Hej, mitt namn är Levi Tuoremaa, och jag har skrivit ett program som slumpar fram städer i spelmotorn Godot.

Programmet består av två huvudsakliga komponenter. Vi slumpar fram byggnader med tekniker för procedurell geometri, likt dom som togs upp under en tidigare föreläsning. Sedan placerar vi dessa byggnader i en terrängkarta som genererats med algoritmen Wave Function Collapse. Innan vi förklarar vad detta innebär kan vi ta en titt på det slutliga resultatet.

(under trailer)

Allt ni ser här är procedurellt. Det finns inte en enda modell eller textur i projektet. Det är bara vektormatte. Allting är även gjort på en ganska låg nivå, med triangeltabeller och manuellt beräknade normaler och UV-koordinater och allt sånt roligt. Det innebär att all grafikprogrammering borde vara lätt att överföra till OpenGL, då den inte bygger på någon funktionalitet som är unik till Godot. Ljussättningen är dock 100% Godot, har inte skrivit någon egen kod för det.

Vi går över till generering av byggnader. Genereringen sker enligt en hierarkisk modell där objekt, vid instansiering, kör kod som skapar nya objekt som i sin tur kör kod som skapar nya objekt. I vårt fall har vi tomter, som skapar husets fundament, som skapar väggar, som skapar fönster, som skapar ljuskällor ifall de är upplysta fönster. Vissa objekt kommer också att slumpa några av sina egna parametrar, för att skapa variationer i form och färg.

Vi går igenom processen steg för steg. Vi börjar med en tomt, som markerar den yttre gränsen för huset som ska genereras däri. Ingen del av huset får sticka utanför tomtgränsen. I nästa steg skapas en rektangel någonstans i tomten. Denna kommer utgöra husets kärna. För att huset ska få en mer intressant form, klipps några av hörnen av lite slumpmässigt. I nästa steg genereras väggar. Väggarna saknar back-faces, så de ser lite konstiga ut innan vi ritat tak. Väggarna får också en rad med fönster för varje våning. I nästa steg lägger vi till utbyggnader på en del av väggarna. För att en vägg ska få en utbyggnad måste den ha en viss minimilängd, och det måste också finnas utrymme i tomten. Slutligen skapas tak. Det finns en bugg i programmet; taket har en tendens att hänga över tomtgränsen. Detta är generellt dåligt för grannsämjan i området, så det är något jag borde åtgärda.

Över till Wave Function Collapse, hädanefter WFC. WFC är en algoritm som blivit populär på senare år som används för bland annat procedurella modeller och texturer, samt slumpgenererade nivåer i tv-spel. Begreppet kommer från kvantmekaniken, där en vågfunktion existerar i en superposition av flera kvanttillstånd tills den observeras, varpå den kollapsar till ett enda definitivt tillstånd. Alternativt kan man se den som en Markovkedja i flera dimensioner.

Men jag tror att det blir enklare med ett exempel, så jag har förberett en demo som illustrerar WFC i två dimensioner. Till vänster har vi en samling tiles, som alla har regler definerade för vilka andra tiles de kan gränsa mot i 4-grannskap, dvs. upp, ner, vänster, höger. Så länge vi har dessa regler definerade kan WFC-algoritmen självmant slumpa fram en karta som följer dessa regler. (Tryck)

Vi går igenom ett steg i taget. Vi börjar med en tom karta, och placerar en slumpmässig tile någonstans. Då dyker det upp en massa siffror. Dessa representerar entropin i varje ruta, dvs. antalet tiles som kan platsa in där. Eller i kvantfysik-termer, antalet tiles som existerar i superposition där. Vi väljer en ruta där entropin är lägst, och kollapsar vågfunktionen där, dvs. vi väljer slumpmässigt en av de tiles som är möjliga där. När vi gör detta ändras entropin i angränsande rutor, för det finns nu färre rutor som kan platsa in där. Vi upprepar processen tills kartan är ifylld.

Det fina med WFC är att man lätt kan slänga in nya tiles i genereringen, så länge de har regler för grannskap definerade. Så vi testar inkludera lite tiles som representerar floder, sjöar och broar… Och så får vi en lite mer varierad karta. Det blir nollor ibland. Det kan uppstå situationer där ingen tile platsar in i en cell. Då blir det hål som inte kan fyllas i. Vad man kan göra i dessa situationer är att gå tillbaka några steg i genereringen, och försöka igen med en ny random seed. Har dock inte implementerat detta.

Samma princip fungerar i 3D, men nu jobbar vi med 3-dimensionella block som har grannskap definerade i 6 riktningar, +/- i x, y och z-axeln. Tyvärr är min terräng jättetråkig, jag har inte fått in så många terrängblock ännu. Det tar tid att skriva triangeltabeller. Ni ser till exempel att vi har bara höjdskillnader i en dimension, för jag har inte gjort några hörnblock där två kullar kan mötas. Det blir även hål i terrängen för att jag inte gjort terrängblock som platsar in överallt.

Terrängblocken innehåller också tomter, som fylls i med hus enligt tekniken jag beskrev tidigare. På så sätt får vi ett stadslandskap. Jag nämnde inget om texturerna; de är procedurella, och består av en slumpmässig färg blandat med lite noise för att få fasaden att se vädrad ut. UV-koordinaterna för husväggarna beräknas med avseende till husets totala omkrets för att få texturerna att löpa runt hörn. Vi kan se UV-koordinaterna bättre om vi byter till en gradient-textur. UV-koordinaterna är dock lite buggiga; i en del fall löper de runt hörn korrekt, men i andra inte.

Vi kan också ändra lite parametrar i genereringen. Vi kan öka den maximala våningshöjden för att få något mer likt en storstad. Vi kan också justera stadens totala storlek.

Jag har i detta projekt använt WFC för terräng, vilket egentligen inte är det idéella användningsområdet för WFC. WFC är nödvändigtvis rutnätsbaserad, vilket ger ett kantigt, oorganiskt utseende. Det är till fördel om du använder algoritmen för att t.ex. göra nivåer till ett rutnätsbaserat spel, eller om du jobbar med syntetiska motiv som gärna får vara lite kantiga. Det är generellt inte en teknik som du vill använda för realistisk terränggenerering.

WFC följer regler lokalt, men är nonsens på makronivå. Om vi går tillbaka till demon och gör ett vägnätverk, så ser ni att alla vägar ansluter korrekt till andra vägar, men på makronivå blir det ett labyrintartat vägnätverk som ingen stadsplanerare i världen hade godkänt.

Tankar om Godot, nu när jag använt det för ett helt skolprojekt? Det jag gillar med Godot är att det är ett oerhört lättviktigt program. Det är en 100 mb executable som inte kräver någon installation och som startar på nolltid. Ändå är det förvånansvärt nära Unity i funktionalitet. Det har kraftfulla, intuitiva verktyg för gränssnittsdesign som gör att jag mycket lätt kan koppla funktionalitet i mina program till knappar och sliders. Jag vet inget annat program där jag lika snabbt kan omvandla en idé till en funktionell prototyp. Det är också open-source, utvecklat av volontärer och gratis nu och för evigt. Starkt rekommenderat, jättekul program.

Det var allt jag hade att ta upp. Några frågor?