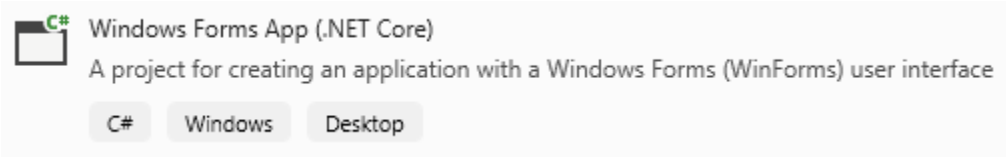
Deel GFX1 - Overzicht en oefeningen

Maak een nieuw project aan van het type "Windows Forms App (.Net Core) :

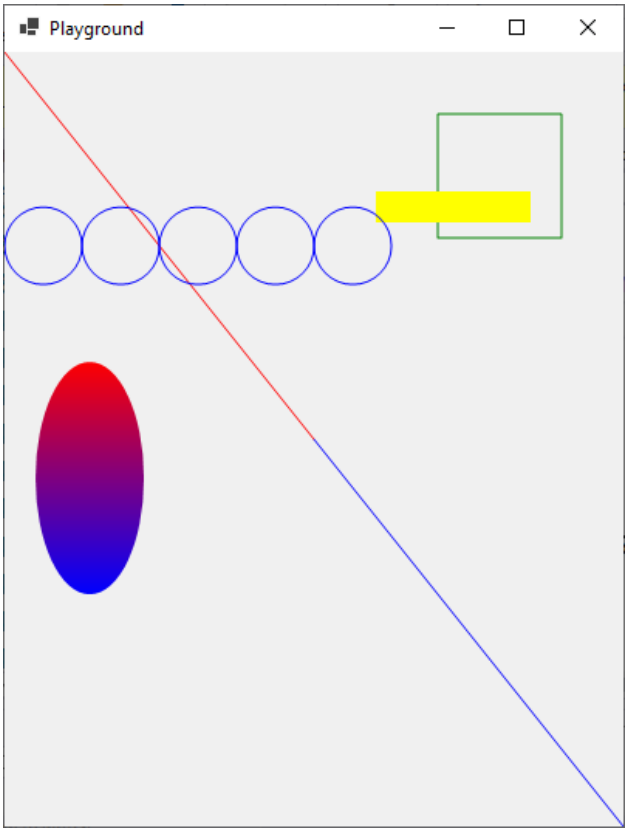


Let op, bij sommigen heet het "Windows Forms App (.NET)" of gewoon "Windows Forms App"!

In de Solution Explorer verwijder je nu de files voor Form1.cs en Program.cs, we zullen deze niet nodig hebben.

Unzip het bestand 'deel-gfx1-playground-code.zip' en stop de .cs files in het Playground project. De Program klasse bevat de Main method, stel deze in als het **Startup Object** van het project.

Start het programma, als alles correct is ingesteld krijg je onderstaande te zien :



Maak het venster eens kleiner/groter en let erop dat de figuren stil blijven staan maar dat de schuine lijn zich steeds aanpast.

Het programma bestaat uit 2 klassen :

Program
deze maakt het venster aan, voegt een user interface (UI) control toe en toont het venster

ViewControl
dit is het user-interface (UI) element waarop we gaan tekenen

Het eigenlijke tekenen gebeurt in de ViewControl.OnPaint() method, i.e. de OnPaint() method van de klasse ViewControl. Deze method wordt automatisch opgeroepen telkens de control moet hertekend worden (bv. bij het vergroten/verkleinen van het venster).

Voorlopig is voor jullie in method OnPaint(), enkel de code van belang die onder het commentaar "HIERONDER PLAATS JE JE EIGEN CODE OM TE TEKENEN" staat.

Je kunt daar via de 'gfx' variabele gebruik maken van een Graphics object dat een tekenoppervlak voorstelt. Je kunt de documentatie van deze klasse nalezen op

https://docs.microsoft.com/en-us/dotnet/api/system.drawing.graphics

Deze klasse bevat heel erg veel methods (en properties). Geen paniek echter, we zullen maar een klein stukje van de mogelijkheden gebruiken.

De tekenopdrachten die wij gebruiken zijn

- DrawLine
- DrawRectangle en FillRectangle
- DrawEllipse en FillEllipse

Om de opdrachten te kunnen oplossen zul je de documentatie van deze methods moeten opzoeken en/of wat experimenteren door de code te bewerken en te zien wat het resultaat is.

Merk op dat je steeds moet aangeven dat je klaar bent met een Pen of Brush object, door hun Dispose() method op te roepen! Op het einde van de onPaint method doen we dit als volgt :

```
redPen.Dispose();
greenPen.Dispose();
bluePen.Dispose();
yellowBrush.Dispose();
```

Oefening DGFX1.01

Alle tekenopdrachten gebruiken een plaatsbepaling op het tekenoppervlak die in x en y coordinaten uitgedrukt worden. Een beeldpunt (ook wel een pixel genoemd) kunnen we dus aanduiden met een koppel getallen die we bv. als (x,y) kunnen noteren.

Die x en y waarden zijn resp. de horizontale en verticale afstanden tot het punt (0,0) dat men de oorsprong noemt, **deze oorsprong bevindt zich linksboven in de hoek**.

Stel dat we een tekenoppervlak hebben van 200 pixels breed en 100 pixels hoog. Wat zijn dan de beeldpunt coordinaten - dus in (x,y) vorm - van de vier hoekpunten : linksboven, linksonder, rechtsboven en rechtsonder?

Oefening DGFX1.02

De DrawLine method tekent een lijn tussen 2 punten, bv. (x1, y1) en (x2, y2). Hoe worden de coordinaten van deze punten meegegeven aan de DrawLine oproep?

Oefening DGFX1.03

De DrawRectangle en FillRectangle methods verwachten bij hun oproep de nodige parameters om de rechthoek te definieren. Hoeveel parameters zijn er en wat stelt elke parameter voor?

Hoe kun je met deze methods vierkanten tekenen?

Oefening DGFX1.04

De DrawEllipse en FillEllipse methods verwachten bij hun oproep eveneens de nodige parameters om de te tekenen ovaal op te geven. Hoeveel parameters zijn er en wat stelt elke parameter voor?

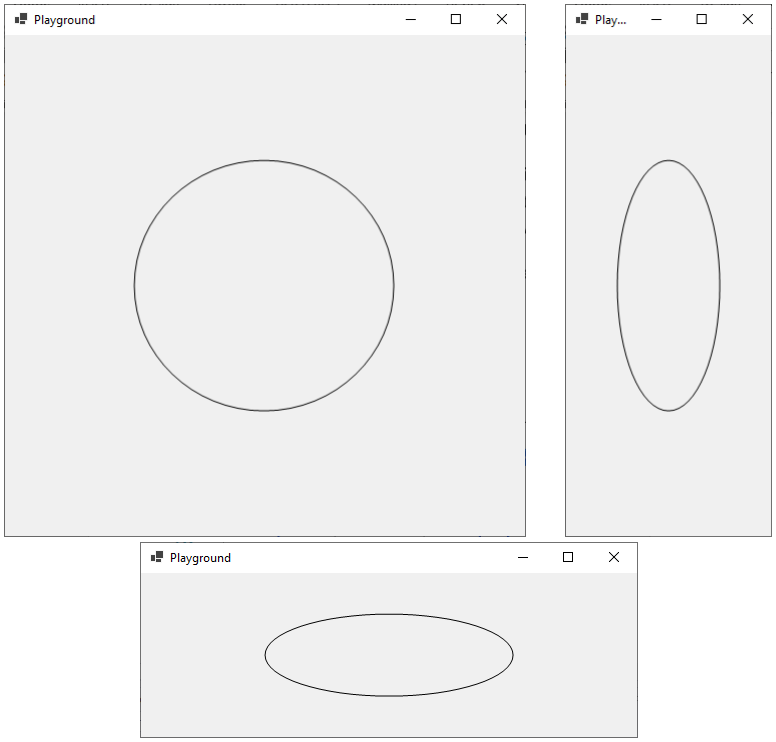
We zagen eerder dat als we het venster resizen, dat de figuren blijven staan maar dat de schuine roodblauwe lijn zich aanpast aan het nieuwe venster formaat.

Dit komt omdat de coordinaten van de lijn (2 lijnen eigenlijk) afhankelijk zijn van de afmetingen van het UI-element. De waarden 'this.Width' en 'this.Height' stellen resp. de breedte en de hoogte voor van de ViewControl.

Bij het tekenen van de overige figuren gebruiken we hardgecodeerde waarden, dit zijn waarden die letterlijk in de broncode staan, waardoor de figuren zich niet aanpassen aan de beschikbare ruimte.

Oefening DGFX1.05

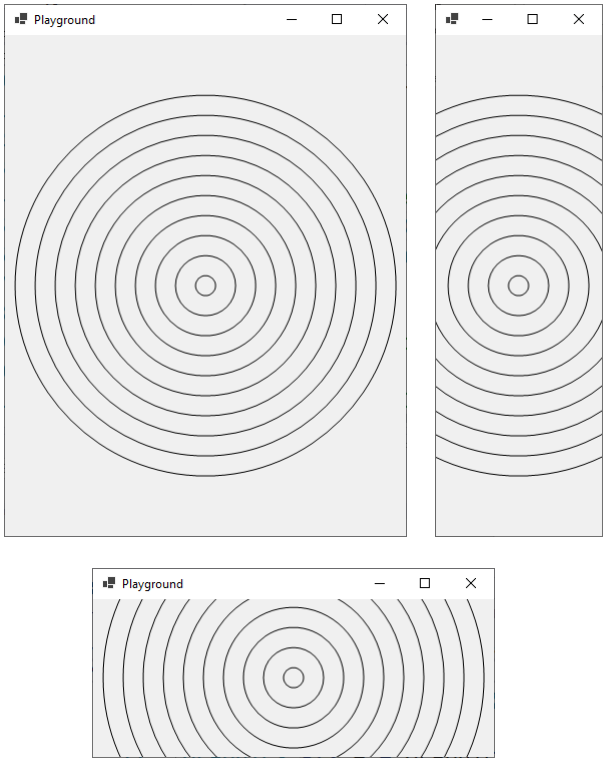
Pas de code aan zodat je volgende output krijgt :



De getekende ovaal past zich aan, aan de beschikbare ruimte. De breedte/hoogte van de ovaal is steeds de helft van breedte/hoogte van het tekenoppervlak. De ovaal is ook steeds gecentreerd in het venster.

Oefening DGFX1.06

Pas de code aan zodat je de volgende output krijgt

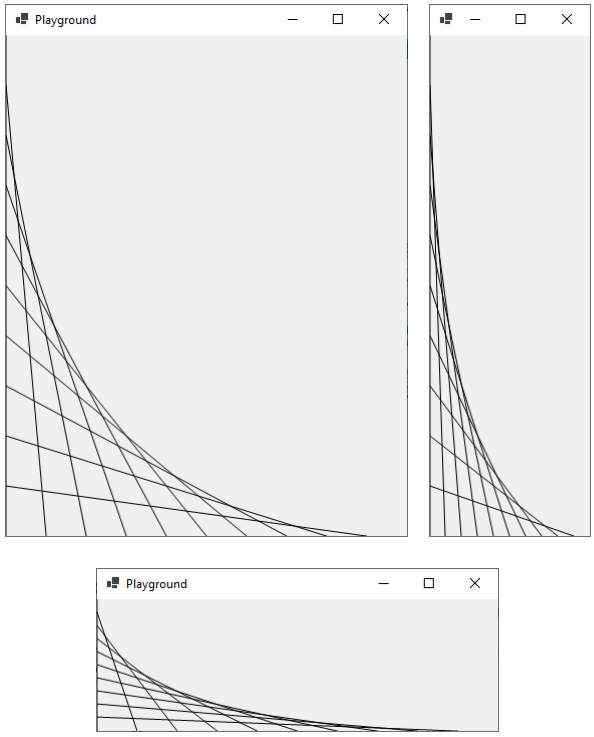


Dit zijn 10 cirkels wiens middelpunt netjes het middelpunt van het tekenoppervlak is. De kleinste cirkel heeft een straal van 10 pixels. De volgende 30pixel, dan 50pixels. De grootste cirkel heeft een straal van 210 pixels.

Merk op dat de cirkels steeds even groot zijn, ongeacht de afmetingen van het venster.

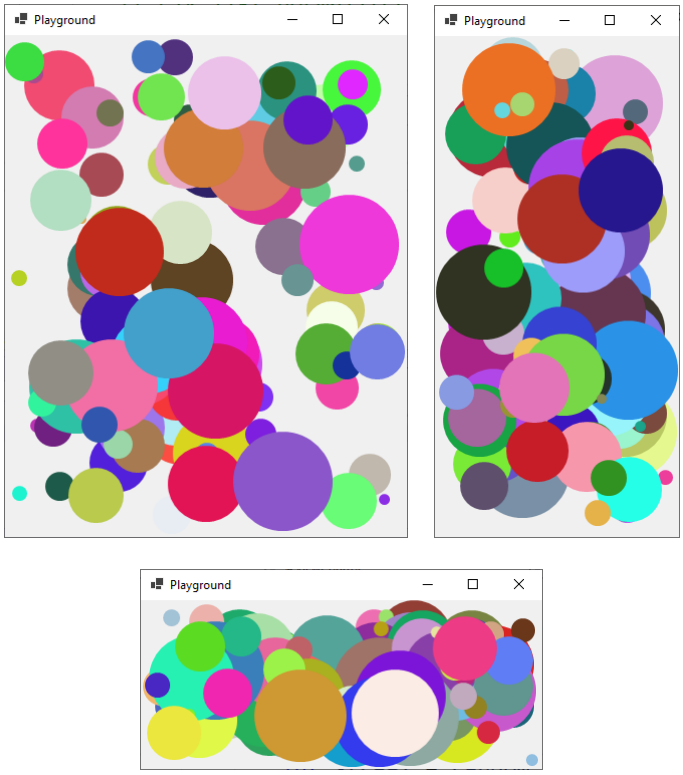
Oefening DGFX1.07

Pas de code aan zodat je de volgende output krijgt



Oefening DGFX1.08

Pas de code aan zodat je de volgende output krijgt



Telkens het venster hertekend moet worden, tekenen we 100 willekeurige cirkels. De straal is maximum 100 pixels groot. De cirkel ligt binnen de grenzen van het tekenoppervlak. De kleur is willekeurig gekozen door een Color object aan te maken met rood/groen/blauw waarden tussen 20 en 255 (grenzen inclusief).

Om willekeurige getallen aan te maken kun je de Random klasse gebruiken en herhaaldelijk de method Random.Next op te roepen. Bv.

```
Random random=new Random();
int getal1=random.Next(0, 100); // getal1 is >= 0 en < 100
int getal2=random.Next (0, 256) // getal2 is >= 0 en <= 255
…
```