Reflektionsdokument Peck ’n Flap

Elias Backström Thiel, eliba473
Leo Diaz Sanchez, leodi727

Del 1, Originalkonceptet

Till en början var planen att göra en Terraria-klon. Efter att ha lärt känna mallen för gameloopen och states som vi fått började vi med vår map. Till en början försökte vi skapa allt relaterat till vår map i en egen klass men det blev snabbt uppenbart att vi behövde dela upp det i en tile-klass som håller koll på varje enskild tile och en map-klass som genererar mappen och sparar vart alla tiles ligger. Det tog ett tag att komma på en bra metod för att slumpa mappen, men slutligen kom vi fram till att det bästa är att generera en slumpmässig höjd och sedan fylla upp allt under den med tiles. Efter ett tag så kom vi fram till att en sigmoid-kurva var effektiv, men denna fick problem när man bytte vilken min- och maxhöjd generationen skulle ha, därför bytte vi senare till perlin noise-generation.

Nästa problem var kollision. Den första versionen gav spelaren ett ”grounded”-värde som bestämde ifall spelaren skulle fortsätta falla neråt eller inte, denna sattes till false när spelaren rörde en tile. Detta skapade dock buggar när man byggde in sig själv då man kunde falla genom mappen, den fungerade inte heller i 100% av fallen. Efter mycket klurande gick vi över till en kollisionscheck som kollar ifall spelaren kommer röra sig in i en tile och då stoppar den. Detta hade dock 2 problem, ifall man gick in i ett block som satt ungefär på mitten av spelaren kunde man passera igenom och ibland skapades små mellanrum mellan blocken och spelaren. Det första problemet berodde på att kollisionen bara mättes i hörnen av spelaren (som är 2 tiles hög) vilket gjorde så att ett ”fritt” område skapades i mitten. Genom att täcka upp det fria området löstes detta. Det andra problemet uppkom när spelarens hastighet var större än distansen till en tile, detta löstes genom att beräkna in spelarens hastighet så att man kunde komma hela vägen fram till väggen.

Det sista, och största problemet, var kameran. Eftersom världen var större än skärmen (den synliga ytan) vill man ju kunna röra sig runt, och då ha en kamera som fokuserar på spelaren. I början försökte vi använda oss av javaFX:s translate-funktion. Detta gjorde dock så att draw-funktionen helt slutade fungera när spelaren gick utanför ”skärmen”. Detta gick att lösa genom att modifiera vilka värden som sätts in i translate (dvs inte bara spelarens position), men x- och y-koordinaterna för interaktion (tex kollision) blev helt fel så fort programmet började använda translate. Det var ungefär vid den här punkten vi kom på att vi inte skulle hinna klart och gick över till att skapa en flappy bird-klon.

Del 2, flappy bird

Denna process gick betydligt mycket smidigare. Detta förmodligen då projektet var bättre planerat. En större designutmaning var timingen på när nya rör kom ut. I början skapades nämligen illusionen av att de ”sögs” bort på vänster sida, vilket nästan gjorde en åksjuk av att spela. Ett annat, mindre, problem var att man fick dubbla poäng varje gång man passerade en röröppning. I början var detta inte ett problem då man bara kunde halvera poängen när spelaren dog, men när vi implementerade en live-uppdatering av poängen blev det

problematiskt. Att halvera poängen när man fick dem fungerade inte eftersom poängen var en integer. Den slutgiltiga lösningen var att redan när rören genereras markera bottenröret som ”scored”, dvs att man endast får poäng från det övre röret.

Övrig reflektion

Hur var planeringen av projektet?
Anledningen till att vi fick så många problem, och inte redan i början insåg hur långt tid det skulle ta med vår första idé var förmodligen pga dålig planering. Ifall vi redan innan lagt uppdelarna som behövdes framför oss hade vi nog insett hur viktig kameran var för projektet. Ifall vi vetat det hade vi förmodligen kunnat börja med kameran och möjligtvis löst det, eller i alla fall kommit fram till att projektet skulle tagit för långt tid tidigare.

Hur hjälpte utvecklingsverktygen er?
Vi valde att använda intellij vilket fungerat fenomenalt tillsammans med git, det har varit enkelt att pusha, commita och merga då dessa funktioner är inbyggda i inttelij. Utöver ett par merge-konflikter har vi varit bra på att dela upp arbeta vilket låtit oss undvika de mesta konflikter.

Vad hade kunnat göras annorlunda designmässigt?
I överlag så är designen på Peck ’n flap rätt så bra, förmodligen eftersom vi kom in med en tydlig plan. Om något skulle ändras skulle man möjligtvis kunna sätta ihop fiendeklasserna då det ända som skiljer dem åt är deras skada och hastighet/håll de rör sig åt. Det kan däremot lätt bli otydligt när man läser koden ifall alla fiender är samma klass, och dessutom skulle konstruktorn behöva väldigt mycket information. Det är därför tydligare och bättre att ha dem i separata subklasser som båda ärver från en superklass.

Vad var den största lärdomen från projektet?
Definitivt planeringen. Vi borde redan från börjat förstått vikten av en fungerande kamera och byggt projektet med det i åtanke. När vi väl började med kameran så var koden redan så pass komplex att det blev svårt att se vad som inte fungerade. Vi har även lärt oss en hel del om att generera en värld med olika slump-metoder. Förutom att lära oss mer om sigmoid-kurvor och perlin noise så försökte vi även utveckla egna metoder, och utforskade användbarheten av voronai noise för tex generation av bergsmineraler i Terraria-klonen.
Vi har även fördjupat oss i spel-loopen och hur den fungerar. Koncept som deltatime har blivit mer logiska och vi har lärt hos använda det för att skapa ett välfungerande spel.

UML-diagram (översikt)