Användningen av Genetiska Algoritmer i Spel

Stefan Virtanen, 34209

Kandidatavhandling i datateknik

Åbo Akademi

Institutionen för Informationsteknologi

Referat

Evolutionära algoritmer är ett samlingsnamn för algoritmer som baserar sig på idéer från biologin om evolution för att nå en optimal lösning på ett problem, en av dess delgrupper är genetiska algoritmer. Denna avhandling är en kort introduktion till genetiska algoritmer och hur de kan och har applicerats för procedurell innehållsgenerering i spel. Avhandlingen kommer ta genetiska algoritmer i fokus och presentera användningsområden i och spelets innehåll.

Nyckelord: Evolutionära algoritmer, genetiska algoritmer, procedurell innehållsgenerering

Innehållsförteckning

Referat....2
1. Inledning....4
2. Evolutionära Algoritmer....5
 2.1 Introduktion....5
  2.1.1 Kort historia....6
 2.2 Typer av evolutionära algoritmer....6
2. Genetiska Algoritmer....7
 2. 1 Introduktion....7
 2.2 Representation av individer....7
  2.2.1 Binärtalsrepresentation....8
  2.2.2 Heltalsrepresentation 8
  2.2.3 Övriga typer av representation....8
  2.2.3 Lämplighetsberäkning....9
  2.2.4 Val av individer....9
  2.2.5 Mutation....10
  2.2.6 Rekombination....10
 2.3 För- och nackdelar....11
 2.4 Användning i spel....13
3. Procedurell innehållsgenerering....13
 3.1 Procedurell generering av banor med genetiska algoritmer....14
 3.2 Procedurell generering av grafik med genetiska algoritmer....16
 3.3 Procedurell generering av musik med genetiska algoritmer....17
5. Avslutning....18

1. Inledning

I dagsläget ökar spelindustrin kraftigt och förutsägs till 2014 uppnå $86 miljarder globalt.[18] De kvalitetskrav som i dagsläget ställs på så kallade 'AAA' (tripple-A) titlar har lett till en kraftigt ökad kostnad att producera spel som anses dugliga, eller ”bra nog”. Exempelvis uppskattas det nya MMORPG-spelet Star Wars: The Old republic ha kostat §200 miljoner att utveckla.[7]

Dessa kostnader, samt en önskan om möjligheten att göra ändringar i spelet smidigare, har lett till att industrin hela tiden söker efter kostnadseffektivare sätt att skapa innehåll i spelen,[5] exempelvis algoritmer för att generera banor och terräng istället för att ha anställda att för hand skapa hela spelvärlden. Ett annat problemområde inom spelindustrin är utvecklingen av artificiell intelligens för karaktärerna i spel. Just dessa två problemområden har jag valt att beröra i det här arbetet.

Genetiska algoritmer, en delgrupp till evolutionära algoritmer är algoritmer som tar idéer från evolutionen och naturligt urval. Algoritmerna tar främst idén om arv genom parning och mutation från evolutionsteorin och applicerar dessa för exempelvis funktionsoptimering[1] eller för att komma fram till nya lösningar på problem[13]. Genetiska algoritmer är en av de lösningar som prövats och fortsättningsvis används inom spelindustrin för att skapa innehåll kostnadseffektivare.

Genetiska algoritmer har den egenheten att de i teorin går att applicera på alla problem, problemet är hur man ska specificera algoritmen och sätta krav på den på ett sådant sätt att den producerar användbara resultat.

2. Evolutionära Algoritmer

2.1 Introduktion

Evolutionära algoritmer (EA) är algoritmer som baserar sig på idéer från biologin[1], där den underliggande och gemensamma idén är att given en mängd individer och begränsade resurser, tävling mellan individerna orsakar naturligt urval som i sin tur orsakar att den allmänna lämpligheten hos individerna. Lämpligheten bestäms av en funktion som beräknar ett värde som representerar hur bra individen passar för det önskade resultatet. Efter detta låts individerna ändra genom variation, exempelvis mutera och rekombinera tills ett önskat resultat nås. Rekombination kan enkelt beskrivas som en parning av två eller flera individer (”föräldrar”) för att skapa nya individer (barn). Mutation är processen att välja en individ och ändra den till en annan, ny individ. Efter detta väljs de bästa individerna ut för nästa generation.

En typisk EA fungerar som illustrerat i Fig. 1

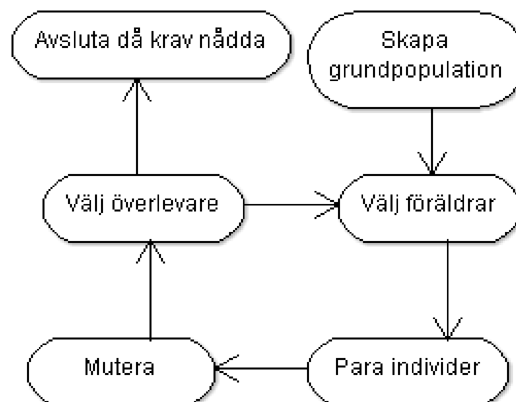


Fig 1: Enkel beskrivning av hur EA i grunden fungerar

Det finns några delar och funktioner i EA som är speciellt viktiga och krävs för att skapa en komplett EA. [1]

- Representation av individerna
- Evalueringsfunktion (lämplighetsfunktion)
- Population
- Funktionen för att välja föräldrar
- Variationsfunktioner
- Val av överlevare

2.1.1 Kort historia

Idén med att basera problemlösning på Darwinism daterar ända bak till 1948 till Alan Turing då han föreslog sökning genom evolution[1]. Dock var det först 1962 som Bremermann utförde optimering med hjälp av evolution. Under 60-talet skapades tre olika implementationer i USA.

Lawrence J. Fogel, Alvin J. Owens och Michael J. Walsh introducerade paradigmen evolutionär programmering, John Holland uppfann genetiska algoritmer och i Tyskland skapades Evolutionsstrategier av Ingo Rechenberg och Hans-Paul Schwefel. Även genetisk programmering introducerades under 60-talet, men har fram till 90-talet använts endast för att lösa relativt enkla problem på grund av hur beräkningstunga de är. Sedan början av 90-talet har det dock skett stora framsteg inom genetisk programmering, bland annat John R. Kozas flera verk inom ämnet.

Dessa var från början helt skilda tillvägagångssätt, men har sedan 90-talet behandlats som skilda representationer av samma teknologi, nämligen evolutionär beräkning. I dagsläget betecknas fältet som evolutionära beräkningar, medan algoritmerna som används betecknas som evolutionära algoritmer. Evolutionära algoritmer delas vidare in i undergrupperna evolutionär programmering, genetiska algoritmer, evolutionsstrategier och genetisk programmering.

2.2 Typer av evolutionära algoritmer

Evolutionära algoritmer har historiskt delats in i flera delgrupper, även om alla delgrupperna är väldigt lika varandra. Delgrupperna genetiska algoritmer (GA), evolutionära strategier (ES), evolutionär programmering (EP) och genetisk programmering (GP). Medan det i praktiken blir stora skillnader på hur dessa algoritmer implementeras och hanteras är i teorin den enda egentliga skillnaden mellan dem hur de representerar data.

Generellt kan man säga att genetiska algoritmer representerar data som finita strängar bestående av ett finit alfabet. Evolutionära strategier består av realvärdes vektorer, evolutionär programmering representeras av finita tillståndsautomater och genetisk programmering består av dataträd.

Medan det algoritmerna historiskt har haft de här namnet tycks kutymen inte vara helt fastslagen och namnen evolutionära algoritmer eller evolutionär beräkning används istället för en av delgrupps namnen.

2. Genetiska Algoritmer

2. 1 Introduktion

Genetiska algoritmer (GA) är, precis som andra evolutionära algoritmer, inspirerade av evolution och hur successiva generationer ändrar för att nå en optimal lösning för en given omgivning.[2] GA används generellt för funktionsoptimering, men kan även användas för att simulera adaptivt beteende.

Det kan dock diskuteras huruvida genetiska algoritmer egentligen lämpar sig för funktionsoptimering då de snabbt kan hitta ett område där ett globalt eller lokalt optimalt värde befinner sig, men kommer att ha avsevärt större problem med att hitta det egentligen optimala värdet. En genetisk algoritm kan dock enkelt modifieras för att användas som funktionsoptimerare, dock skulle den ha stora skillnader från en typisk genetisk algoritm. [11]

En individ i populationen i en genetisk algoritm representerar ofta flera datapunkter, exempelvis en räcka av heltal. Summan av dessa datapunkter kallas för en individs genotyp, medan denna genotyps resultat vid användning kallas för fenotyp.

En genetisk algoritm kan kort sägas fungera som följande pseudokod:

```
initialisePopulation()
while(!terminationCondition)
        evaluateFitnees()
        selectFittest()
        recombinePopulation()
        mutatePopulation()
```

2.2 Representation av individer

I genetiska algoritmer kan individer representeras på många olika sätt, men binärtals-, heltals- och flyttalsrepresentation är vanligast eftersom genetiska algoritmer ofta används för funktionsoptimering.

Det kan vara mycket värt att vid implementeringen av en genetisk algoritm fundera hur individerna ska representeras för att göra det lättast möjligt att implementera och kontrollera. Det är också viktigt att ta i beaktande om individen ska representeras med en varierbar längd eller med en fast längd.

2.2.1 Binärtalsrepresentation

Binärtalsrepresentation av individerna har historiskt varit den vanligast typen av representation, och har ofta används oberoende av problemet som ska lösas.[2] Medan binärtalsrepresentation anses vara det generellt enklaste sätet att representera lösningar på medföljer tyvärr en del problem, det störast av dem är Hammingavståndet mellan två varandra nära värden. Exempelvis de binära värdena 1000 och 0111 är varandra väldigt nära, dock är Hammingavståndet jämförelsevis stort.

Ett Hammingavstånd, uppkallat efter Richard Hamming, är inom informationsteori ett mått på avståndet mellan två lika långa strängar av tecken där avståndet är högre desto flera tecken skiljer.

Problemet med Hammingavståndet kan kringås genom att implementera ett annat system för tolkning av binära tal, nämligen Graykod, även känd som reflekterad binärkod. Reflekterad binärkod utmärker sig att två intilliggande heltal alltid har Hammingavståndet 1 till varandra.

2.2.2 Heltalsrepresentation

I vissa fall kan det vara lämpligt att representera individer med hjälp av heltal istället för binärtal. Detta kan exempelvis vara lämpligt då man ska representera en stig på ett fält med en sträng av variabel länd. Strängen kan då exempelvis vara uppbyggd av talen 1,2,3 och 4 för att representera norr, väst, öst och söder på fältet. Medan detta självfallet är gångbart även i binärtalsrepresentation är det aningen smidigare och mera överskådligt att representera det i heltal.

2.2.3 Övriga typer av representation

Individer kan representeras på många andra sätt helt beroende på vad som lämpar sig bäst och det finns inga egentliga begränsningar på hur många olika värden som representeras i en individ. Eftersom genetiska algoritmer generellt används för funktionsoptimering kan det vara klokt att representera individen med flyttal. Tyvärr är precisionen hos flyttal begränsad av sin implementation.

Individer i genetiska algoritmer kan även representeras som trädstrukturer eller neurala nätverk. Om individerna representeras av ett neuralt nätverk kallas det för neuroevolution. Exempel på en genetisk algoritm för neuroevolution är NeuroEvolution of Augmenting Topologies (NEAT) av Stanley et al. [12]

Genetiska algoritmer lämpar sig även för att lösa permutationsproblem, exempelvis

handelsresandeproblemet. I sådana fall kan individen representeras av tex. en permutation av heltal som representerar de olika möjligheterna.

2.2.3 Lämplighetsberäkning

Efter att ha skapat en startpopulation, eller efter varje omgång av parning och mutation beräknas varje individs lämplighetsvärde. Värdet representerar hur bra individen uppfyller problemets krav.[8]

2.2.4 Val av individer

När lämplighetsvärdet för varje individ är känt väljer man ut individer att skapa nästa population med. Valet av individer är deterministiskt eller stokastiskt, där deterministiskt innebär att den bättre individen av två alltid väljs, medan stokastisk innebär att den bästa av två eller flera individer väljs med en sannolikhet relaterad till lösningarnas lämplighet.

En av de vanligast metoderna för att välja individer är den stokastiska metoden roulettejuls val. I roulettehjulsmetoden tilldelas varje individ en del av ett roulettejul, där storleken av delen, p_i, är sannolikheten att den väljs. Sannolikheten bestäms av funktionen:

$$p_i = f_i / \sum_j f_j$$

Där f är en individs lämplighetsvärde. Rouletthjulet snurras sedan och individen roulettehjulet stannar på väljs till den nya populationen. Metoden upprepas tills den nya populationen är lika stor som startpopulationen.

Medan roulettehjulsmetoden är enkel har den nackdelen att individer med hög lämplighet lätt inte kommer med till den nya populationen, vilket kan orsaka en långsammare konvergering av algoritmen. Dessutom blir sannolikheten att den bästa individen försvinner större ju närmare alla lämplighetsvärden är varandra.

En annan populär metod är den deterministiska metoden turneringsval. Turneringsmetoden lämpar sig för om man har en mycket stor population, eller om populationen är distribuerad över flera system. [2] Detta på grund av att metoden inte behöver summera alla lämplighetsvärden eller kontrollera alla individers lämplighet.

Metoden fungerar genom att slumpartat välja individer i grupper ur populationen, jämför deras lämplighet och spara den bättre av dem till den nya populationen tills den nya populationen är lika stor som ursprungspopulationen.

2.2.5 Mutation

Mutation används inom genetiska algoritmer för att introducera variation till genpoolen och hindra den från att stagnera och säkerställa den når ett mera optimalt värden än den annars skulle göra. Om mutation inte används kommer inom kort alla individer vara nästan identiska eftersom endast de från grunden bästa individerna för sina gener vidare. [8] Sannolikheten för mutation per gen ligger generellt omkring 0.1%, men kan variera kraftigt.

Mutation introducerar visserligen variation till genpoolen, men tyvärr förstör den även eventuellt viktig information. På grund av det här måste en balans uppnås mellan att införa variation och minimera skadan den gör.

2.2.6 Rekombination

Rekombination är processen att skapa en ny lösning utgående från två eller flera tidigare lösningar, så kallade föräldralösningar. [2]

De vanligaste typerna av rekombination är den som används för räckor, samma typ av rekombination som används för lösningar med binär representation.

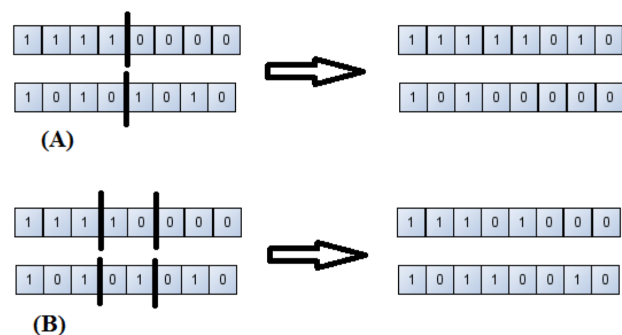


Fig 2: Illustration över (A) enpunktsrekombination, och (B) N-punktsrekombination med n=2

Enpunktsrekombination fungerar genom att man väljer två föräldrar och sedan väljer en punkt på dem. Denna punkt är korsningspunkten och båda föräldrarna delas i denna punkt och byter ”svans” som illustrerat i Fig. 2.

N-punktsrekombination fungerar på samma sätt, men med *n* stycken punkter istället, där *n* är ett slumpmässigt antal längder.

Till skillnad från enpunkts- och N-punktsrekombination fungerar likformig rekombination genom att istället slumpmässigt välja för varje del av strängen i ena föräldern och bestämma om den ska byta plats med samma del i den andra föräldern.

Vid rekombinering av individer bestående av en räcka heltal eller flyttal kan samma operatorer som för binärtalsrepresentation användas, och så är ofta fallet för heltal, detta eftersom en blandning, så kallad aritmetisk rekombination, av av ett udda och ett jämt heltal ger ett svar som inte är ett heltal.

Rekombination av räckor med flyttal använder sig generellt av aritmetisk rekombination. Rekombination av en räcka flyttal fungerar i princip som vanlig rekombination, förutom att de tal som flyttas mellan föräldrarna får ett medelvärde av båda föräldrarna enligt:

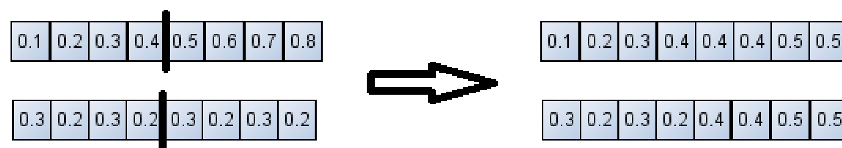


Fig. 3: Artitmetisk enpunktsrekombination av flyttal, då $\alpha = 0.5$

$$\text{Barn } 1 = \alpha \cdot x + (1 - \alpha) \cdot y$$
$$\text{Barn } 2 = \alpha \cdot y + (1 - \alpha) \cdot y$$

Exempelvis leder detta i enpunktsrekombination till resultatet i Fig. 3.

Vidare finns det många andra typer av rekombination, exempelvis rekombination som kombinerar flera än två föräldrar och kant-, ordnings- och cykelrekombination som alla kan användas i olika situationer. Dessa kommer dock inte beröras.

2.3 För- och nackdelar

Genetiska algoritmer har många fördelar, där en av de främsta är att genetiska algoritmer som sådana egentligen är parallella. Medan de flesta algoritmer arbetar seriellt, och måste överge arbetet och börja om då den inte kommer fram till en lösning, kan genetiska algoritmer istället överge bara den lösningen och fortsätta med alla andra lösningar den arbetar med samtidigt. [3]

Samtidigt är genetiska algoritmer lämpade för olinjära problem, speciellt de som är för stora för att lösa med seriella algoritmer. Genetiska algoritmers förmåga att arbeta på flera lösningar samtidigt, och för den delen genetiska algoritmers snabba konvergering mot en lösning gör att de lämpar sig för de flesta problem. Exempelvis lyckades ingenjörer från General Electric och Rensselae Plytechnic Institute tillsammans producera en jetmotor som var tre gånger bättre än någon människodesignad, och 50% bättre en den skapad av ett expertsystem. Detta genom att söka igenom en sökrymd på 10^{387} lösningar efter en optimal lösning. Ett arbete som annars kan ta upp till fem år och kosta två miljarder dollar att slutföra, med den genetiska algoritmen de utvecklade lyckades på skapa den på två dagar med en vanlig ingenjörs arbetsdator.[9]

En annan av genetiska algoritmers styrkor är att de till skillnad från de flesta optimeringsalgoritmer relativt enkelt kan navigera igenom problem med stor sökrymd,[3] samt genom problem som har många lokala optimala värden och hitta det globala optimala värdet, eller åtminstone ett som är nära. Detta är på grund av att algoritmen

söker på flera ställen samtidigt och jämför deras lämplighet och i sin tur konvergerar mot det globala optimala värdet.

Genetiska algoritmer har även den fördelen att de kan behandla många parametrar samtidigt, till skillnad från många seriella algoritmer, vilket ofta kan leda till flera lika bra lösningar för ett problem.

Någonting som från början kan tros vara en nackdel med genetiska algoritmer, men visar sig vara till fördel är att de inte vet någonting om problemet de försöker lösa. Algoritmen gör slumpartade förändringar i lösningsförslagen och testar hur bra de lämpar sig. Fördelen med det här är att genetiska algoritmer alla val är baserade i slumpen, alla sökvägar är teoretiskt sätt öppna för algoritmen, till skillnad från algoritmer med förkunskap som eliminerar vissa sökområden, och därav eventuellt vissa lösningar. Det här kan leda till originella lösningar som annars inte skulle ha nåtts.

Genetiska algoritmer har givetvis också sina nackdelar, varav den viktigaste är att lösningarna måste kunna representeras på ett bra och robust sätt. Lösningens representation måste tillåta mutation och korsning med andra lösningar för att gå att använda. Till på det måste representationen alltid ge någon form av resultat, och inte evaluera till nonsens, någonting som lätt kan hända vid implementation i olika programmeringsspråk.

Att skriva en lämplighetsfunktion blir också något av ett problem då funktionen måste definieras precist och se till att bättre lösningar får ett högre lämplighetsvärde.

Utöver lämplighetsfunktionen måste parametrarna för själva genetiska algoritmen, dvs populationsstorlek, mutationssannolikhet, korsningssannolikhet och metod för att välja föräldrar, preciseras för problemet. Detta är väldigt viktigt då en för liten population kan leda till att algoritmen missar viktiga lösningsområden. Om takten som lösningarna ändras är för stor eller algoritmen för val av föräldrar är opassande kan goda lösningar gå förlorade innan de kan testas. Dessa fel kan i värsta fall leda till att algoritmen aldrig konvergerar och resultatet är helt slumpartat.

Ett annat stort problem med genetiska algoritmer är risken för förtida konvergering. Förtida konvergering sker då en av lösningarna är långt bättre än alla andra i populationen, vilket orsakar att den lösningen snabbt tar över populationen och eliminerar populationens mångfald för tidigt och i slutändan ofta leder till att resultatet blir ett lokalt optimalt värde istället för ett globalt. Många olika typer av selektion har introducerats för att behandla problemet, däribland turnerningsval, en algoritm för att välja individer, samt olika funktioner för att skala lämplighetsvärdena, sigmaskalning.

Genetiska algoritmer kan troligen användas för att lösa alla problem, minimera alla funktioner etc., dock är det inte rekommenderat. [9] Detta på grund av att det i många fall, exempelvis analytiska problem, helt enkelt är snabbare och enklare att lösa problemet med en vanlig seriell algoritm.

2.4 Användning i spel

Spel är dynamiska och konkurrenskraftiga, en ideal testbädd för beräkningsbar intelligens teorier och algoritmer. [13] Enkelt sagt kan man se evolution som ett spel där belöningen för ett bra ”spel” är att låtas föra vidare sitt genetiska material till kommande generationer och fortsatt överlevnad. I naturlig evolution bestäms lämpligheten av en individ i samband med motstånd och medarbetare, ett koncept som inom evolutionär beräkning kallas samevolution. Med hjälp av samevolution har expertfulla spelstrategier utvecklats utan behovet av input från mänsklig expertis.

3. Procedurell innehållsgenerering

Procedurell generering introducerades på 1980-talet då enkla algoritmer applicerades för att skapa enorma, möjligen oändliga mängder spelinnehåll med endast begränsade resurser. Exempel på detta är de många rollspel som genererar oändligt många fängelsehålor att spela igenom, exempelvis datorspelet Rouge av Michael Toy och Glenn Wichman.

Andra modernare spel som använt procedurell innehållsgenerering är Diablo och Diablo 2 som använt dem för att generera spelplanerna, Spore av Maxis som låter spelaren skapa sina egna varelser och procedurellt skapar animationerna för varelserna. Ett annat exempel är Left4Dead[10] som lägger till fiender beroende på spelarens rörelser, så att fienderna alltid läggs till utanför spelarens synhåll för att skrämma spelaren. I Left4Dead är även delar av musiken procedurellt genererad. Det finns ett grundbibliotek musik, sedan blandas de och arrangeras dynamiskt baserat på spelarens tillstånd.

Ett problem i dagens spelindustri är att utvecklarna ofta vält att närma sig problemen de ställs in för med uttömmande algoritmer för att skapa innehållet de vill ha. [5] Detta blir dock mer och mer sällan möjligt på grund av den mängd processeringskraft som skulle behövas. Men ändå kan många av de problem man ställs inför i spelutveckling lösas med hjälp av genetiska algoritmer.

En annan fördel med att använda procedurell innehållsgenerering, istället för att skapa allting för hand, är att vissa saker man annars inte skulle ha kunnat göra blir möjliga.

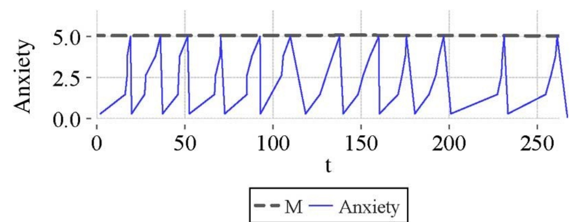
Procedurell generering har dock givetvis sina nackdelar. Små ändringar i koden eller variablerna för att generera procedurellt innehåll kan ha stora och oanade konsekvenser. Dessa konsekvenser kan handla om klara synliga fel, eller bara en par små områden i en världskarta som inte blir som den ska och problemet inte upptäcks förrän kvalitetskontrollen. Detta leder till att koden måste skrivas om och det komplette resultatet än en gång gås igenom.

3.1 Procedurell generering av banor med genetiska algoritmer

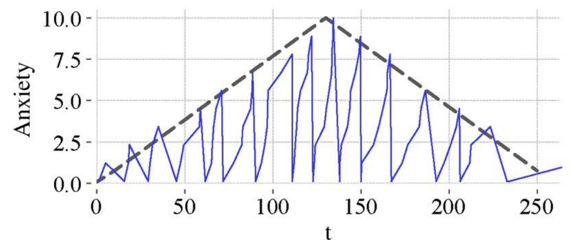
Dagens spel består generellt en stor mängd innehåll som ofta är dyr att skapa, exempelvis stora detaljerade spelvärldar. En lösning på det kostnadsproblemet är algoritmer för att generera innehållet, varav genetiska algoritmer är ett av alternativen.

Troligen allt innehåll i spel kan skapas med hjälp av genetiska algoritmer, problemen med att förverkliga det är att bestämma en grundpopulation och hur man ska beräkna lämplighetsvärdet för varje ny lösning.

En av de generellt största delarna av spels innehåll är deras banor. Att skapa en bana med algoritmer, och för den delen att sedan använda genetiska algoritmer för att korsa dem skulle troligen vara rätt enkelt, problemet är när man ska bestämma vilken av alla de skapade banorna som är bäst, eller i alla fall tillräckligt bra.



(a)



(b)

Fig 4: Mängden ackumulerad utmaning över tid, då T_{window}=10, m=1 (a) Statisk M (b) Varierande M

Sorenson et al.[16] bestämde lämpligheten av tvådimensionella, genererade banor genom att först generera banor och dela in dessa i *n* stycken rytmgrupper. Dessa rytmgrupper har gränserna $t_0, t_1, \ldots, t_n$. Rytmgrupperna identifieras sedan genom att analysera den skapade banan och söka efter områden med tillräckligt låg utmaning, där utmaningen definieras som *c(t)*. Ett fönster av storleken T_{window} skiftas sedan över banan och lägger rytmgruppernas gränser där den totala utmaningen är lägre än tröskelvärdet *m*. Gränserna placeras på positionerna *t*, där $\int_{t-T_{window}}^{t} c(t)\,dt \leq m$. Efter att ett en gränspunkt placerats placeras inte en till förrän den totala utmaningen gått över tröskelvärdet *m*. Om utmaningen för en rytmgrupp definieras som $\int_{t-T_{window}}^{t} c(t)\,dt \leq m$ kan genom applikation av Yerkes-Dodsons lag[17] mängden nöje över hela banan definieras som

$f=\sum_{i=0}^{n}\frac{2c_i}{M}-\frac{c_i^2}{M^2}$ där M är en övre gräns för mängden utmaning. Resultatet av den funktionen är illustrerad i Fig 2[16].

Den här modellen applicerades av Sorenson et al. på spelen Inifinite Mario, en java implementation av Nintendos Super Mario Bros. skriven av Markus Persson, och Nintendos The Legend of Zelda. Super Mario Bros. är ett tvådimensionellt plattformsspel och The Legend of Zelda är ett actionäventysspel där spelaren måste ta sig genom sammanhängande fängelsehålor fyllda med fiender, föremål och pussel.

Sorenson et al. Implementation av algoritmen för Infinite Mario var mycket lyckad och placerade tredje i en AI bandesignstävling, dock blev arbetet mycket mera avancerat när den skulle implementeras för The Legend of Zelda och kraven för många och exakta, vilket ledde till att det tog upp emot tio minuter att skapa en godkänd bana.

Ett annat arbete som gjorts är det av Loiacono et al. [6] för att generera banor till racingsimulatorn TORCS (The Open Source Racing Car Simulator). Liacano et al. valde att försöka generera banor som är varierande, istället för banor som matchar en viss typ av spelarprofil, där de viktigaste metriken är variationen mellan kurvor och raksträckor och hur långt vissa körhastigheter kan hållas på banan.

I TORCS representeras banor som en sorterad lista, där varje element i listan är endera en raksträcka eller en kurva. Om elementet är en raksträcka har den bara längd som parameter, medan om det är en kurva representeras den av riktning, radianer (vinkel), start radie och slutradie. Ett av de krav som ställs på generering är att banan är sluten. Vilket innebär att varje segment måste överlappa med föregående, och det sista med det första.

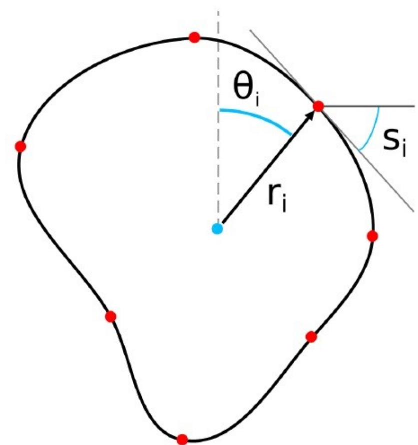


Fig 5:Röda punkterna representerar p medan blå punkten är utgångspunkt. Kurvorna är resultatet av genotyp till fenotyp anpassningen.[6]

Liacono et al. Valde att representera banan som en räcka kontrollpunkter $\vec{p}=\{p_{1,}\cdots p_n\}$, där p_i består av de tre parametrarna r_i, θ_i och $s_{i.}$ där r_i och θ_i representerar kontrollpunkten. r_i Är kontrollpunktens avstånd till banans utgångspunkt och θ_i är vinkeln till punkten. s_i Representerar punktens riktning, dvs vart vägen är riktad i den punkten. Representationen av punkterna är illustrerad i Fig. 3.

Sedan genereras punkterna till en TORCS bana, och om banan är sluten evalueras den. Först definieras banans variation som entropin av banans kurvprofil C, dvs fördelningen

kurvvärdena över alla bansegment. Sedan bestäms entropin av banans hastighetsprofil *S*. Hastighetsprofilen bestäms genom att låta bottar köra igenom banan och varje speltick hämta bilens hastighet. Dessa profiler kategoriseras sedan hur väl de stämmer överens med föranalyserade kurvprofiler och hastighetsprofiler av människogjorda banor, indelade i 16 kategorier vardera. För varje del av profilen som stämmer överens med en de av 16 fördefinierade typerna inkrementeras värdet av den typen i vardera räcka $S=\{s_1,\dots s_{16}\}$ och $C=\{c_1,\dots c_{16}\}$. Lämplighetsvärdet för banorna representeras sedan av entropin värdena för S och C , som bestäms av formlerna:

$$H(S)=-\sum_{i=1}^{16} s_i \log_2 s_i \mid H(S)\geq 0,\; H(S)\leq \log_2 16$$

$$H(C)=-\sum_{i=1}^{16} c_i \log_2 c_i \mid H(C)\geq 0,\; H(C)\leq \log_2 16$$

3.2 Procedurell generering av grafik med genetiska algoritmer

Genetiska algoritmer kan även med fördel användas för att procedurellt generera adaptivt innehåll, dvs innehåll som ändras utgående från respons från spelaren.

Genetiska algoritmer har också tidigare tillämpats för procedurell generering av texturer. [15] Individerna representerades då som uttrycksträd och rekombination görs genom att kombinera grenar från träden. Vegetation, dvs träd och andra växter, har också genererats på ett liknande sätt.

Realistiska modeller och animationer av moln är en av de svåraste saker man kan göra inom animation. [4] Ett dock rätt enkelt sätt att göra realistiska modeller av moln är med genetiska algoritmer. Individen kan definieras som en räcka värden implicita sfärer, där sfären representeras av x,y,z,r,R_{max} och w. Där x,y,z representerar en position i tre dimensioner, r en intern radie, R_{max} ett avstånd från den interna radien, och w vikten som den interna sfären har.

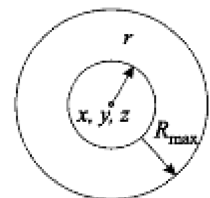


Fig. 6: Illustration av implicit sfär

Sedan evalueras dessa räckors lämplighet som moln, exempelvis med lämplighetsfunktionen: $f=\sum_{i=1}^{n} | b_i \cdot \sin(\arccos\left(b_i^2+v_i^2-\frac{a_i^2}{2\mathrm{b}_i c_i}\right))$

3.3 Procedurell generering av musik med genetiska algoritmer

Storskaliga experiment har visat att det så kallade normaliserade informationsavståndet är bland de bästa metriken för melodiklassificering. [14] Detta har givetvis lett till att den använts för genereringen av musik.

Exempelvis kan noter reprepresenteras som heltal. Ett vanligt piano har 88 tangenter, representerade från talen 1 till 88, med 0 för att representera tystnad. Då har vi ett alfabet av längden 89. Dock kan denna vidare begränsas ner till två oktaver, 24 – 48 och 0, eftersom mycket musik endast använder dessa två och det reducerar sökrymden. Man kan då välja att representera endast noterna, och inte längden av noterna, detta eftersom man funnit att musik ännu är igenkännbar även om längden av alla noter ändras.

Medan det normaliserade informationsavståndet inte är direkt tillämplig finns en modifikation av det, nämligen de normaliserade kompressionsavståndet (NCD):

$$NCD(x,y)=\frac{max\{C(xy)-C(x),C(yx)-C(y)\}}{max\{C(x),C(y)\}}$$

Där *xy* är konkateneringen av strängarna *x* och *y*, *C(x)* är längden av strängen x, komprimerad av valfri kompressionsalgoritm. Lämpligheten räknas sedan ut enligt

$$f(x)\left(\sum_{g_i} NCD(x,g_i)\right)^{-1}$$

där g_i representerar en fördefinierad guide, exempelvis NCD värdet av en låt av Chopin. Detta får algoritmen att skapa musik i en liknande still till valfri låt genom att endast ge den en guide, eller i stilen av en kompositör genom att ge algoritmen flera guider.

5. Avslutning

Sedan John Holland introducerade genetiska algoritmer har de applicerats på alla möjliga problemområden och med 80-talets introduktion av procedurell generering av innehåll i spel var det bara en tidsfråga innan genetiska algoritmer skulle anpassas för de många problem man ställs inför i spelutveckling.

Genetiska algoritmer kan i princip appliceras på alla problem, förutsatt att de går att uttrycka och evaluera, detta har lett till att de börjat appliceras på stora problemrymder så som genereringen av spelbanor, musik och grafik.

I dagsläget skapas stora delar av innehållet i spel procedurellt och mera av arbetet börjar falla på algoritmer för att skapa innehåll istället för kodare och designers.

I framtiden kan man troligen använda genetiska algoritmer för att skapa stora delar av innehållet i spel. Allt från ansiktsanimationer till musik till spelbanor kan genereras med hjälp av genetiska algoritmer och i och med Moores lag finns bara mer och mer processorkraft tillgänglig och med den nuvarande trenden inom parallellisering istället för att satsa på högre klockfrekvenser ligger genetiska algoritmer ypperligt till med deras högra grad av parallelisering och snabba kovergering inom stora problemområden.

Illustrationsförteckning

Fig 1: Enkel beskrivning av hur EA i grunden fungerar..5
Fig 2: Illustration över (A) enpunkts rekombination, och (B) N-punkts rekombination med n=2….10
Fig. 3: Artitmetisk enpunktsrekombination av flyttal, då 11
Fig 4: Mängden ackumulerad utmaning över tid, då Twindow=10, m=1 (a) Statisk M (b) Varierande M...14
Fig 5:Röda punkterna representerar p medan blå punkten är utgångspunkt. Kurvorna är resultatet av genotyp till fenotyp anpassningen.[6]..15
Fig. 6: Illustration av implicit sfär...16

Litteraturförteckning

1: A.E Eiben & J.E Smith, Introduction to evolutionary computing, 2 ed., Springer, år 2007, Sid 15-37
2: A.E Eiben & J.E Smith, Introduction to evolutionary computing, 2 ed., Springer, år 2007, Sid 37-69
3: Adam Marcyck (2004-04-23), "Genetic Algorithms and Evolutionary Computation", Tillgänglig:http://www.talkorigins.org/faqs/genalg/genalg.html (Artikel) Hämtad: 2012-03-30
4: Bogdan Lipus, Nikola Guid, "Genetic Algorithms in Animation of Volumetic Clouds", i "Information Technology Interfaces, 2002. ITI 2002. Proceedings of the 24th International Conference on Digital Object Identifier", 2002, sidor: 423-428
5: Chris Remo (2008-11-18), "MIGS: Far Cry 2's Guay On The Importance Of Procedural Content", Tillgänglig: http://www.gamasutra.com/php-bin/news_index.php?story=21165, hämtad: 2012-03-30
6: Daniele Loiacono et al., "Automatic Track generation for High-End Racing Games Using Evolutionary Computation", Transactions on Computational Intelligence and AI in Games, IEEE , Sep. 2011, vol. 3
7: Eddie Makuch (2012-01-20), "Star wars: The Old Republic cost §200 million to develop" (Artikel) Tillgänglig: http://www.gamespot.com/news/star-wars-the-old-republic-cost-200-million-to-develop-6348959 Hämtad: 2012-03-30
8: H. M. Cartwright, "An Introduction to Evolutionary Comutation and Evolutionary Algorithms" i Applications of Evolutionary Computation in Chemistry, R.L. Johnston, Springer, år 2004, sid 2-29
9: Holland, John. "Genetic algorithms." Scientific American, July 1992, p. 66-72.
10: http://pcg.wikidot.com/pcg-games:left4dead, Revision: 9, Ändrad: 2009-10-18, Hämtad: 2012-03-30
11: K. A De Jong, "Genetic Algorithms Are NOT Function Optimizers" i Foundations of Genetic Algorithms 2, Morgan Kaufmann, år 1993, sid 5-17
12: K. Stanley daterat 2012.01.12 (läst 2012.03.09) The NeuroEvolution of Augmenting Topologies (NEAT) Users Page http://www.cs.ucf.edu/~kstanley/neat.html
13: Lucas S.M., Kendall G., "Evolutionary Computation in games", Computational Intelligence Magazine, IEEE , Feb. 2006 sidorna 10-18
14: M. Alfonseca et al., "A simple genetic algorithm for music generation by means of algorithmic information theory", i "Evolutionary Computation, 2007, CEC 2007, IEEE Congress On", sid 3035 - 3042
15: Mark Hendrikx et al., "Procedural Content Generation for Games: A Survey", "ACM Trans. Multimedia Comput. Commun. Appl.", Artikel 1, Februari 2011
16: Nathan Sorenson et al., "A Generic Approach to Challenge Modeling for the Procedural Creation of Video Game Levels", Transactions on Computational Intelligence and AI in Games, IEEE , Sep. 2011, vol. 3

17: R. M. Yerkes och J. D. Dodson, "The relation of strength of stimulus to rapidity of habit-formation," J. Comparat. Neurol. Psychol., vol. 18, s. 459-482, 1908
18: William Usher (2011,02,28). "Video Games Market To Grow From §52 Billion in 2009 To §86 billion By 2014" (Artikel). Adress: http://www.cinemablend.com/games/Video-Games-Market-Grow-From-52-Billion-2009-86-Billion-By-2014-30363.html Hämtad: 2012-03-30