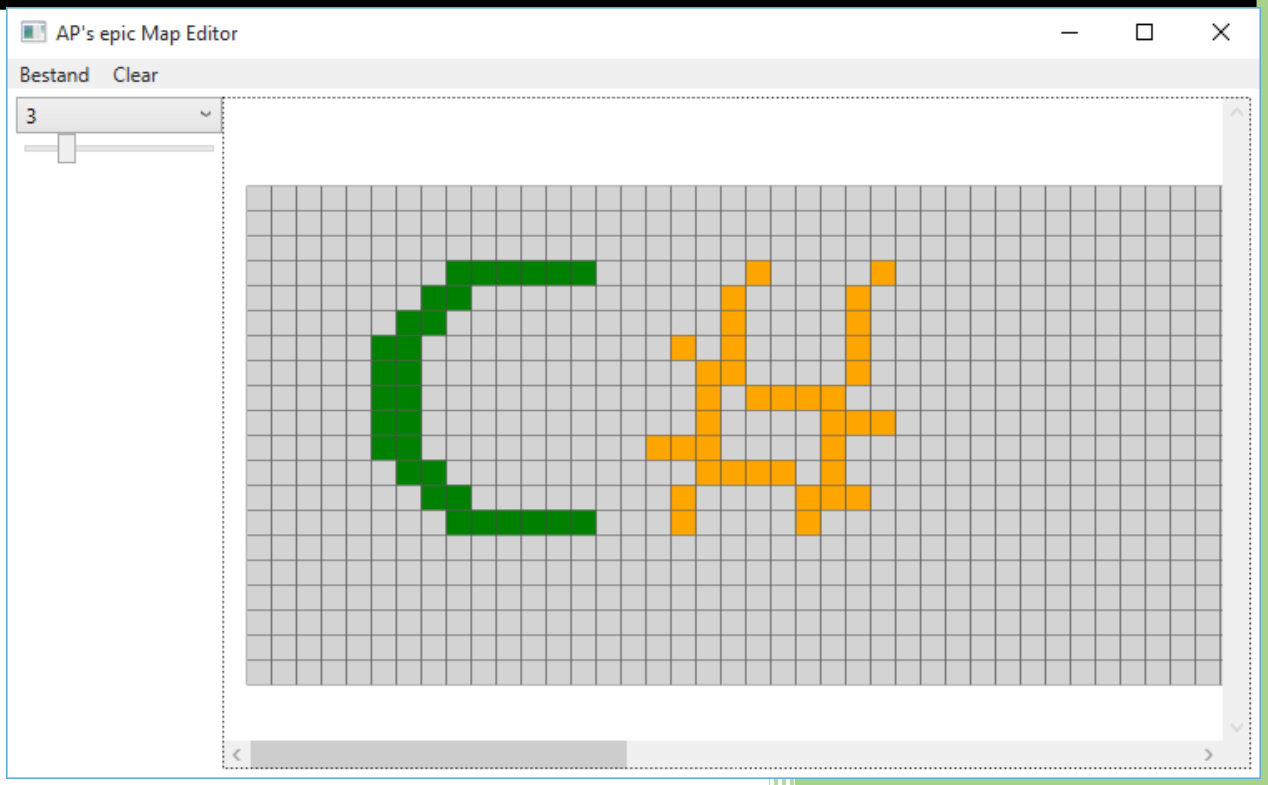
2015

C# Mapeditor



Tim Dams

AP

1-1-2015

Inhoud

Wat vooraf ging…

In de cursus GameDevelopment wordt volgende stuk beschreven ivm hoe je best je level opbouw bewaard mbv een 2D array van getallen. Hierbij stelt ieder getal een specifiek element voor (bijvoorbeeld 1= vernietigbaar blok, 2=vast blok, 3= monster Type 1, 4=monster Type 2, etc):

> *Onderstaande array bepaalt de plaats van een object: de array is 50x20 dimensies groot, en dit alles doe je maal 30, omdat elk object een veelvoud van 30 pixels is. Dus met andere woorden heb ik een level opgebouwd dat 50x30 = 1500 pixels breed en 20x30=600 pixels hoog. Onderstaande array fungeert als plattegrond van je level.*

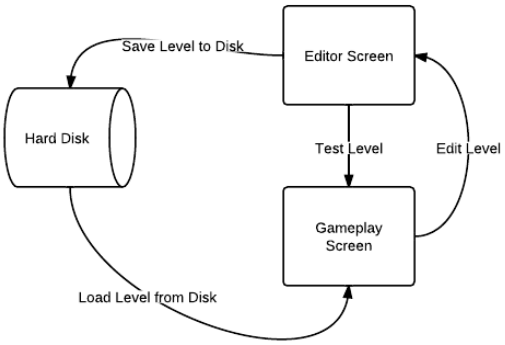
```
public byte[,] intTileArray = new byte[,] // elk getal > 0 stelt een bepaald blokje voor
{
    {0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,4,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0,0,0,2,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0,0,0,3,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {0,2,2,0,0,0,0,2,2,2,2,0,0,0,0,0,1,0,0,2,2,2,2,2,2,2,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
};
```

Via de volgende workshop zullen we ervoor zorgen dat de 2D-dimensionele array, die ons spel voorstelt, niet meer in code moet geschreven worden maar als aparte file kan ingeladen worden.

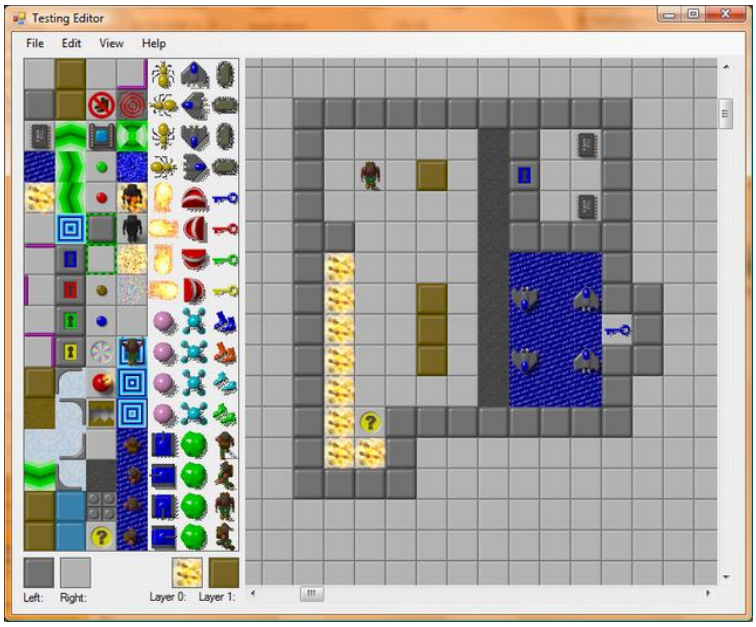
Het begin van een mapeditor

Concept

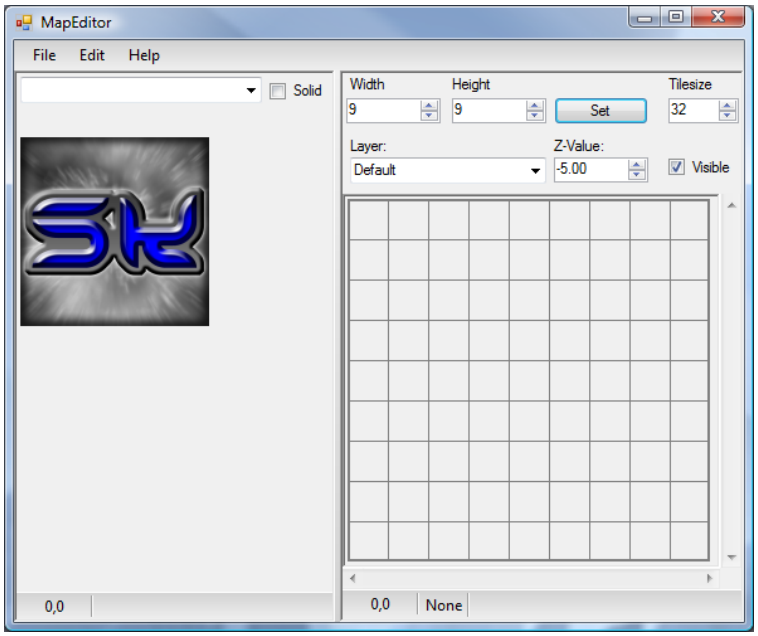
De ideale mapeditor voor een 2D-spel is eentje die ons toelaat om snel mappen te maken, zonder dat we daarvoor C# van het spel zelf moeten maken (denk bijvoorbeeld aan Tiled, één van de beste opensource editor). De mappen worden met andere woorden niet in de code zelf aangemaakt, maar op externe bestanden.



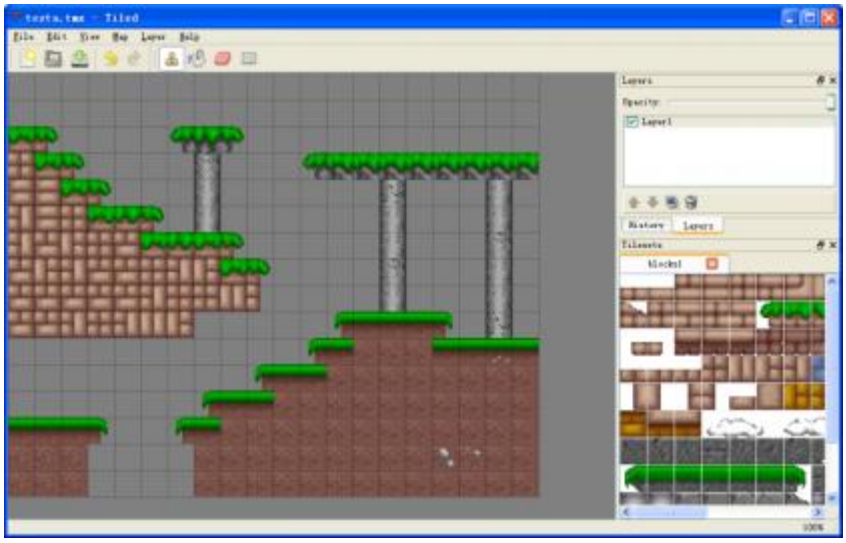
Bestaande voorbeelden van mapeditor



http://www.codeproject.com/Articles/26044/Tile-Editor-Control



http://www.codeproject.com/Articles/26166/2D-Map-Editor

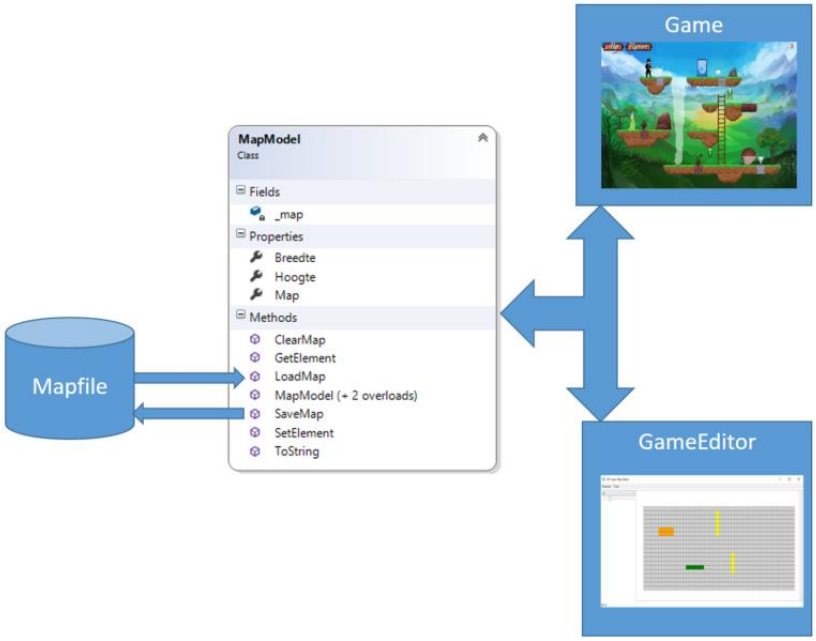


http://www.mapeditor.org/

Doel

We zullen een mapeditor maken in WPF. We gaan hierbij zo object georiënteerd mogelijk werken en zullen de “MapModel”klasse ontkoppelen van het grafisch gedeelte.

Door dit te bewerkstelligen kunnen we de MapModel klasse hergebruiken in het spel zelf, zonder dat hierbij nog code moet aangepast worden. MapModel is als het ware een bibliotheek.



Deel 1: MapModel

De MapModel klasse is het hart van ons project. Het is een klasse die een interne voorstelling van de kaart zal hebben en die enerzijds zal toelaten dat kaarten kunnen bewaard en ingeladen worden, en anderzijds opdat de GameEngine en/of MapEditor de kaart kan gebruiken en aanpassen.

Basis MapModel

We maken een nieuwe klasse MapModel die een kaart voorstelt in zijn rudimentairste vorm: een 2D-arrys van bytes (bytes zodat we minder plek nodig hebben per element. Nadeel is wel dat iedere element maar 255 mogelijke waarden kan hebben):

```
private byte[,] _map;
```

We willen dat de buitenwereld deze kaart enkel kan uitlezen, en maken dus een read-only property:

```
public byte[,] Map
{
    get
    {
        if (_map != null)
            return _map;
        throw new NullReferenceException("Map not created");
    }

}
```

[We gooien de NullReferenceException op voor het geval de gebruiker (programmeur) van deze klasse is vergeten een kaart ook effectief aan te maken (zie verder)]

Om de buitenwereld nog te helpen voegen we nog enkele handige properties toe die de Hoogte en Breedte van een map teruggeven:

```
public int Hoogte
{
    get { return _map.GetLength(0); }
}
public int Breedte
{
    get { return _map.GetLength(1); }
}
```

Voorts maken we 2 methoden opdat de buitenwereld individuele elementen op de kaart kan uitlezen en aanpassen:

```
public void SetElement(int x, int y, int value)
{
    _map[y, x] = (byte)value; //Todo check if valid x, y value
}
public int GetElement(int x, int y)
{
    return Convert.ToInt32(_map[y, x]); //Todo check if valid x, y value
}
```

Alsook een methode om de volledige kaart leeg te maken:

```
public void ClearMap()
{
```

```
            for (int i = 0; i < Hoogte; i++)
            {
                for (int j = 0; j < Breedte; j++)
                {
                    _map[i, j] = 0;
                }
            }
        }
```

Constructors

We voorzien 3 constructors, opdat de gebruiker op verschillende manieren een nieuwe kaart kan aanmaken:

1. Door een breedte en hoogte in te geven
2. Door een bestaande 2D bytearray in te geven
3. Door de locatie van een mapfile in te geven die dan zal ingeladen worden

```
        public MapModel(int breedte, int hoogte)
        {
            _map = new byte[hoogte, breedte];
        }
        public MapModel(byte[,] map)
        {
            _map = map;
        }
        public MapModel(string path)
        {
            this.LoadMap(path); //Komt later
        }
```

[De LoadMap()-methode ontbreekt nog maar dat lossen we zo meteen in het volgende hoofdstuk op.]

Deel 2: MapModel IO

Alles staat of valt uiteraard met het correct in en uitladen van een tekstbestand waarin we de kaart zullen beschrijven in een voor normale stervelingen leesbare tekst.

Opbouw file

We kiezen als bestandinhoud voor volgende layout:

- Breedte map = lengte van iedere lijn
- Hoogte map = aantal lijnen
- Data per lijn, gescheiden met een ko[1]mma

Een 5 bij 3 map waarbij er in het midden een blokje (1) staat zouden we als volgt naar een bestand wegschrijven:

5

3

0,0,0,0,0

0,0,1,0,0

0,0,0,0,0

Load en Save

We gaan 2 methoden toevoegen aan ons MapModel:

- LoadMap: inladen van map uit bestand
- SaveMap: bewaren van map naar bestand

We gaan het MapModel niet zelf de locatie van de file laten bewaren, dit omdat ik daar geen zin in heb ☺ Beide methoden moeten dus aangeroepen van buitenuit met als extra argument een string die het path bevat van de te gebruiken file:

```
public void LoadMap(string path){…}
public void SaveMap(string path){…}
```

Wanneer we data naar een bestand wegschrijven is het belangrijk dat je steeds de Laad en Bewaar methoden “synchroon” houdt, dit maakt het programmeren ervan een pak eenvoudiger.

Beide methoden moeten namelijk de data in het bestand op dezeldfe manier gebruiken. Als bijvoorbeeld de LoadMap methode de eerste lijn in het bestand als Breedte gebruikt, maar de SaveMap bewaard de Hoogte in de eerste lijn…dan is er een synchronisatieprobleem.

[1] We kiezen er bewust voor om ieder element met een komma te scheiden. Dit laat ons toe om getallen groter dan 9 toe te laten als waarde (daar we met bytes werken kunnen we dus eender welk getal tussen 0 en 255 gebruiken).

SaveMap

Het wegschrijven van de map naar een file zal uit 4 delen bestaan:

1. Steamwriter naar file openen
2. Breedte en hoogte wegschrijven
3. 2D array wegschrijven
4. Streamwriter sluiten

Laten we dit doen:

Streamwriter naar file openen:

```
StreamWriter writer = new StreamWriter(path);
```

Breedte en hoogte wegschrijven:

```
writer.WriteLine(Breedte);
writer.WriteLine(Hoogte);
```

2DArray wegschrijven:

```
for (int i = 0; i < Hoogte; i++)
{
    for (int j = 0; j < Breedte; j++)
    {
        writer.Write(_map[i,j]);
        if (j < Breedte - 1) //Geen komma op einde van een lijn
            writer.Write(",");
    }
    writer.WriteLine();
}
```

Streamwriter sluiten

```
writer.Close();
```

LoadMap

De LoadMap-methode zal dezelfde 4 delen bevatten , daar deze 'synchroon' moet werken zoals de SaveMap methode:

1. Steam**reader** naar file openen
2. Breedte en hoogte **uitlezen**
3. 2D array **uitlezen**
4. Stream**reader** sluiten

Uiteraard zullen we bij het uitlezen van data gebruik moeten maken van de Convert-bibliotheek zodat we de data naar de juiste vorm kunnen uitlezen[2].

Steamreader naar file openen:

```
StreamReader reader = new StreamReader(path);
```

Breedte en hoogte uitlezen:

```
int breedte = Convert.ToInt32(reader.ReadLine());
int hoogte = Convert.ToInt32(reader.ReadLine());
```

2D array uitlezen:

```
for (int i = 0; i < hoogte; i++)
{
    //lees lijn per lijn
    var lijn = reader.ReadLine();
    //Splits op komma's
    var gesplitst = lijn.Split(',');
    for (int j = 0; j < gesplitst.Length; j++)
    {
        resultaat[i, j] = (byte)Convert.ToInt32(gesplitst[j]);
    }

}
_map = resultaat;
```

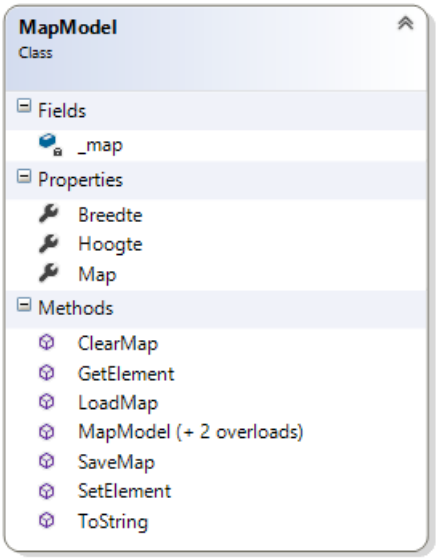
Steamreader sluiten

```
reader.Close();
```

[2] We laten het aan de lezer over om de nodige foutcontrole in te bouwen bij het uitlezen van foutieve informatie en deze vervolgens alsnog trachten te converteren.

Conclusie

De MapModel klasse is reeds klaar en kan integraal gebruikt worden in andere projecten:



We hebben er bewust voor gekozen om zo weinig mogelijk publieke methoden en properties aan te bieden, wat de bruikbaarheid van onze klasse, hopelijk verhoogd.

Je zou deze klasse nu reeds kunnen testen in een Console applicatie. Een voorbeeld gebruik:

```
//Maken nieuwe kaart, 5 breed, 3 hoog
MapModel mijnMap = new MapModel(5, 3);
//Element met coordinaten 2,2 op 6 zetten
mijnMap.SetElement(2, 2, 6);
//Kaart wegschrijven
mijnMap.SaveMap("testje.map");


//Kaart opnieuw inladen
MapModel mijnandereMap = new MapModel("testje.map");
//Waarde van element 2,2 uitlezen
int waarde = mijnandereMap.GetElement(2, 2);
//Waarde op scherm zetten, hopelijk komt er 6
Console.WriteLine(waarde.ToString());
```

Tekst in testje.map:

```
5
3
0,0,0,0,0
0,0,0,0,0
0,0,6,0,0
```

Deel 3: WPF Mapeditor

De WPF MAPEditor UI bestaat uit 3 delen:

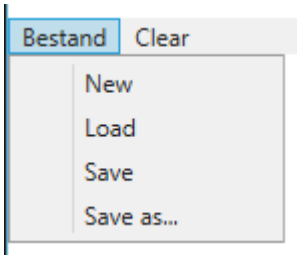
1. Een **Menu** bovenaan die we gebruiken om kaarten in te laden, bewaren, leegmaken, etc
2. Een soort **Toolbox** links waar we alle zaken zetten om de kaart mee te manipuleren.
3. De **Canvas** rechts waar de eigenlijke map zal getoond worden

Menu [XAML]

We definiëren in het hoofdgrid 2 rijen. De bovenste rij zal het Menu bevatten en zetten we dus op Auto. Alle overige ruimte is voor de Toolbox en het Canvas:

```
<Grid.RowDefinitions>
    <RowDefinition Height="auto"></RowDefinition>
    <RowDefinition></RowDefinition>
</Grid.RowDefinitions>
```

Het menu bestaat uit een handvol knoppen met bijhorende acties:




```
<Menu Grid.Row="0">
    <MenuItem Header="Bestand">
        <MenuItem Header="New" Name="menuNew" Click="menuNew_Click"></MenuItem>
        <MenuItem Header="Load" Name="menuLoad" Click="menuLoad_Click"></MenuItem>
        <MenuItem Header="Save" Name="menuSave" Click="menuSave_Click"></MenuItem>
        <MenuItem Header="Save as..." Name="menuSaveAs" Click="menuSaveAs_Click"></MenuItem>
    </MenuItem>
    <MenuItem Header="Clear" Name="menuClear" Click="menuClear_Click"></MenuItem>
</Menu>
```

Menu acties voor IO [C#]

We willen overal in de applicatie aan de kaart (currentMap) en z'n filelocatie (currentMapPath) geraken en defenieren daarom in de MainWindow klasse volgende 2 private fields:

```
private MapModel currentMap;
private string currentMapPath="";
```

We kunnen nu achter de verschillende menuitems de nodige FileDialogs plaatsen:

New

Klikken op New behandelen we verderop.

Load:

We vragen aan de gebruiker welke file moet geopend worden en slaan de locatie vervolgens op en laden de kaart in.

```
OpenFileDialog dialog = new OpenFileDialog();
if (dialog.ShowDialog() == true)
{
    currentMapPath = dialog.FileName;
    currentMap = new MapModel(currentMapPath);

    LoadMapOnView();
}
```

[De LoadMapOnView method zal de kaart op het canvas tekenen, wat verderop zal uitgelegd worden]

Save:

We kunnen enkel Save gebruiken indien er reeds een file effectief gekozen is[3]:

```
if (currentMapPath == "" )
{
    SaveFileDialog dialog = new SaveFileDialog();
    if (dialog.ShowDialog() == true)
    {
        currentMapPath = dialog.FileName;
        currentMap.SaveMap(currentMapPath);

    }
}
```

Save As:

Deze is quasi identiek aan Save, alleen moeten we niet controleren of de file reeds bestaat daar de gebruiker dit sowieso zal doen[4]:

```
SaveFileDialog dialog = new SaveFileDialog();
if (dialog.ShowDialog() == true)
{
    currentMapPath = dialog.FileName;
    currentMap.SaveMap(currentMapPath);
}
```

[3] Tip: je kan de UX verbeteren door deze knop pas actief te zetten wanneer hij mag gebruikt worden.
[4] We zouden dus eigenlijk deze knop ook gewoon de Save –code kunnen laten uitvoeren, wat weer wat code uitspaart.

Menu acties voor Map [C#]

We voegen een “Clear” actie toe aan het menu dat onze huidige kaart zal leegmaken. We gebruiken hiervoor de Clear-methode van MapModel en zorgen enkel ervoor dat de gebruiker de nodige waarschuwingen krijgt te zien en dus kan annuleren indien gewenst:

```
            if (MessageBox.Show("Dit zal uw huidige kaart volledig reseten.Bent u zeker?", "OPGELET",
MessageBoxButton.YesNo, MessageBoxImage.Warning) == MessageBoxResult.Yes)
            {
                currentMap.ClearMap();
                LoadMapOnView();
            }
```

Canvas [XAML]

We voegen een Canvas toe aan de applicatie (kies zelf waar je ze zet)

```
<ScrollViewer HorizontalScrollBarVisibility="Auto">
    <Canvas Name="mapCanvas"  MouseLeftButtonUp="mapCanvas_MouseLeftButtonUp" ></Canvas>
</ScrollViewer>
```

We plaatsen rondom de Canvas control een ScrollViewercontrol die zal toelaten om scrollbars te tonen indien onze kaart niet volledig op het zichtbare gedeelte van het canvas past.

Het MouseLeftButtonUp event zullen we verderop gebruiken.

Canvas Tekenen [C#]

We definiëren de pixelgrootte dat ieder blokje (element) op de kaart moet hebben als een field dat de hele klasse kan zien. Dit zal ons toelaten om later de grootte te veranderen:

```
private int blokscale = 15;
```

Iedere keer dat de kaart getekend moet worden roepen we de LoadMapOnView()methode aan, die eerst controleert of er wel een map is ingeladen:

```
        private void LoadMapOnView()
        {
            if (currentMap != null)
            {
                    //Tekencode
            }
        }
```

Het tekenen gaat als volgt:

1° Huidige Canvas clearen:

```
mapCanvas.Children.Clear();
```

2° Vervolgens stellen we in hoe groot het Canvas moet zijn (doen we dit niet dan zullen de scrollbars niet verschijnen). We rekenen dit uit door het aantal blokjes in de breedte (of hoogte) vermenigvuldigen met de grootte van 1 blokje (blokscale). We doen er voor alle zekerheid nog 10 pixels bij zodat de blokjes niet tegen de rand van de kaart plakken:

```
mapCanvas.Width = (currentMap.Breedte * blokscale) + 10;
mapCanvas.Height = (currentMap.Hoogte * blokscale) + 10;
```

3° We gaan nu de map doorlopen, lijn er lijn, van boven naar beneden, van links naar rechts:

```
for (int i = 0; i < currentMap.Hoogte; i++)
{
    for (int j = 0; j < currentMap.Breedte; j++)
    {
```

4° Per element op de kaart maken we nu klein rechthoekje aan met de grootte die is ingesteld in blokscale.

```
Rectangle blok = new Rectangle();
blok.Stroke = new SolidColorBrush(Colors.Black);
blok.StrokeThickness = 0.3;
blok.Width = blokscale;
blok.Height = blokscale;
```

5° Afhankelijk van de waarde dat het element heeft krijgt hij een ander kleurtje (vul zelf aan):

```
switch (currentMap.GetElement(j, i))
{
    case 0:
        blok.Fill = new SolidColorBrush(Colors.LightGray);
        break;
    case 1:
        blok.Fill = new SolidColorBrush(Colors.Red);
        break;
    case 2:
        blok.Fill = new SolidColorBrush(Colors.Green);
        break;
    default:
        blok.Fill = new SolidColorBrush(Colors.Black);
        break;

}
```

6° We berekenen dan de positie op het canvas waar het rechthoekje moet getekend worden:

```
blok.SetValue(Canvas.LeftProperty, (double)(blokscale * (j + 1)));
blok.SetValue(Canvas.TopProperty, (double)(blokscale * (i + 1)));
```

7° En last but not least voegen we het blokje aan het Canvas toe:

```
            mapCanvas.Children.Add(blok);
        }
    }
```

Slider om te zoomen

In en uitzoomen van het canvas met een slider is nu kinderspel dankzij de blokscale variabele.

Eerst voegen we ergens in de Toolbox een Slider toe:

```
<Slider Name="slidesScale" ValueChanged="slidesScale_ValueChanged" Value="15" Maximum="50" Minimum="5" IsSnapToTickEnabled="True" TickFrequency="2"  ></Slider>
```

We stellen deze in dat deze tussen 5 en 50 kan staan. We stellen een integer TickFrequency van 2 in alsook zetten we IsSnapToTickEnabled aan: dit zorgt ervoor dat de gebruiker enkel gehele waarden zal kunnen kiezen en we dus ook blokscale als int kunnen blijven gebruiken.

De achterliggende ValueChanged eventhandler zal vervolgens onze kaart hertekenen telkens de slidervalue aanpast:

```
private void slidesScale_ValueChanged(object sender, RoutedPropertyChangedEventArgs<double> e)
{
    blokscale = (int)e.NewValue;
    LoadMapOnView();
}
```

Blokjes in kleuren

Telkens de gebruiker op een blokje op het canvas klikt en muis loslaat willen we dat blokje de waarde geven die op dat moment 'actief' is.

De gebruiker selecteert daarvoor eerst in een combobox[5] de waarde die hij wilt gebruiken.

We plaatsen in de toolbox een Combobox:

```
<ComboBox Name="cmbBrush">
    <ComboBox.Items>
        <ComboBoxItem>1</ComboBoxItem>
        <ComboBoxItem>2</ComboBoxItem>
        <ComboBoxItem>3</ComboBoxItem>
        <ComboBoxItem>4</ComboBoxItem>
    </ComboBox.Items>
</ComboBox>
```

Vervolgens definiëren we wat er moet gebeuren bij het MouseLeftButtonUp event van ons canvas:

```
private void mapCanvas_MouseLeftButtonUp(object sender, MouseButtonEventArgs e)
{

}
```

1° We controleren of er wel een combobox-selectie is gemaakt en lezen deze uit:

```
if (cmbBrush.SelectedIndex > -1)
{
    var t = (cmbBrush.SelectedItem as ComboBoxItem).Content.ToString();
```

2° We halen dan de huidige muispositie terug, maar willen deze ten opzichte van het canvas. Als dus de gebruiker in de linkerbovenhoek van het canvas de muis loslaat zouden click.X en click.Y beiden 0 zijn.

```
Point click = e.MouseDevice.GetPosition(mapCanvas);
```

3° We berekenen dan welk binnen welk blokje de muis stond:

```
int x = (int)((click.X / blokscale)) - 1;
int y = (int)((click.Y / blokscale)) - 1;
```

Dit is uiteraard dezelfde , omgevormde formule die we gebruikten in stap 6[6] om de kaart te tekenen.

4° We kunnen nu aan ons MapModel vragen om het blokje op locatie (x,y) de nieuwe waar t te geven

```
currentMap.SetElement(x, y, Convert.ToInt32(t));
```

5° We hertekenen ineens de kaart terug zodat de gebruiker ook effectief de veranderen ziet:

```
LoadMapOnView();
```

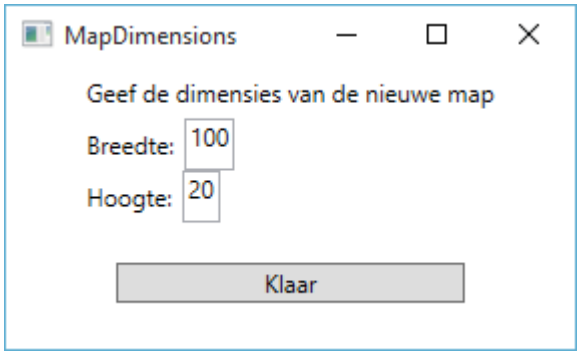
---

[5] Dit kan uiteraard ook door andere controls verzorgd worden (Textbox, slider, ToggleButtons, Buttons, Radiobuttons). Kies zelf wat je een goede UX vindt.

[6] `blok.SetValue(Canvas.LeftProperty, (double)(blokscale * (j + 1)));`

Deel 4: “Nieuwe kaart”-scherm

Wanneer we een nieuwe kaart willen maken en via het Menu op New klikken willen we eerst aan de gebruiker vragen wat de dimensies van de nieuwe kaart moeten zijn:



We vragen dit via een apart scherm dat we moeten maken.

- Rechterklik daarom op je project en kies voor Add -> New Item -> Window
- Geef het nieuwe bestand de naam “**MapDimensions**”.

De layout is redelijk eenvoudig:

```
<StackPanel Orientation="Vertical" HorizontalAlignment="Center">
    <Label>Geef de dimensies van de nieuwe map</Label>
    <StackPanel Orientation="Horizontal">
        <Label>Breedte: </Label>
        <TextBox Text="100" Name="txbBreedte"></TextBox>

    </StackPanel>
    <StackPanel Orientation="Horizontal">
        <Label>Hoogte:</Label>
        <TextBox Text="20" Name="txbHoogte"></TextBox>

    </StackPanel>
    <Button Name="btnKlaar" Margin="20" Click="Button_Click">Klaar</Button>
</StackPanel>
```

In de code-behind definieren we 2 properties waarin we de ingegeven hoogte en breedte zullen bewaren zodat het hoofdscherm deze nadien kan uitlezen.

```
public int Hoogte { get; set; }
public int Breedte { get; set; }
```

Het enige dat de klaar knop doet is ervoor zorgen dat de textbox waarden woorden uitgelezen en zal vervolgens het scherm sluiten:

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    //TODO: controleer ingegeven waarden
    Hoogte = Convert.ToInt32(txbHoogte.Text);
    Breedte = Convert.ToInt32(txbBreedte.Text);
    this.Close();
}
```

Nieuwe kaart-scherm tonen

In het hoofdscherm (MainWindow) voegen we nu de event hanlder toe voor de New-knop van de menu balk.

Deze zal het zonet aangemaakte nieuwe MapDimensions scherm tonen en wanneer deze is gesloten de Breedte en Hoogte uiltezen en gebruiken om een nieuwe kaart te maken:

```
private void menuNew_Click(object sender, RoutedEventArgs e)
{
    //TODO: check if current map needs to be saved
    MapDimensions askdims = new MapDimensions();
    askdims.ShowDialog();
    currentMap = new MapModel(askdims.Breedte, askdims.Hoogte);
    LoadMapOnView();
}
```

THE END!

De volledige codebron:

https://github.com/timdams/MapEditor_LearnSomeWPF