syns tydligt att karparna i stimmet bildar cirklar kring stören. Stören är inget direkt hot, men karparna rör sig naturligt iväg från den. Cirklarna som bildas kring stören tyder på att fiskarna rör sig direkt bort från stören.

Denna funktion beter sig på så vis som en motsatt *cohesion*-metod, men med större verkan, på ett mindre avstånd [bilaga 7]. Tillsammans bildar de en sorts jämvikt, då fiskarna strävar efter att lägga sig på ett fixt avstånd till varandra. Källkod för algoritmen syns nedanför:

```
public void repulsion(){
       double xDist, yDist, rSquare = (fish[0].rAway)*(fish[0].rAway);
       for(int a = 0; a < boidNr - 1; a++){
              for(int b = a + 1; b < boidNr; b++){
                     xDist = fish[a].x - fish[b].x;
                     yDist = fish[a].y - fish[b].y;
                     double distSquared = xDist*xDist + yDist*yDist;
                     if(distSquared <= rSquare && distSquared != 0){
                         double sinA, cosA, dist = Math.sqrt(distSquared);
                         sinA = yDist/dist;
                         cosA = xDist/dist;
                         //sinus/cosinus-uttryck för vinkeln mellan fiskarna.
                         //Dessa används för att bestämma riktningen.
                         double acc = Math.pow(fish[a].rAway - dist, 2)*skillz*0.08;
                         double accX = acc * cosA, accY = acc * sinA;
                         fish[a].dx += accX;
                         fish[b].dx -= accX;
                         fish[a].dy += accY;
                         fish[b].dy -= accY;
                         //När ‘repulsion’ agerar på en fisk,
                         //agerar den motsatt för den andra.
                     }
                }
            }
        }
```

När programmet körs, uppdateras fiskarnas positioner 40 gånger/s enligt algoritmerna, och fiskarna rör sig kontinuerligt. Fiskarna rör sig då som ett mycket koordinerat fiskstim. En tänkbar utökning på deras beteende, är att få fiskarna att röra sig något mer självständigt i förhållande till varandra. Detta för att matcha deras rörelser till arten *Notemigonus crysoleucas*. Därför undersöktes videon given av Tunström [bilaga 1].

Det tycks inte finnas något klart samband om när fiskarna gör utlopp, utan det verkar ske tämligen slumpmässigt. Det skulle även kunna bero på till exempel hur stressat stimmet är och i vilken hastighet fiskar i grannskapet rör sig. För att få en ungefärlig uppskattning av hur ofta utloppen sker bestämdes att fiskarna i programmet skulle få en chans att göra ett utlopp för varje programvarv. Enligt observationerna som gjordes drogs slutsatsen att en fisk gör en spurt ungefär två gånger per sekund. För att erhålla detta resultat räknades fiskarnas utlopp under 54 sekunder, varefter summan dividerades med antalet sekunder. Kvoten som erhålls visar hur många utlopp per sekund som sker.

Ett sätt att simulera detta i programmet är att för varje programvarv ge vardera fisk en 1/20 chans att göra ett utlopp. Eftersom det är 40 programvarv per sekund, sker ett utlopp ungefär två gånger per sekund för en fisk, men slumpmässigheten gör det utspritt. Denna algoritm namnger vi *surge* (en benämning för ändring av acceleration).

Vad som sker under denna algoritms tid är att *cohesion* och *adaption* blir tillfälligt starkare under några få programvarv. Algoritmen som hanterar repulsion hålls dock konstant, då den annars blir för stark. När en fisk gör ett sådant utlopp följer den fortfarande samma beteendemönster, men det förstärks som följd av dess förhöjda hastighet.

När en fisk befinner sig i ett stim i programmet förhöjs kontinuerligt dess fart som resultat av accelerationen. Om detta tillåts utan en gräns, skulle fiskarna få en obegränsad fart, så länge som de befinner sig i ett stim. Därför behöver det tillsättas en begränsning för fiskarnas fart.

För att simulera detta på ett så korrekt sätt som möjligt är det lämpligt att välja att använda vattenmotstånd som utgångspunkt för denna retardation. Vi utgår från formeln för detta:

$$F_D = \tfrac{1}{2}\rho v^2 C_D A$$

Vid praktisk tillämpning vid kodning kan uttrycket förenklas för att erhålla smidigare uträkningar (Khan Academy, 2014). För ett objekt i en vätska, i detta fall en fisk i vatten, är alla ovanstående termer förutom farten konstanter. Dessa kan således sammanfattas till en enda variabel. Ekvationen blir då:

$$F_D = C\,v^2$$

Uttryckt i acceleration divideras båda sidor med massan, men samma resonemang kan utnyttjas här -- massan är enbart en konstant, vilken kan sammanfattas hos de andra:

$$a_D = C\,v^2$$

Källkod syns nedanför:

```
public void applyDrag(){
        double dx;
        double dy;
        for (int n = 0; n < boidNr; n++){
            dx = fish[n].dx;
            dy = fish[n].dy;
            double speedSq = dx*dx + dy*dy;
            double speed = Math.sqrt(speedSq);
            double c = 0.1;
            double sinA = dy/speed;
```

```
            double cosA = dx/speed;
            dy -= speedSq*c*sinA;
            dx -= speedSq*c*cosA;
        }
    }
```

2:2. Materiel

1. **Datorer: HP Folio 9470m**

Dessa datorer användes i största utsträckning för att skriva koden i. Datorerna stödjer Java, vilket är ett krav för att skriva och tolka programmets kod.

2. **NetBeans IDE**

Detta är ett utvecklingsprogram för olika typer av kod, men den enda som användes var Java.

3. **Kamera: Sony Alpha QX1**

För studierna av fiskar på Universeum användes denna kamera, med en bildfrekvens av 29 bildrutor/sek.

4. **GeoGebra 5.0**

För att konstruera funktionerna som finns med i algoritmerna användes GeoGebra 5.0, vilket är ett användbart program för grafritning.

5. **Adobe Photoshop CS5**

Bildmanipulering och sammanställning av data.

3. Resultat

Nedanför syns en länk till det resulterande programmet.
https://drive.google.com/file/d/0B2Ldobs8S5ivT1ROZUQ0aUFVOW8/view?usp=sharing

4. Diskussion och slutsatser

Kaosartade rörelser

För att undersöka om kaosteori även gäller för simuleringar av typen programmet utför, skrevs det till ett litet tillägg. Vid knapptryck sparas fiskarnas positioner och acceleration i x- och y-led i ett textdokument. Dessa data går sedan att sätta in i programmet om och om igen för att bilda experimentellt perfekta förhållanden. Programmet kan köras flera gånger en viss tid och positionerna sparas som bilder. Resultatet blir detsamma varje gång, eftersom alla indata är identiska. Man kan sedan redigera datafilen för att ändra en fisks position något och köra simulationen igen. Enligt kaosteori kommer denna lilla ändring ändra resultatet markant. Från en enda förskjutning på en fisk kan hela stimmets struktur förändras. Experimentet baseras alltså på en enda förändring. I verkligheten “hoppar” fiskar runt slumpartat (shoaling), vilket betyder att stimmets struktur förändras konstant på grund kaos och faktisk

rörelse. Dock är det omöjligt att få svar på om strukturen beror direkt eller indirekt på dessa förändringar, så dessa måstre reduceras till en enda förändring [bilaga 8-9].

Slumpartade rörelser
Genom att ge fiskarna förmågan att begå snabba hastighetsökningar erhölls en mycket mer realistisk simulation. Genom att göra dessa rörelser kan det även leda till att stimmet som de befinner sig i byter riktning på ett oförutsägbart sätt, så som i Tunströms video. Detta har sin grund i att fiskarna vill följa varandra, och om en av de gör ett plötsligt utlopp kommer rörelsen att fortplanta sig i närliggande fiskars rörelser. Ett plötsligt utlopp av hastighet kan leda till att en fisk 'översvänger', det vill säga att den faktiskt hamnar en bit bort från den position som den ursprungligen siktade på att hamna vid. Då störs stimmets bana något, och dess riktning blir inte lika förutsägbar. Måhända är detta del av varför fiskarna beter sig så som de gör i videon.

Altruistiskt beteende
Simuleringen är ett exempel på altruistiskt beteende, vilket är en typ av beteende som gynnar fler än bara en individ. Detta visar sig i programmet genom att fiskarna hjälper varandra, då då de alla anpassar sig efter varandras hastigheter och positioner. Om en fisk får syn på ett hot och därav accelererar bort från det, fortplantar sig denna rörelse i hela stimmet likt en kedjereaktion. Detta beteende gynnar artens överlevnad, då det ökar stimmets förmåga att överleva. Utav detta kan fler fiskar få avkomma. En enstaka fisks signifikans kan ytterligare motiveras av fjärilseffekten som demonstreras av fiskarnas 'kaosartade rörelser', och visar på att om en ytterst liten förändring sker, påverkar det alla individer, och ofta för det bättre.

Tillägg till beteendealgoritmerna
Givetvis kan extremt många tillägg göras till algoritmerna. De som ansågs vara viktigast inkluderades i programmet, men det finns även fler faktorer som möjligen kan tilläggas, av vilka en är att fiskarna kan få syn på mat. Detta är ju en annan mycket stor prioritering för fiskarnas överlevnad, och skulle troligen mycket väl påverka hur realistiskt programmet framstår.

Utöver de faktorer som finns med i programmet kan man även räkna med att vattnet strömmar. Detta bör påverka hur fiskarna reagerar till omgivningen. Till exempel kanske de blir mer stressade utav den rörliga miljön, då detta känns av bland annat vid deras sidolinjer.

Flöden och bilar
Att bygga en självstyrande bil är inte så svårt. Visst behöver man en radar, diverse andra mätinstrument samt en styrenhet för att få allt att fungera, men efter det har man i princip en bil som kan styra sig själv. Den kan åka framåt, svänga och kanske till och med väja för hinder. Det är inte längre det stora problemet med självstyrande bilar. Problemet står i hur de styr i trafik. En vanlig bil har en förare som känner av omgivningen, ser till att styra undan när andra bilar kommer in från en infart och så vidare. En dator kan inte hantera praktiskt

tänkande, utan måste läras vad den skall göra i vissa situationer. Tänk nu istället att datorn utrustas med flockinstinkt. Den kan se inkommande bilar som hot och väja undan samtidigt som den "puttar" på andra självstyrande bilar på samma altruistiska sätt som i en flock under hot. Bilen kan följa efter andra bilar på säkert avstånd likt *cohesion* och bestämma svängningar med hänsyn till andra bilar, precis på samma sätt som *adaption*.

I en flock är målet att hålla sig borta från fara genom att hålla gruppen kompakt samtidigt som ingen individ utsätts för fara av de andra i flocken. På detta sätt skulle en motorväg kunna fyllas till bredden med bilar som färdas snabbt och säkert med hjälp av kontakt med hela "flocken".

Källförteckning

Conway, John, *Inventing Game of Life - Numberphile*, 2014-03-05 [https://www.youtube.com/watch?v=R9Plq-D1gEk] Hämtat 2016-01-25

Eaves Laurence, Fromhold Mark, *Chaos and Butterfly Effect - Sixty Symbols*, 2011-07-25 [https://www.youtube.com/watch?v=WepOorvo2I4] Hämtat 2016-01-05

Lundquist, Anders, *Info om djur: Fråga en zoofysiolog*, 2012 [http://www.djur.cob.lu.se/svar/Sinne.html#ogon-synfalt-framat-at-sidan-stereosyn] Hämtat 2016-01-04

pamela (Khan Team), *Air and fluid resistance*, 2014 [https://www.khanacademy.org/computing/computer-programming/programming-natural-simulations/programming-forces/a/air-and-fluid-resistance] Hämtat 2016-01-03

Reynolds, Craig, *Boids (Flocks Herds and schools: a Distributed Behavioral Model)*, 2001 [http://www.red3d.com/cwr/boids/] Hämtat 2015-12-15

Sumpter David, Mann Richard, Perna Andrea, *The modelling cycle for collective animal behaviour*, 2012-07-15 [https://drive.google.com/drive/folders/0B2Ldobs8S5ivdXBqZTU0ZWN0ekk] - Länk nr 1 Hämtat 2015-12-14

Tunström, Kolbjörn, 2015, [http://hieraticdemonstrator.net/] Hämtat 2015-12-14

Weisheng, Lin, *Difference Between Shoaling & Schooling*, 2014, [http://www.aquaticstory.com/our-articles/general-topics/difference-between-shoaling-schooling] Hämtat 2016-02-08

Wikipedia, *Drag (physics)*, 2016-01-02 [https://en.wikipedia.org/wiki/Drag_(physics)] Hämtat 2016-01-15

Wikipedia, *Magnet*, 2016-01-11 [https://en.wikipedia.org/wiki/Magnet#Force_between_two_bar_magnets] Hämtat 2016-01-12

Wikipedia, *Sidolinjeorganet*, 2016-12-19 [https://sv.wikipedia.org/wiki/Sidolinjeorganet] Hämtat 2016-01-22

Zug, George R, *lateral line system*, 2014-04-03 [http://www.britannica.com/science/lateral-line-system] Hämtat 2016-02-08

Källkritik

Craig Reynolds användes som en källa när det skrevs om algoritmen Boids i bakgrunden. Eftersom det är Reynolds själv som har skapat algoritmen är han troligen en säker källa att använda sig av. Han är en utbildad grafikexpert och ingenjör och har vunnit priser för sina arbeten angående flockbeteende bland annat. Reynolds algoritmer är väldigt vedertagna och många liknande algoritmer byggs upp som varianter av Boids. Det finns inte mycket anledning till att tvivla på Reynolds expertis inom området då han är väldigt erfaren i området. Däremot kan det tvistas kring huruvida Reynolds släpper alla sina ‘hemligheter’ kring sin algoritm för allmänheten. Det lästes inte någon väldigt utförlig uppsats av Reynolds, utan bara en grundlig genomgång om hur algoritmen är uppbyggd. Eftersom vår produkt överensstämmer relativt väl med Boids-algoritmens regler, samt har gett ett önskvärt resultat, finns ytterligare mindre anledning att betvivla Reynolds.

Vad gäller videorna om kaosteori och Game of Life, är källorna optimala. Videon om kaosteori presenteras av två experter på området (professorer på Nottinghams universitet) och bör vara trovärdiga av just denna anledning. Dessutom är kaosteori mycket utvecklad som

vetenskaplig inriktning och dess effekter är beprövade. Flärilseffekten är mer eller mindre ett officiellt förklaringssätt inom kaosteorin. Videon om Game of Life presenteras av John Conway själv, skapare av GOL. Någon säkrare källa finns knappast i detta område. Båda källor är dock översiktliga, men ger tillräckligt med information för att kunna beskrivas i ett arbete som detta.

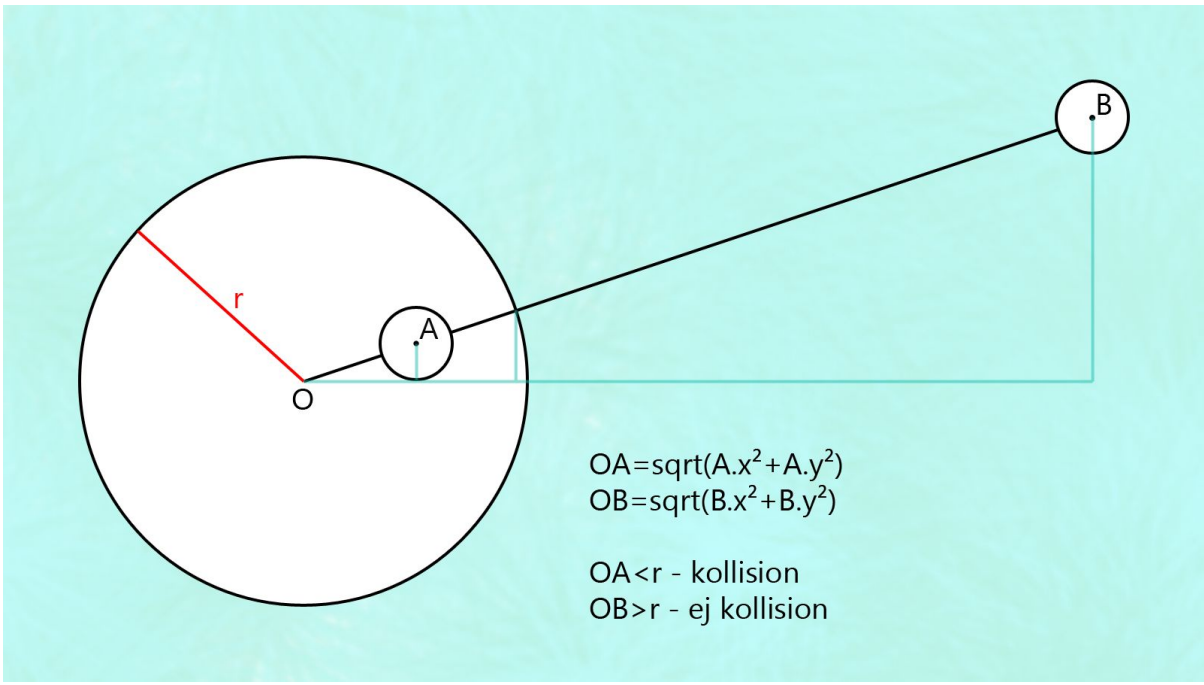
När fluidmotstånd togs upp användes *pamela* från Khan Team vid Khan Academy som källa. Khan Academy är en erkänd sida att lära sig om vetenskap på, men de individuella lärarna är svåra att säga mycket om. Vid studering av pamelas profil står det inte mycket om henne som person och det är inte möjligt att lätt få reda på hennes identitet. Således är det omöjligt att veta vad för typ av utbildning som läraren har genomgått och svårt att säga om källan är trovärdig enbart utav detta. Dock stämmer källans information väl överens med andra källor som berör samma ämne.

Bilagor

Bilaga 1:
http://hieraticdemonstrator.net/

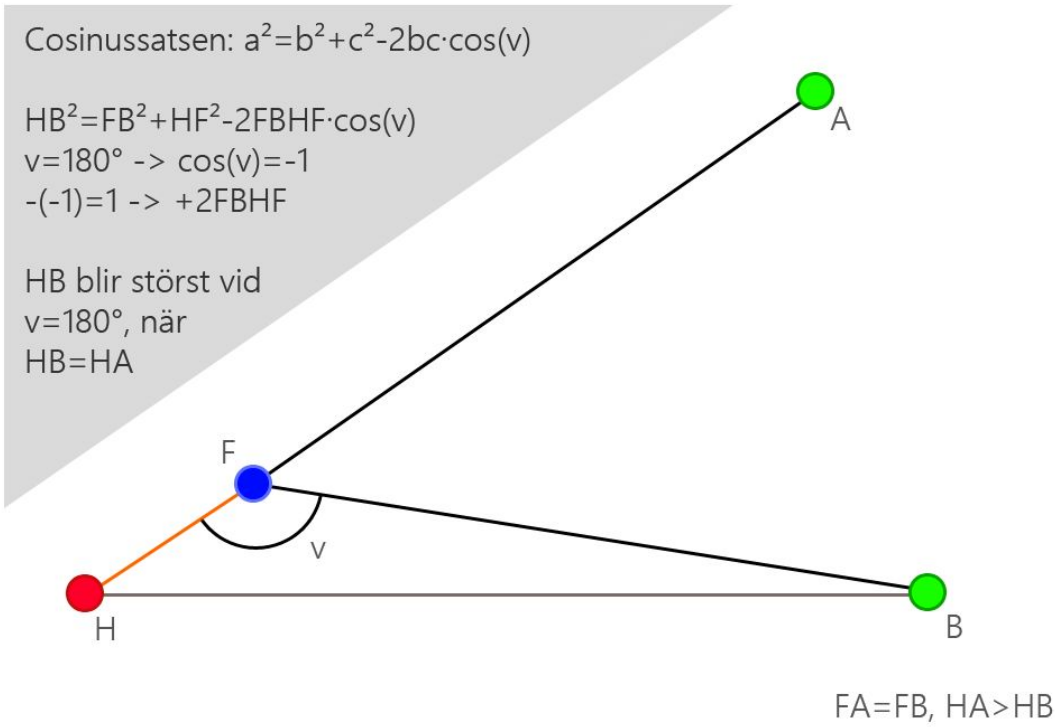
Bilaga 2:
https://drive.google.com/folderview?id=0B2Ldobs8S5ivY2RBdkxFZERlN00&usp=sharing

Bilaga 3:



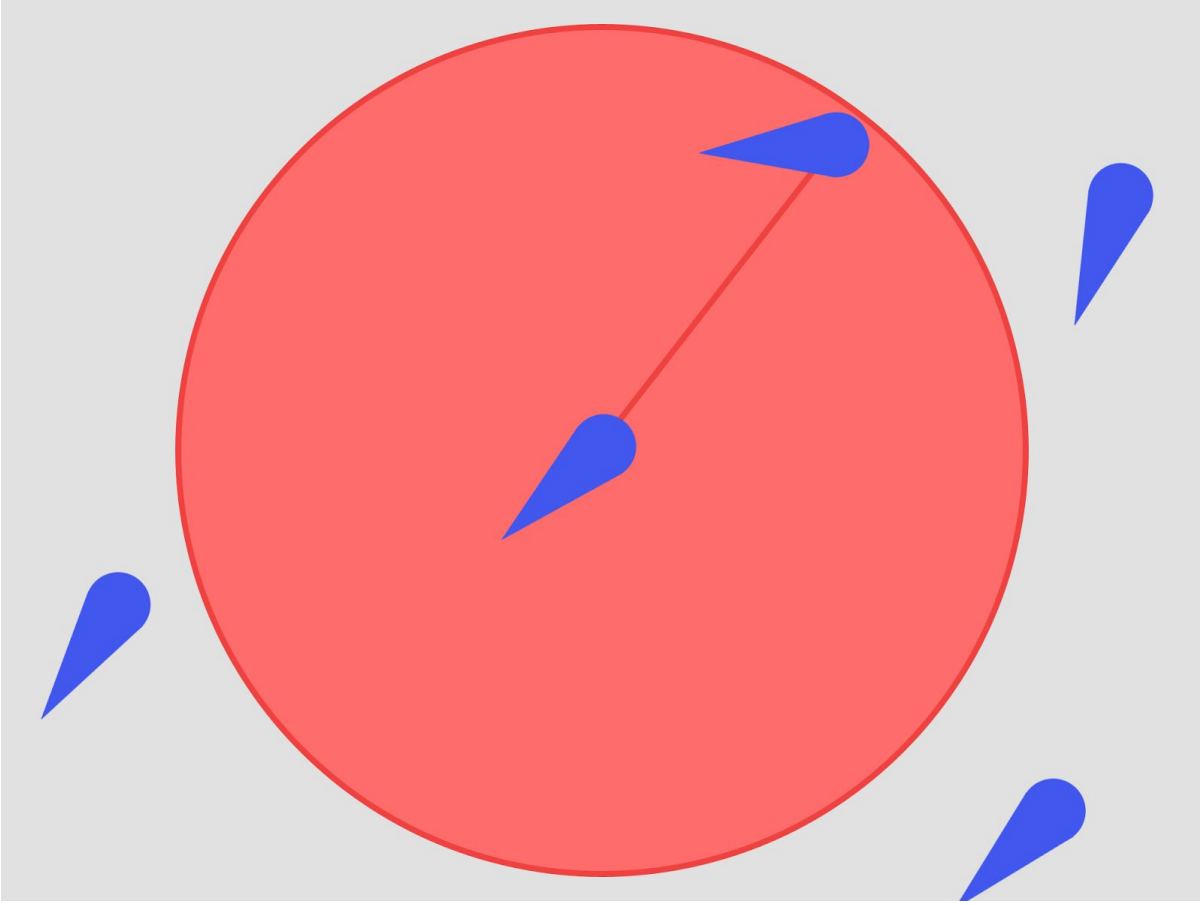
Beräkning av en annan fisks position i jämförelse med aktiva detektionsområden för cohesion, adaption och repulsion.

Bilaga 4:



Optimal väg ifrån hot. Genom att simma åt motsatt riktning kommer fisken så långt bort från hotet som möjligt på kortast möjliga tid.

Bilaga 5:

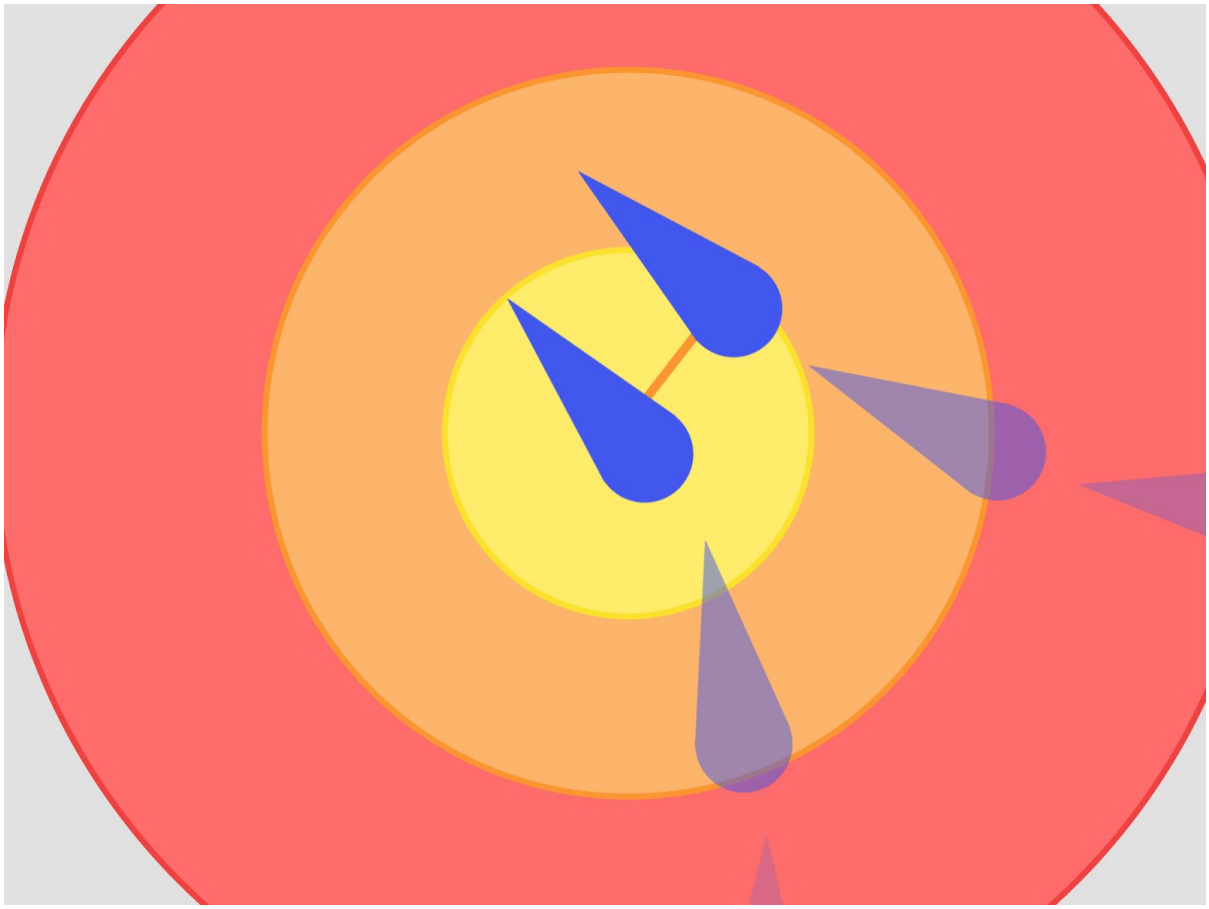


Cohesion: Om en fisk kommer inom en viss radie börjar de styra mot varandra.

Bilaga 6:

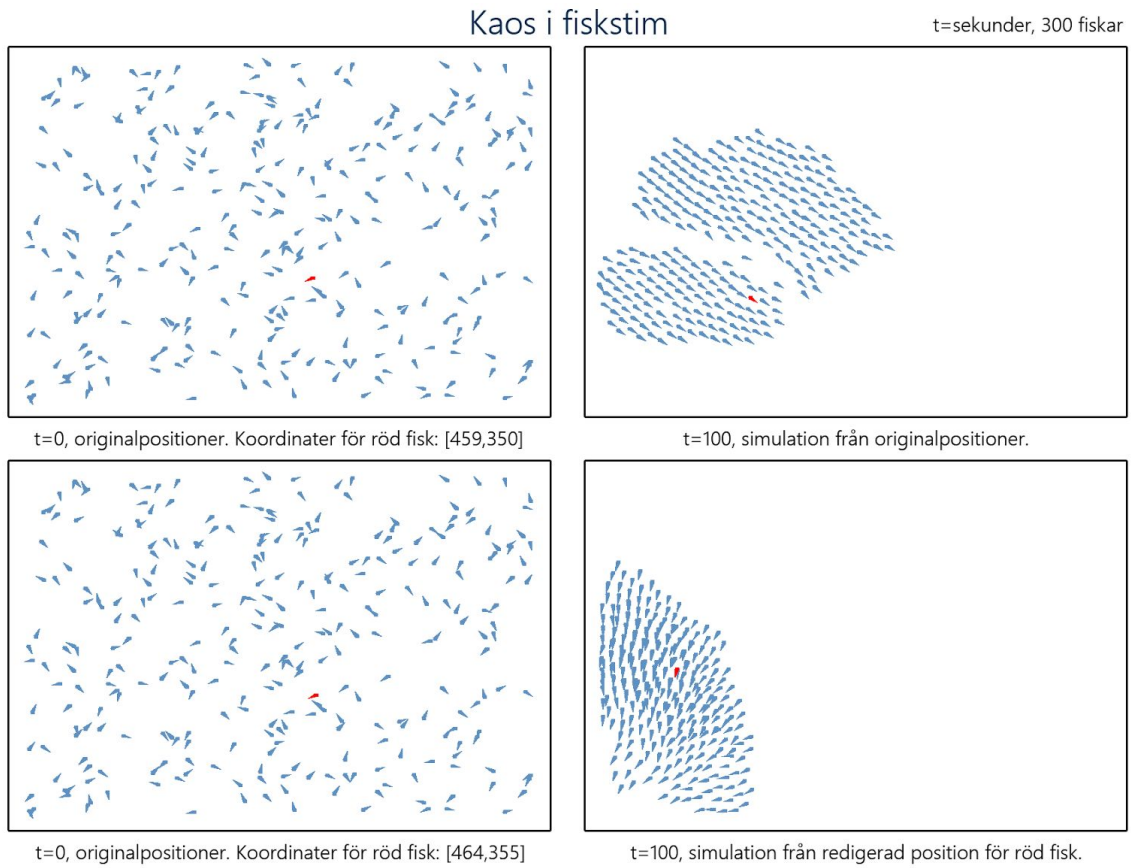
Adaption: De två fiskarna anpassar sina hastigheter och simvinklar till varandra.

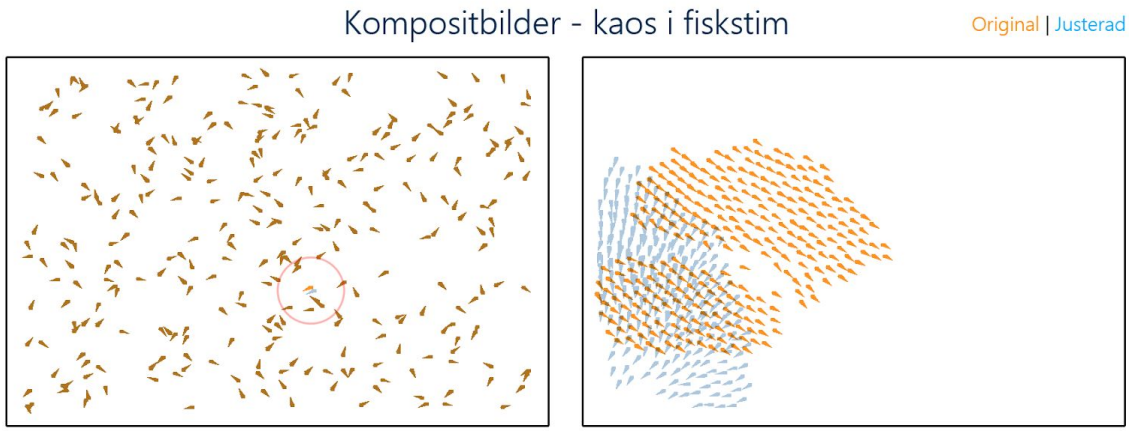
Bilaga 7:



Repulsion: fiskarnas proximitet leder till en fara och de separeras till en säkrare längd mellan dem.

Bilaga 8:





Två experiment och dess utgångs- och slutpositioner överlagda för att framhäva skillnader.