**Innlevering i Tools Programmering**

**Philip Christensen (chrphi14) & Alexander Larsen (larale14) - Westerdals Oslo ACT 2016**

**Tutorial**

**Level Editor**

Vi har laget en kort video-tutorial som du kan bruke hvis du skulle lure på hvordan man navigerer seg rundt i selve map-editoren. Den ligger i den øverste folderen i hierarkiet, og har navnet “MapEditorTutorial.webm”.

*Hvis du ikke ser videoen - slik bruker man editoren:*

Helt øverst i venstre hjørne finner man filmenyen. Her kan man lagre til, eller hente fra, databasen man benytter seg av.

Den største delen av vinduet opptas av tile-mappet. Det er lokalisert i øvre venstre område av vinduet, som en stor grid. Her kan man hovre over ulike tiles, og man kan dra tiles fra preview-vinduet (mer om det under) og over hit for å legge til/endre på tiles.

Til høyre for tile-mappet finner man fil-hierarkiet. Her kan man bla seg gjennom filer på PC-en sin helt til man finner .png-filer, trykke på en av dem, og til slutt trykke på knappen hvor det står “Add image file to project”. Da vil filen legges til i området i nederste venstre hjørne av vinduet, der hvor alle “current” bildefiler vises. I dette området kan man trykke på et av bildene man har lagt til, og dette bildet vil deretter dukke opp i preview-vinduet i nederste høyre område av vinduet.

Preview-vinduet viser det bildet man opererer på for øyeblikket, samt en “info-grid” og en rotasjonsknapp. Info-griden representerer hvorvidt de ulike områdene av en tile skal være “traversable” eller ikke. Den verdien info-griden viser i det man dragger og slipper preview-bildet til en tile, er det som vil bli lagret for den tilen. Tilsvarende vil rotasjonsknappen gjøre en 90-graders (per klikk) rotasjon mot klokken på preview-bildet, og den rotasjonen som er gjeldende for preview-bildet vil lagres til sin respektive tile dersom preview-bildet dragges og slippes på en tile.

Til slutt har vi også noen felter i øvre høyre hjørne av vinduet. Her kan man skrive inn et brukernavn for å registrere seg i brukerdatabasen, og submitte navnet med et trykk på knappen. Knappen lengst til høyre vil vise hvilke *andre* som er pålogget for øyeblikket.

**Database**

For å opprette databasen så kan du åpne Visual Studio-prosjektet og i Solution Explorer så vil du se MapEditor-databasen. Her kan du se hvordan databasen er bygd opp av to tabeller: “Tiles” og “Users”. For å publisere databasenmå du høyreklikke på “MapEditor” og trykke “publish”. Deretter må du finne din SQL Server, og koble til den. Når dette er gjort, kan du prøve å kjøre programmet. Hvis du eventuelt skulle fått en feilmelding der du får feil på en funskjon som gjør en “foreach Tile T in Tiles” eller liknende, så må du slette “LinqToSQLClasses.dbml”, og lage en ny “Linq to SQL Classes”-fil med nøyaktig samme navn. Når du har gjort dét må du importere tabell-koblingen fra SQL-databasen til Linq-filen.

Når du kjører programmet for første gang så sjekker programmet om det finnes noen “Tiles”-kolonner i databasen. Hvis ikke dette er tilfellet lager den 100 tiles for at programmet skal kunne bli brukt mot databasen. Programmet sjekker også om du har en allerede registrert bruker i programmet via en “User.txt”-fil som blir lagret i \MapEditor\CodeMvvm\bin\Debug. Denne filen blir opprettet hvis du for første gang skriver inn et brukernavn og trykker på “Enter Username”-knappen. Da sendes denne informasjonen til nevnte .txt-fil, og til databasen. Du kan ikke registrere en bruker som allerede er registrert i databasen, og etter du har registrert deg vil du forbli logget inn hver gang du åpner programmet. Hvis noen andre åpner programmet når du er logget inn, så vil de kunne se ditt brukernavn i tekstboksen øverst til høyre, og tilsvarende motsatt vei. For å teste dette kan du lage en falsk bruker ved å gå inn i SLQ-serveren sin “User”-tabell og finne dataene i tabellen. Her kan du legge til en ny bruker ved å skrive inn et navn, og når du så kjører programmet kan du manuelt endre variabelen “IsUserOn” fra 0 (avlogget) til 1 (pålogget) i databasen. Du vil da se i programmet at brukeren blir lagt til og logges av i realtime. Hvis du legger til selve brukeren i databasen som en ny rad samtidig som programmet kjører, så vil du ikke få den opp i programmet før du avslutter det og kjører det på nytt (vi har ikke fått lagt til en realtime “refresh”-funksjon på akkurat dette). Denne brukeren vil imidlertid dukke opp neste gang du kjører programmet.

*VELDIG VIKTIG:*

*Du må alltid krysse ut programmet og ikke stoppe programmet med stopp-knappen til Visual Studio!*

Grunnen til dette er at vi foreløpig kun registrerer at en bruker logger ut dersom vedkommende krysser ut vinduet manuelt.

Hvis du skulle være uheldig og avslutte programmet på en annen måte,så må du manuelt gå inn i User-databasen og endre “IsUserOn”-verdien for din bruker til 0.

**Unity**

Vi har laget en Unity plugin i Visual Studio som manifesterer seg som en “.lib”-klasse. Denne klassen består av en SQL-connection til SQL-serveren og henter all informasjon i tiles og legger dette i lister. Denne pluginen blir buildet til en “.dll”-fil, og Unity henter denne filen ved hjelp av et Unity script vi har laget. Scriptet tar informasjonen fra pluginen og printer ut Tilene ut som GameObjects. Akkurat nå vil Unity spawne mappet “opp-ned”. Vi vet hvordan vi får fikset dette (og vi har funksjonaliteten på plass i scriptet som kommentar), men dette førte til noen andre problemer, så vi lot Unity bare importere mappet “opp-ned” enn så lenge. Vi forsøkte også å få Unity til å rotere bildene riktig vei, men dette var mer utfordrende enn ventet, ettersom Unity tilsynelatende satte sitt eget origo på hvert av bildene, og rotasjonene ble feil rendret deretter. Dette er noe vi gjerne skulle sett nærmere på, hvis vi hadde hatt tid til det.

Hvis “path”-en til en sprite på din PC ikke korresponderer med “path”-en som ligger i databasen, så sjekker scriptet opp navnet på tilen, og søker etter det navnet i “Resources/Sprites” under egen prosjekt-folder. Pluginen er ikke koblet opp mot den lokale databasen, men en Azure-database. Dette er fordi maskinen ikke tillater at du kobler Unity mot den lokale SQL-databasen (i hvert fall ikke hos oss), derfor måtte vi da bruke en ekstern SQL server. Linken til vår Azure-database er hardkodet i scriptet, men vær obs på at vi kun benytter oss av en trial-versjon av Azure, som skal gå ut om en måned fra og med 17.12.2016.

**Systemarkitektur**

**Level Editor**

Innledningsvis ønsket vi å følge MVVM relativt slavisk. Det ble imidlertid ganske raskt tydelig at vi ikke hadde anledning til å gjøre nettopp det, slik vi ønsket. En del av funksjonene vi ønsket å implementere var rett og slett for avanserte å implementere riktig i henhold til MVVM.

For å ta et eksempel, så representerer vi alle tiles i mappet vårt med en selvlaget Tile-klasse. Denne Tile-klassen holder blant annet på et Image-objekt, og dette Image-objektet ønsker vi å adde til en grid som inneholder og viser alle tiles i viewet. Griden i viewet ser naturligvis kun hvert *Image*, ikke hver *Tile*, ettersom den har restriksjoner på hva den kan inneholde. Derfor trengte vi å knytte hvert enkelt Image til forskjellige eventer, blant annet når musen “hovrer” over dem, eller når brukeren “dragger” en tile fra Preview-vinduet til tile-mappet.

Da vi prøvde å gjøre dette via bindings og commands i trofast MVVM-struktur, tenkte vi først at det smarteste måtte være å representere alle Imagene som en ItemsCollection (gjør det mulig med Foreach-aktig behaviour i xaml) og legge ulike commands på imagene deretter. Å tilegne commands til ulike elementer i en ItemsCollection lar seg imidlertid ikke gjøre i WPF (vi aner ikke hvorfor). Derfor ble det en aktuell problemstilling å legge til hvert enkelt Image manuelt i xaml og legge til alle specs og commands for dem manuelt. Ettersom vi benytter oss av nøyaktig 100 Tiles i TileMapet vårt, var dette helt uaktuelt. Derfor endte vi opp med en hybrid-struktur, hvor vi gjør view-operasjoner i MainWindow.xaml og dens tilhørende .cs-fil, mens viewmodel/model-operasjoner fortrinnsvis gjøres i MainViewModel.cs.

Dette byr selvsagt på en del knot, en del repetativ kode, samt en del konvertering man gjerne skulle spart seg for. Det sier seg selv at vi gjerne skulle unngått å ha en egen Tile-klasse som genereres av databasen, samtidig som vi har en custom Tile-klasse vi selv har laget som brukes i forbindelse med viewet, og hver gang vi ønsker å lagre/hente data så må vi foreta konverteringer mellom disse. Vi klarte riktignok å jobbe rundt det til slutt, men vi påstår overhodet ikke at den overordnede arkitekturen er spesielt vakker.

En konsekvens av dette, er at spesielt loading og saving av Tiles tar unødvendig mye tid og ressurser. Ideelt sett kunne vi for eksempel bare oppdatert verdier i ViewModelen vår under loading og oppdatert Viewet automatisk via INotifyPropertyChanged-interfaces, slik vi har gjort med Users, men det ble vanskelig med den arkitekturen vi endte opp med.

Enkelte av funksjonene som foretar operasjoner til/fra databasen kjører i egne threads. Vi har riktignok ikke satt locks på threadene med tanke på nettverking (hvis én saver samtidig som én loader har vi ikke en løsning for øyeblikket). Threadene som eksisterer vil imidlertid ta en del av arbeidsmengden. Enkelte steder prøvde vi å opprette andre threads, men vi ble stoppet av at to threads ikke kunne dele objekter.

**Database**

SQL-serveren består av to tabeller: “Users” og “Tiles”. Vi hadde planer om å lage flere tabeller for å spre dataen mer effektivt (for eksempel ha en egen tabell med “traversable”-kombinasjoner og “mappe” hver enkelt tile til denne tabellen via en foreign key), men vi fant ingen måte å gjøre dette riktig på med Linq.

**Unity**

Bruker en “managed” plugin som foretar seg all tilkobling til SQL-serveren.

**Known Bugs**

*Utover de utfordringene som allerede har vært nevnt, så har vi registrert følgende bugs:*

**Sletting av tiles**

Vi har ikke lagt inn noen “slette”-funksjon for tiles i mappet, men det skal i utgangspunktet gå an å dragge en hvilken som helst ny tile over en som allerede har et bilde. Når vi gjør dette oppstår det imidlertid ofte noen problemer, som at en tile nekter å endre seg, eller at når vi slipper en tile oppå en annen, så vil verdien i en helt annen tile endre seg.

**Relative vs absolute paths**

I databasen lagrer vi hver enkelt tile med en absolute path (med unntak av de aller første default-tilene). Dette gjorde vi fordi vi antok at Unity krever en absolute path til filene når bildene skal importeres dit. Vi kom på senere at dette vil gjøre networkingen buggy hvis man skal jobbe sammen på samme prosjekt, ettersom én absolute path på én PC ikke nødvendigvis korresponderer med én absolute path på en annen PC. Vi har ikke hatt tid til å jobbe rundt dette, men hvis vi skulle jobbet mer på prosjektet i fremtiden ville vi ha tvunget brukerne av editoren til å forholde seg til en relative path i forhold til prosjekt-mappen, og gitt muligheten for å spesifisere en path til denne mappen i Unity når man vil importere.

**Kilder**

For å kunne ta i bruk et explorer tree som viser fil-hierarkiet slik at brukeren kan bla seg frem til en hvilken som helst .png-fil, har vi fulgt denne tutorialen, med enkelte endringer:  
<https://www.codeproject.com/articles/21248/a-simple-wpf-explorer-tree> (Sacha Barber, sist redigert 08.11.2007, brukt 12.12.2016).