

SchetsControl.cs

Hier hebben wij een member variabele (`win`) toegevoegd, die het schets window van de bijbehorende schets control bijhoudt. Dit deden wij omdat wij access nodig hadden tot de `opgeslagen` member van de `SchetsWindow` klasse (deze variabele houdt bij of het bestand al is opgeslagen of niet).

SchetsEditor.cs

Op regel 31 van `SchetsEditor.cs` hebben wij een nieuwe entry voor de file menu in de menu strip toegevoegd, deze is om het geopende schets window te sluiten. Verder hebben we twee extra methodes toegevoegd: `DeserializeTekenbareElementen` (nederlands is lekker lang, in het engels zouden wij het gewoon `Deserialize` genoemd hebben maar het hele project is half nederlands dus ja) en `Openen`. De eerste methode deserialized een file, waarvan de directory als parameter wordt toegevoegd in de methode, en zet deze om naar een lijst van tekenbare elementen. Dan de tweede methode, deze is voor het openen van een bestand (of een *.sp bestand of een normaal plaatje, dit is ook waar de eerste methode aangeroepen wordt). Tot slot hebben wij de `Nieuw` methode aangepast waardoor deze het nieuw aangemaakte schets window returned, dit was nodig voor de `Openen` window om een instance van het window object te krijgen, hierdoor moesten wij dus ook bij de event toewijzigingen de `Openen` methode aanroep wrappen in een lambda functie.

SchetsWin.cs

Hier hebben wij twee nieuwe member variabelen toegevoegd: `vast` en `opgeslagen`, `vast` geeft aan of de muis ingedrukt is en `opgeslagen` geeft aan of het bijbehorende bestand is opgeslagen of niet. Voor de rest hebben wij in de tools array nog twee tools toegevoegd, namelijk beide cirkel tools. We hebben ook een event handler gemaakt die `Afsluiten` heet, deze zorgt ervoor dat je een berichtje krijgt als je je bestand niet hebt opgeslagen en probeert de editor of het schets window te sluiten. Verder hebben wij een `Opslaan` methode toegevoegd, deze slaat het bestand op de gekozen manier op (dus ook het *.sp bestand waarmee je dus je stappen etc opslaat). Voor de vorig genoemde methode hebben wij dus ook een methode nodig die het bestand kan serializen: `SerializeTekenbareElementen` (vanzelfsprekend). We hebben bij `MaakFileManager` een entry voor het opslaan menutje toegevoegd.

Tools.cs

Hier hebben wij alleen twee tools toegevoegd en alle tools gewijzigd zodat we deze kunnen serializen en hebben we de teken logica weggehaald sinds die in `DrawableElements` hoort. We hebben `CirkelTool` en `VolCirkelTool` toegevoegd (deze klassen zijn vanzelfsprekend). Dus in elke klasse wordt een tekenbare object toegevoegd als je tekent, en bij de gumtool wordt uitgevonden welke objecten onder je muiscursor zitten en worden deze dan weggehaald.

Schets.cs

Hier hebben wij een nieuwe member variable toegevoegd, namelijk de `elementen` lijst, deze houd dus alle elementen die op de schets zitten bij. Verder hebben wij de `Slapen` methode gemaakt, deze wordt in `SchetsWin.cs` aangeroepen en slaat het plaatje zelf op (in png of jpeg bijvoorbeeld). Als compliment van deze methode hebben wij ook een `LaadPlaatje` methode toegevoegd, deze laadt een nieuw plaatje met de parameter als bestandsnaam. We hebben ook `Hertekken` gewijzigd, hier tekenen we alle elementen in de elementen lijst, in volgorde.

TekenbaarElement.cs

In dit bestand zitten alle types van tekenbare elementen. Deze zijn subtypes van de abstract class `TekenbaarElement` en hebben als annotation `[SERIALIZABLE]`, dit zorgt ervoor dat ze in een binair getal omgezet kunnen worden om dus in het *.sp bestand opgeslagen kunnen worden en geladen kunnen worden. De eerste subklasse dient als tekenbaar element voor een lijn en heeft als teken logica dat wat oorspronkelijk in `Tools.cs` stond, verder hebben we het `RechthoekElement` toegevoegd die een rechthoek represeneert en dan als laatste `CirkelElement` wat dus een cirkel represeneert. Ook hebben we een methode die elk object wiskundig roteert, ook de tekst (alleen roteert hij de tekst positie en niet de letters zelf, "It's a feature not a bug"). Tot slot hebben we dus de `HitTest` methode toegevoegd die in `Tools.cs` wordt aangeroepen bij de gumtool om te checken welke tekenbare elementen onder de muiscursor zitten om deze vervolgens te verwijderen. We hebben ervoor gekozen om de pentool tekeningen te zien als kleine lijntjes die ieder individueel uit gegumt kunnen worden. Dit omdat je soms een deel van je pentool tekening wilt wissen en niet meteen alles. Daarnaast hebben we ervoor gekozen om elke letter individueel te wissen, zodat je toch nog spaties in het programma kunt toevoegen.