Odisee Hogeschool
Gebroeders de Smetstraat 1, 9000 Gent

Advanced Applied Programming

Project 1:

Opdracht 3D

Steven Impens

Elektronica-Ict
2017-2018

Inhoudsopgave

Codefragmentenlijst

Figurenlijst

Inleiding

Het doel van dit project is het ontwerpen van een 3D spelbord waarbij een bal zich moet navigeren door een doolhof. De muren worden programmatisch toegevoegd. De bal kan navigeren aan de hand van interactie in de gui die het spelbord laten kantelen. Het is dus eerst de bedoeling een 3D geheel te ontwerpen waarna de controle van de 3D objecten en de animatie van een bal. Er wordt gebruik gemaakt van het WPF-raamwerk en C# in Visual studio. In het eerste deel is onderzocht wat WPF is en hoe een 3D scene is opgebouwd. In het tweede deel hoe alle nodige elementen worden geïmplementeerd. Nadien zullen fysica wetten toegepast worden zodanig dat de bal zich beweegt door de zwaartekracht door het kantelen en kan botsen met de muren.

1 Onderzoek 3D

1.1 Windows Presentation Foundation

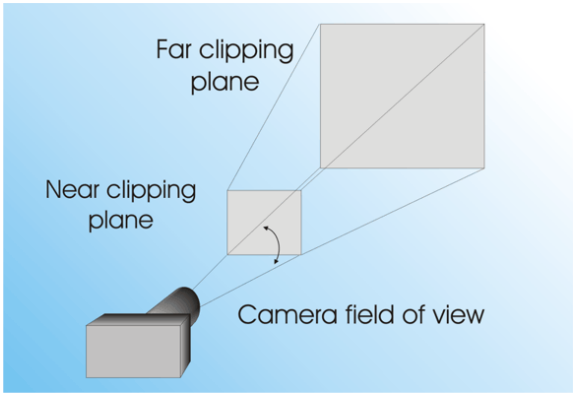
WPF kort voor Windows Presentation Foundation is een grafisch systeem voor het renderen van userinterfaces. WPF maakt gebruik van DirectX, een bundel van Application Programing interfaces. Deze APIs worden toegepast voor multimedia en games op het Windows platform. WPF heeft veel verschillende features waaronder Data Binding voor de interactie en manipulatie van data en Animaties voor het laten bewegen van objecten. Er wordt gebruik gemaakt van Extensible Application Markup Language, XAML. Hiermee is het mogelijk om delen van een programma te definiëren zonder code te schrijven maar deze methode is niet noodzakelijk sinds alles ook met code kan worden opgebouwd. [1] [2]

1.2 3D Scene in WPF

Een 3D scene in WPF heeft enkele onderdelen nodig. Een viewport waarin alle 3D objecten in bestaan, dit is een soort virtueel universum. Om de objecten die zich bevinden in een vierport te zien is er een camera en een lichtbron nodig. De camera kan enkel de zichtbare delen van opgelichte 3D objecten zien.

1.2.1 Camera

Een camera in WPF heeft een minimum en maximum bereik (Far clipping en Near clipping) alsook een Field of View (Figuur 1-1: Camera). Deze eigenschappen zorgen voor de zichtbaar ruimte. Indien een object uit deze zone ligt dan zal deze niet meer zichtbaar zijn. De camera heeft ook een positie in de viewport en een kijkrichting. Het is ook noodzakelijk om de bovenkant mee te geven zodat de camera weet welke kant de bovenkant is.



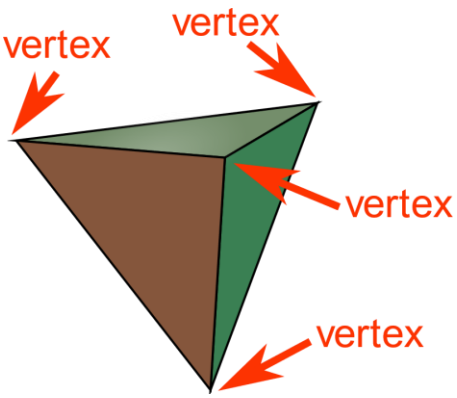
Figuur 1-1: Camera [3]

1.2.2 Licht

Lichtbronnen zorgen ervoor dat objecten worden opgelicht en dus zichtbaar zijn. Zonder licht is er ook geen reflectie en kan de camera niets opnemen. Licht is dus noodzakelijk voor een 3D scene. WPF bevat vier soorten lichtbronnen. AmbientLight zorgt voor een lichtinval op alle objecten ongeacht locatie of oriëntatie, alle vlakken worden opgelicht. In tegenstelling tot een DirectionalLight die geen locatie heeft maar wel een richting, alle objecten worden opgelicht vanuit een ingestelde directie. Er bestaan ook meer specifiek lichtbronnen, de PointLight en de SpotLight. De eerste is net zoals een lamp en zal dus rond de bron verlichten in een gespecifieerd bereik alsook vermindering van sterkte naargelang de afstand. De tweede is een gerichte lichtbron in de vorm van een kegel.

1.2.3 3D objecten

Alle 3D objecten worden beschreven door een verzameling van driehoeken. De software die de objecten weergeeft kan de kleur van elke driehoek berekenen aan de hand van hun materiaal en de lichtinval van alle lichtbronnen. De oppervlakte van een 3D object genaamd een mesh bestaat uit vertices. Dit zijn de punten waaruit de driehoeken worden gevormd. Figuur 1-2: Tetrahedron is een 3D object bestaande uit vier vertices die samen driehoeken vormen tot een geheel. Enkel de voorkant van een mesh wordt weergegeven. Om een driehoek weer te geven zijn er 3 coördinaten nodig en hun relatie met elkaar. Hun relatie moet in een tegenwijzers zin gedefinieerd worden anders zal de achterkant als voorkant gezien worden wat de driehoek dus niet zichtbaar maakt. Dit is vooral opvallend in volledige objecten. Door een verkeerd ingestelde driehoek zal er een gat ontstaan in het object. [4]



Figuur 1-2: Tetrahedron [5]

2 Uitwerking 3D

2.1 Controls

De interface wordt opgedeeld in twee delen namelijk een plaats voor de 3D scene waar het spelbord zich in bevindt en een plaats voor de bedieningselementen die zorgen voor het kantelen van het bord. Hiervoor wordt gebruik gemaakt van een Grid element. De interface kan zo ingedeeld worden als een tabel waarbij de hoogte van een rij wordt ingesteld.

Sinds dat het bord in 2 kanten moet draaien zullen er 2 elementen nodig zijn. Hiervoor zijn sliders een goede keuze. De sliders worden in het midden afgesteld zodat ze de waarde van nul graden hebben. Indien een slider naar links wordt verschoven zal een negatieve hoek ingesteld worden, rechts een positieve hoek. Een slider zal zorgen voor de rotatie rond de x-as terwijl de andere zorgt voor een rotatie rond de z-as. Beide sliders worden in een StackPanel gestoken, hierdoor kan hun positie makkelijk worden uitgelijnd. Het panel krijgt ook een rij nummer, namelijk 0 zodat dit gedeelte als eerste wordt getoond in de interface. De viewport zal de 3D scene bevatten met het spelbord.

```
<Grid>
    <Grid.RowDefinitions>
        <RowDefinition Height="*"></RowDefinition>
        <RowDefinition Height="15*"></RowDefinition>
    </Grid.RowDefinitions>
    <GridSplitter Grid.Row="0" VerticalAlignment="Top"></GridSplitter>

    <StackPanel Grid.Row="0" Orientation="Horizontal" HorizontalAlignment="Stretch">
        <Slider Width="500" Minimum="-90" Maximum="90" VerticalAlignment="Top"
                Margin="40,28,0,0" Name="Slider1"
                ValueChanged="Slider1_ValueChanged"/>
        <Slider Width="500" Minimum="-90" Maximum="90" VerticalAlignment="Top"
                Margin="40,28,0,0" Name="Slider2"
                ValueChanged="Slider2_ValueChanged"/>
    </StackPanel>


    <Viewport3D>
    </Viewport3D>
</Grid>
```

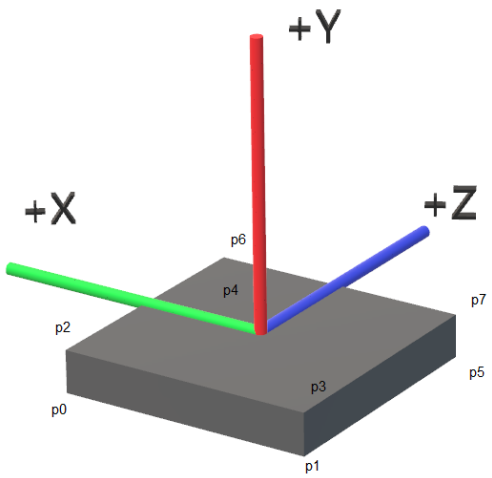
Codefragment 2-1: UI controls

Beide sliders krijgen een “ValueChanged” attribuut, als de slider wordt gebruikt zal de bijhorende functie aangeroepen worden. De functie zal dan de code uitvoeren om het bord te kantelen.

2.2 3D Scene

2.2.1 Spelbord

Het spelbord is een 3D object en heeft een Mesh nodig. Een ModelVisual3D zal content moeten bevatten namelijk een GeometryModel3D, hierin wordt niet enkel de geometrie van het bord in gedefinieerd maar ook het materiaal ervan. De Mesh van het spelbord is beschreven in het MeshGeometry3D element. Zoals beschreven in het onderdeel 1.2.3 3D objecten moet een mesh bestaan uit vertices en hun positie. Een bord met een lengte en breedte van 100 en een dikte van 2 is opgebouwd uit 8 vertices en 12 driehoeken.



Figuur 2-1: Coördinatensysteem

Punt 0 zal in de “Positions” attribuut op de eerste plaats komen en heeft als coördinaten (50,-2,-50), punt 1 met (-50,-2,50) op de tweede plaats gescheiden door een spatie. Om het midden van het bord op het nulpunt te krijgen zal het bord spreiden van 50 tot -50 op de x- en z-as. De gebruikte positie verdeling is weergegeven op Figuur 2-1: Coördinatensysteem.

Na het toevoegen van de punten moeten ze verbonden worden voor het maken van driehoeken. De driehoek van punt 0, 1 en 3 en die van punt 3, 2, 0 vormen samen 1 vlak. Deze punten moeten tegenwijzers zin toegevoegd worden anders kan het zijn dat een deel van het bord niet juist weergegeven wordt en er dus een opening zal ontstaan op de fout ingeven combinaties. De combinaties worden toegevoegd aan de “TriangleIndices” attribuut.

Sinds dat WPF automatisch het voorste vlak kiest kan het zijn dat de belichting niet correct op het bord neerkomt dit kan verholpen worden door de “Normals” attribuut. Door elk punt een vector te geven van (0,1,0) hierdoor richt elk vlak naar boven waar er een lichtbron is.

Het is nu nog nodig om een materiaal toe te voegen. Om een afbeelding als textuur te gebruiken is het nodig om “TextureCoordinates” in te geven. Het bovenste vlak moet de textuur krijgen dus de 4 bovenste punten moeten overeenkomen met de coördinaten voor de afbeelding. Punt 2 krijgt een coördinaat van (0,0), punt 3 (0,1), punt 6 (1,0) en punt 7 (1,1). De andere punten krijgen (0,0). Er is alsook een “BackMaterial” aanwezig om ervoor te zorgen dat de onderkant van het bord niet doorzichtig is.

Alles bij elkaar krijgen we een 3D object die behoord tot de content van een “ModelVisual3D”.

```
<GeometryModel3D>
	<GeometryModel3D.Geometry>
		<MeshGeometry3D Positions="50,-2,-50 -50,-2,-50 50,0,-50 -50,0,-50 50,-
	2,50 -50,-2,50 50,0,50 -50,0,50"
		TriangleIndices="0,1,3 3,2,0 1,5,7 7,3,1 5,4,6 6,7,5 4,0,2 2,6,4 2,3,7
	7,6,2 4,0,1 1,5,4"
		Normals="0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0"
		TextureCoordinates="0,0 0,0 0,0 0,1 0,0 0,0 1,0 1,1"/>
	</GeometryModel3D.Geometry>
	<GeometryModel3D.Material>
		<DiffuseMaterial>
			<DiffuseMaterial.Brush>
				<ImageBrush ImageSource="images/wood1.jpg"/>
			</DiffuseMaterial.Brush>
		</DiffuseMaterial>
	</GeometryModel3D.Material>
	<GeometryModel3D.BackMaterial>
		<DiffuseMaterial Brush="Black"/>
	</GeometryModel3D.BackMaterial>
</GeometryModel3D>
```

Codefragment 2-2: spelbord in XAML

Het is ook mogelijk om de Positions en TriangleIndices via code toe te voegen aan de XAML. Er moet dan gebruik gemaakt worden van “Data Binding”. Dit is een feature van WPF waarbij er verwezen wordt naar een functie in de C#-code. De functie geeft dan de juiste waardes mee. Sinds dat het spelbord statisch is en niet moet veranderen is het handiger om deze feature niet te implementeren. In Codefragment 2-3: Binding is Binding toegepast waarbij FloorPoints3D en FloorPointsIndices de functies zijn.

```
<MeshGeometry3D x:Name="floorGeometry"
	Positions="{Binding FloorPoints3D, ElementName=Window}"
	TriangleIndices="{Binding FloorPointsIndices, ElementName=Window}"
	TextureCoordinates="0,0 1,0 1,1 0,1"
/>
```

Codefragment 2-3: Binding

2.2.2 Belichting

Alle 3D objecten moeten verlicht zijn om ze te kunnen zien. Om dit te verwezenlijk is gebruikt gemaakt van een “AmbientLight”. Die zorgt ervoor dat alles verlicht wordt ongeacht de positie en oriëntatie van het 3D object. Om dit licht niet te uitbundig te maken is een donkere kleur beter. Voor een meer gefocust licht op het bord te krijgen is een “DirectionalLight” van toepassing. Deze heeft een lichtere kleur zodat er duidelijk meer natuurlijk licht aanwezig is. Dit soort moet ook een richting krijgen namelijk (0,-1,-1). Hierdoor zal het licht schijnen naar beneden recht op het spelbord.

```
<ModelVisual3D x:Name="Light">
    <ModelVisual3D.Content>
        <Model3DGroup>
            <AmbientLight Color="#333333"/>
            <DirectionalLight Color="#EEEEEE" Direction="0 -1 0">
            </DirectionalLight>
        </Model3DGroup>
    </ModelVisual3D.Content>
</ModelVisual3D>
```

Codefragment 2-4: Lichtbronnen

2.2.3 Camera

De camera moet gericht zijn op het spelbord. Om dit te verwezenlijken krijgt de camera een positie boven en ver van het bord. Sinds dat het bord in het nulpunt ligt moet da camera in een negatief Z-coördinaat liggen (0, 120, -120) om in een positieve richting te kijken. Door deze coördinaten te gebruiken ligt het bord in het midden en volledig in de zichtbare regio. Het is noodzakelijk om een “UpDirection” mee te geven anders weet de camera niet wat boven en onder is.

```
<Viewport3D.Camera>
      <PerspectiveCamera x:Name="Camera"
                         FarPlaneDistance="300"
                         LookDirection="0,-1,1"
                         UpDirection="0,1,0"
                         NearPlaneDistance="1"
                         Position="0,120,-120"
                         FieldOfView="75">
      </PerspectiveCamera>
 </Viewport3D.Camera>
```

Codefragment 2-5: Camera

2.2.4 Muren

De muren van het spel worden dynamisch toegevoegd aan de 3D scene via C#-code. In de XAML word een “ContainerUIElement” toegevoegd die tevens ook een naam krijgt met de “x:Name” attribuut. Hierdoor kan de container aangesproken worden vanuit de code.

```
<ContainerUIElement3D x:Name="WallContainer" />
```

Sinds dat 3D objecten bestaan uit driehoeken is er een methode nodig die deze kan aanmaken. In de Triangle methode wordt een MeshGeometry3D aangemaakt van 3 punten, hun relatie en de Normals. De methode geeft een Model3DGroup terug met een GeometryModel3D van een driehoek en een materiaal.

```
public Model3DGroup Triangle(Point3D p0, Point3D p1, Point3D p2)
{
    MeshGeometry3D mesh = new MeshGeometry3D();
    mesh.Positions.Add(p0);
    mesh.Positions.Add(p1);
    mesh.Positions.Add(p2);
    mesh.TriangleIndices.Add(0);
    mesh.TriangleIndices.Add(1);
    mesh.TriangleIndices.Add(2);

    Vector3D normals = GetNormals(p0, p1, p2);
    mesh.Normals.Add(normals);
    mesh.Normals.Add(normals);
    mesh.Normals.Add(normals);

    GeometryModel3D triangle = new GeometryModel3D(mesh, new
           DiffuseMaterial(new SolidColorBrush(Colors.Red)));

    Model3DGroup group = new Model3DGroup();
    group.Children.Add(triangle);
    return group;
}
```

Codefragment 2-6: Aanmaken Driehoeken

De Cube methode zal de nodige driehoeken aanvragen en samenvoegen tot een geheel die de kubus vormt. Aan de hand van de meegegeven coördinaten en dimensies worden de nodige punten berekend en de driehoeken aangemaakt. Als alle driehoeken zijn toegevoegd zal er een ModelVisual3D teruggegeven worden die dan kan worden toegevoegd aan een viewport of de ContainerUIElement die voorzien is voor de muren.

```
public ModelVisual3D Cube(int x, int y, int z, double l, double h, double w)
{
    Point3D p0 = new Point3D(x + 0, y + 0, z + 0);
    Point3D p1 = new Point3D(x + w, y + 0, z + 0);
    Point3D p2 = new Point3D(x + 0, y + h, z + 0);
    Point3D p3 = new Point3D(x + w, y + h, z + 0);
    Point3D p4 = new Point3D(x + 0, y + 0, z + l);
    Point3D p5 = new Point3D(x + w, y + 0, z + l);
    Point3D p6 = new Point3D(x + 0, y + h, z + l);
    Point3D p7 = new Point3D(x + w, y + h, z + l);

    Model3DGroup cube = new Model3DGroup();

    //front
    cube.Children.Add(Triangle(p0, p2, p3));
    cube.Children.Add(Triangle(p3, p1, p0));
    //left
    cube.Children.Add(Triangle(p0, p4, p6));
    cube.Children.Add(Triangle(p6, p2, p0));
    //right
    cube.Children.Add(Triangle(p1, p3, p7));
    cube.Children.Add(Triangle(p7, p5, p1));
    //bottom
    cube.Children.Add(Triangle(p0, p4, p5));
    cube.Children.Add(Triangle(p5, p1, p0));
    //top
    cube.Children.Add(Triangle(p2, p6, p7));
    cube.Children.Add(Triangle(p7, p3, p2));
    //back
    cube.Children.Add(Triangle(p4, p5, p7));
    cube.Children.Add(Triangle(p7, p6, p4));

    return new ModelVisual3D{ Content = cube };


}
```

Codefragment 2-7: Aanmaken kubus

2.3 Kantelen Spelbord

De bijhorende functies van de sliders in de interface roepen een functie op met de posities van beide sliders. De functie “ChangeBoardRotation” maakt een transformatie aan die roteert rond 2 assen namelijk de x-as en de z-as. De transformatie wordt toegepast op het spelbord, de muren en de bal. Deze functie is zo geprogrammeerd dat de rotatie van een as niet reset tijdens het roteren op een andere as.

```
private void ChangeBoardRotation(double angleX, double angleZ)
{
    Transform3DGroup myTransform3DGroup = new Transform3DGroup();

    RotateTransform3D rotateTransform3D = new RotateTransform3D(new
           AxisAngleRotation3D(new Vector3D(0, 0, 1), angleZ));

    myTransform3DGroup.Children.Add(rotateTransform3D);

    rotateTransform3D = new RotateTransform3D(new AxisAngleRotation3D(new
           Vector3D(1, 0, 0), angleX));
    myTransform3DGroup.Children.Add(rotateTransform3D);

    Board.Transform = myTransform3DGroup;
    WallContainer.Transform = myTransform3DGroup;
    SphereContainer.Transform = myTransform3DGroup;



}
```

Codefragment 2-8: Bord rotatie

2.4 Simpele animatie van de bal

Animaties gebeuren aan de hand van storyboards. Een storyboard kan meerdere animaties bevatten. Een animatie is een verandering van een waarde over een gespatieerde tijd. Om de bal te laten rollen moeten er 2 waarden veranderen namelijk een hoek voor de rotatie en een afstand. Het aantal graden dat de bal draait hangt af van de afstand die hij aflegt. Als de bal een volledige rotatie maakt van 360 graden dan zal er een afstand van 2*pi*radius afgelegd worden.

```
private void SimpleBallAnimation()
{
    Transform3DGroup myTransform3DGroup = new Transform3DGroup();

    //rotation
    AxisAngleRotation3D axis = new AxisAngleRotation3D(
      new Vector3D(0, 0, 1), 0);

    RotateTransform3D rotate = new RotateTransform3D(axis);
    rotate.CenterX = -46;
    rotate.CenterY = 2;
    rotate.CenterZ = 46;

    myTransform3DGroup.Children.Add(rotate);

    SphereContainer.Transform = myTransform3DGroup;

    NameScope scope = new NameScope();
    FrameworkContentElement element = new FrameworkContentElement();
    NameScope.SetNameScope(element, scope);

    element.RegisterName("rotation", axis);

    DoubleAnimation animation = new DoubleAnimation();
    animation.From = 0;
    animation.To = -1548;
    animation.Duration = TimeSpan.FromSeconds(4);
    animation.AutoReverse = true;
    animation.RepeatBehavior = RepeatBehavior.Forever;

    Storyboard myStoryboard = new Storyboard();

    Storyboard.SetTargetProperty(animation, new PropertyPath("Angle"));

    Storyboard.SetTargetName(animation, "rotation");
    myStoryboard.Children.Add(animation);
    myStoryboard.Duration = TimeSpan.FromSeconds(4);
    myStoryboard.RepeatBehavior = RepeatBehavior.Forever;
    myStoryboard.AutoReverse = true;

    this.Resources.Add("id", myStoryboard);
    myStoryboard.Begin(element, HandoffBehavior.Compose);
}
```

Codefragment 2-9: Rotatie animatie van de bal

Sinds dat de bal niet in het nulpunt ligt is het zeker nodig zijn middelpunt coördinaten in te stellen anders zal de rotatie gebeuren tegenover de assen van de 3D scene en niet de assen van de bal zelf. Gedurende de translatie animatie updaten die coördinaten zelf. Hierdoor moeten ze niet constant meegegeven worden aan de animatie voor de rotatie.

3 Onderzoek game fysica

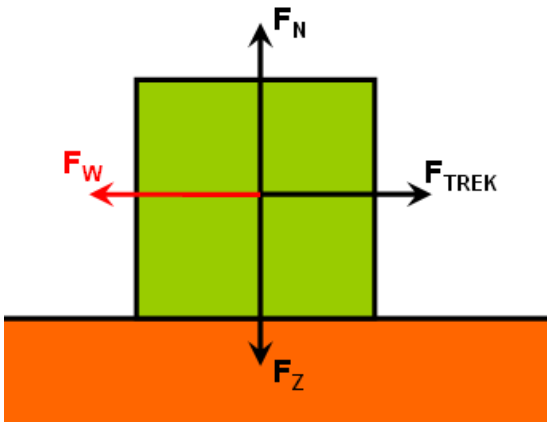
3.1 Game loop

De game loop zorgt ervoor dat om een aantal milliseconden de nodige fysica berekening worden uitgevoerd. Een vooringesteld timer kan de nodige methodes oproepen, deze implementatie wordt ook wel “Fixed Time Step” genoemd. Dit is zeer gemakkelijk te implementeren doordat de berekening stap per stap worden uitgevoerd. Trage computers hebben een groot impact op de snelheid waarbij de timer de berekeningen doorloopt waardoor het geheel trager zal werken. Bij snelle computers is het probleem dat de berekeningen snel gebeuren en de er moet gewacht worden op de nieuwe berekeningen. Een variant hierop is de “Variable Time Step” waarbij de code kan doorlopen worden aan de snelheid van de pc. Deze implementatie bij tragere systemen kan ervoor zorgen dat waardes veel te snel veranderen terwijl de Render loop niet kan volgen. Hierdoor kan het lijken alsof objecten verspringen en helemaal geen fysica wetten volgen. [6] [7]

3.2 Wrijving

Wrijving ontstaat bij de verschuiving van twee oppervlaktes. De wrijvingskracht F_W werkt in de tegengestelde richting waardoor er een negatieve versnelling voorkomt, een vertraging. Een object in beweging zal vertragen tot stilstand als er geen andere krachten aanwezig zijn. De grootte van de wrijving hangt af van de normaalkracht F_N en de wrijvingscoëfficiënt μ. De normaalkracht is de kracht loodrecht op de oppervlakte. Bij een voorwerp die recht op een oppervlakte steunt zal de normaalkracht gelijk zijn aan de zwaartekracht. De wrijvingscoëfficiënt hangt af van de materialen. [8]

$$F_W = \mu * F_N$$
$$F_n = m * a = m * 9.81 m/s^2$$



Figuur 3-1: Vectoriele voorstelling wrijving [8]

3.3 Snelheid en Versnelling

Een eenparige versnelling is de verandering in snelheid over een bepaalde tijdsduur. Een voorwerp valt door de valversnelling die gecreëerd wordt door de zwaartekracht. De snelheid kan berekend worden door de beginsnelheid op te tellen met de versnelling vermenigvuldigd met de tijd. Waarbij a de valversnelling is door de zwaartekracht.

$$v = v_0 + a * t$$

Met de versnelling en de snelheid van een voorwerp kan een afgelegde afstand berekend worden.

$$x = v_0 * t + \frac{1}{2} * a * t^2$$

3.4 Botsing

Een botsing ontstaat bij voorwerpen indien ze elkaar raken door hun beweging. Tussen de 2 objecten worden impuls en energie uitgewisseld. Er bestaan twee soorten botsingen, de elastische en de niet elastische. Als eerste kan een botsing deuken veroorzaken indien de kracht sterk genoeg is. Daarna kan door de veerkracht een restitutiestoot ontstaan die beide voorwerpen van elkaar duwt. Door een onelastische botsingen zullen beide voorwerpen verder bewegen als 1 geheel. De snelheid voor de botsingen is gelijk aan de snelheid na de botsing. Bij een elastische botsing zal de snelheid van richting veranderen alsook wisselen beide voorwerpen hun snelheid. Sinds dat een deel van de energie wordt gebruikt voor de vervorming zal niet de volledige snelheid uitgewisseld worden. [9] Hieruit volgt de formule waarbij e de restitutiecoëfficiënt is:

$$v_{1na} - v_{2na} = -e(v_{1voor} - v_{2voor})$$

4 Uitwerking game fysica

4.1 Game loop

Aan de hand van een timer worden de fysica berekening uitgevoerd. De timer zal om de aantal seconden een methode oproepen die nieuwe positie van de bal bepalen. De timer start de methode in een nieuwe thread waardoor het nodig is om een Dispatcher te gebruiken waardoor de berekeningen kunnen uitgevoerd worden op waardes vanuit de standaard thread. Het interval van de timer bevat het de tijdsduur van een frame in milliseconden. In zo frame wordt de nieuwe positie van de bal berekend. De berekeningen zullen dus dit interval gebruiken als verlopen tijd maar dan in seconden. [10]

```
private void StartGameTimer()
{
    frameInterval = 1 / (double)PhysicsCalPerSec;
    int frameIntervalMs = (int)(this.frameInterval * 1000);
    var timer = new System.Timers.Timer
    {
        Interval = frameInterval
    };
    timer.Elapsed += new ElapsedEventHandler(TimerElapsed);
    timer.Start();
}

private void TimerElapsed(object sender, EventArgs e)
{
    Dispatcher.Invoke((Action)delegate ()
    {
        Frame();
    });
}
```

Codefragment 4-1: Timer voor Game Loop

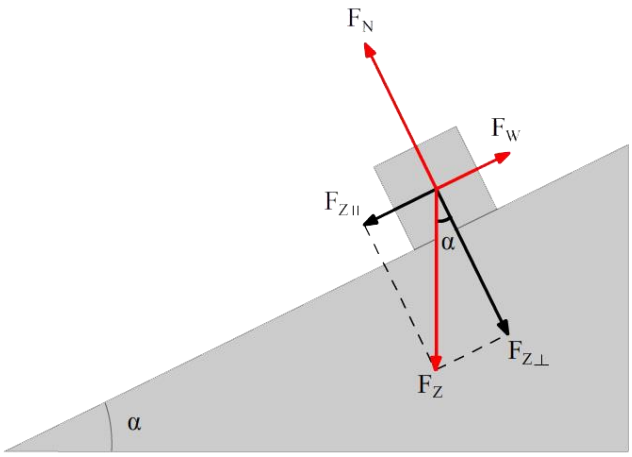
Een frame bevat de methodes die de afgelegde afstand meegeven aan de translatie van de bal. Indien er een object in de weg staat zal da afgelegde afstand op nul worden gezet waardoor de bal niet verder zal bewegen.

```
private void Frame()
{
    double x = 0, z = 0;

    x = MoveX();
    z = MoveZ();

    double[] colissions = Collision(x, z);
    x = colissions[0];
    z = colissions[1];

    sphereTranslation.OffsetX += x;
    sphereTranslation.OffsetZ += z;
}
```

Codefragment 4-2: frame

4.2 Wrijving

De wrijving van de bal wordt berekend aan de hand van de zwaartekracht en de wrijvingscoëfficiënt. Om een realistische wrijving te verkrijgen is gekozen voor een statische coëfficiënt van 0.05 en een dynamische coëfficiënt van 0.025. Dit zijn de waardes van een wrijving tussen metaal en hout. Indien de bal stilstaat wordt gebruik gemaakt van de statische waarde anders de dynamische. Doordat het spelbord gekanteld is zal niet de volledige zwaartekracht in werking zijn op de wrijving van de bal. De richting en grootte wordt bepaald door de cosinus met de hoek van het bord. Het is belangrijk dat de gebruikte waardes in centimeter staan sinds dat het bord in de echte wereld uitgedrukt zou worden in centimeters en niet in meters anders is het spelbord veel te groot en zijn de effecten van de fysica niet goed zichtbaar.



Figuur 4-1: wrijving over een helling [11]

De formule van wrijving wordt: $F_W = F_N * \cos(\alpha) * \mu$ Waarbij $F_N = m * a = m * 9.81 m/s^2$

```
private double GetBallFrictionForce(double mass, double angle)
{
    double frictionCoefficient = KINETIC_FRICTION_COEFFICIENT; //MOVING
    if (ballSpeedX == 0 && ballSpeedZ == 0)
        frictionCoefficient = STATIC_FRICTION_COEFFICIENT; //NOT MOVING
    return mass * (GRAVITY * Math.Cos(angle * Math.PI / 180)) * frictionCoefficient;
}
```

Codefragment 4-3: implementatie wrijving

4.3 Versnelling en snelheid

De versnelling van de bal gebeurd enkel door de zwaartekracht en de hoek van het spelbord. Doordat het bord gekanteld is moet de valversnelling berekend worden over een hoek net zoals bij de wrijving maar nu met de sinus.

```
private double GetBallAcceleration(double angle)
{
	return GRAVITY * Math.Sin(angle * Math.PI / 180);
}
```

Codefragment 4-4: Berekening versnelling

Door de wrijving en de versnelling kan de snelheid van de bal bepaald worden over een frame. De versnelling zal de balversnelling zijn met de wrijving verminderd. Om de acceleratie van de wrijving te verkrijgen wordt de wrijvingskracht gedeeld door de massa van de bal.

```
private double GetBallSpeed(double speed, double acceleration, double friction)
{
	return speed + (acceleration - friction) * frameInterval;
}
```

Codefragment 4-5: Berekening snelheid

Met de versnelling en de snelheid van de bal kan een afgelegde afstand verkregen worden over een tijdeenheid, een frame.

```
private double GetDistance(double friction, double a, double speed)
{
	return (speed * frameInterval) + ((1 / 2) * a * Math.Pow(frameInterval, 2));
}
```

Codefragment 4-6: Berekening afstand

4.4 Botsing

Om botsingen te detecteren wordt de absolute afstand tussen de bal en elke muur berekend dit op de x-as en de y-as. Als de afstand tussen beide voorwerpen kleiner of gelijk is aan de dikte van de muur opgeteld met de radius van de bal dan wordt de bal gestopt. Er wordt ook rekening gehouden of de bal wel in de buurt is van de muur ander zou het kunnen dat de bal tegengehouden wordt in een verlengde van een muur wat niet de bedoeling is. Om een realistische botsing te verkrijgen wordt de snelheid aangepast met de restitutiecoëfficiënt. Hierdoor zal de bal terugkaatsen telkens deze de muur raakt.

```
double wallToBallDistanceX = Math.Abs(ballPosX - cube.X - (cube.LX/2));
double wallToBallDistanceZ = Math.Abs(ballPosZ - cube.Z - (cube.LZ/2));

if ((wallToBallDistanceX <= cube.LX/2 + BALLRADIUS) &&
    (ballPosZ + BALLRADIUS-0.25) > cube.Z &&
    (ballPosZ - BALLRADIUS+0.25) < (cube.Z + cube.LZ))
{
       distanceX = 0;
       //v = -e * v0
       ballSpeedX = -(COR * ballSpeedX);
}
```

Codefragment 4-7: Botsing voor de x-richting

5 Onderzoek doolhof generatie

5.1 Recursieve Backtracking

Om dit soort algoritme toe te passen moet het bord opgedeeld worden in cellen waarvan elke cel muren heeft. Het algoritme kan op eender welke cel starten en stapt over naar een aanliggende cel in een random richting. De muur tussen de 2 cellen wordt verwijderd en de cel is gemarkeerd als bezocht. Dit blijft doorgaan tot alle omliggende cellen al bezocht zijn. Er kan dus niet naar een nieuwe cel gestapt worden waardoor er een stap terug moet gezet worden. Deze stappen worden doorlopen tot alle cellen bezocht zijn. Dit algoritme kan worden geïmplementeerd met gebruik van een Stack in C#, last-in-first-out. Dit algoritme maakt gebruik van depth-first. Hierdoor zal zo ver mogelijke gezocht worden naar vrije cellen tot een doodlopend einde bereikt wordt waarna backtracking is toegepast zodat alle cellen worden overlopen. [12]

De doorlopen stappen zijn als volgend:

1- Selecteer een cel en markeer deze als bezocht.
2- Indien er nog onbezochte cellen zijn:
    a. Als de geselecteerde cel aanliggende cellen heeft die nog niet bezocht zijn:
        i. Voeg de geselecteerde cel toe aan de stack.
        ii. Selecteer een random aanliggende cel en selecteer deze als bezocht.
        iii. Verwijder de muur tussen beide cellen.
    b. Als alle omliggende cellen al bezocht zijn en de stack niet leeg is:
        i. Haal een cel uit de stack en selecteer.

5.2 Recursieve Deling

In tegenstelling tot Recursieve backtracking wordt hier gestart vanuit een leeg bord. Dit bord wordt in twee gedeeld door een random toegevoegde muur met een opening, het bord bevat nu twee kamers die elkaar kunne bereiken via de opening van de muur. Het is ook mogelijk om meerder muren te gebruiken bij de opdeling. Dit proces wordt herhaald tot het bord niet meer kan opgedeeld worden in kleine kamers. [12]

5.3 Hunt and Kill

Dit algoritme start door een random weg te maken door de cellen waar bij elke overgang de muur wordt verwijderd. Indien een cel geen aanliggende onbezochte cellen meer heeft zullen de cellen gescand worden naar een onbezochte cel die een aanliggend is van een bezochte cel en het proces start weer door een random weg te maken. Dit wordt doorlopen tot alle cellen bezocht zijn. Dit algoritme zal lange wegen creëren met weinig doodlopende eindes. [13]

5.4 Eller's Algoritme

Dit algoritme is een van de snelste en kan doolhoven maken van een oneindige grootte over een lineaire tijd. Het doolhof wordt rij per rij opgebouwd. Er wordt bijgehouden welke cellen op een rij die elkaar connecteren. Er wordt nooit gekeken naar andere rijen. [14]

De doorlopen stappen zijn als volgend:

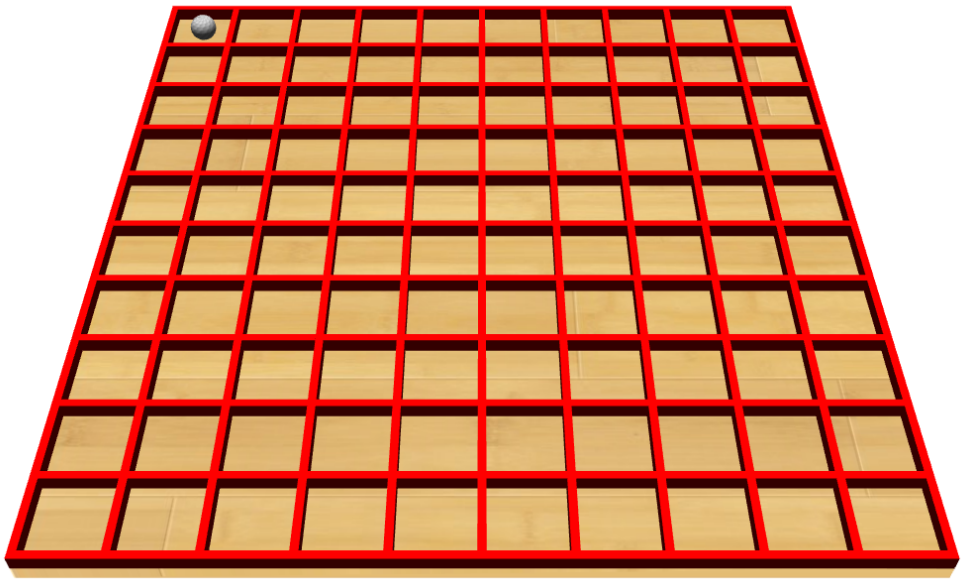
- Elke cel krijgt zijn eigen set.
- Cellen worden random aan elkaar toegevoegd en krijgen dezelfde set.
- De connectie naar de cellen van een rij lager worden random bepaald. Elke set moet minstens een connectie hebben. De cellen worden die geconnecteerd zijn krijgen ook dezelfde set.
- De overblijvende cellen krijgen hun eigen set.

Dit proces wordt doorlopen tot de laatste rij is bereikt.

6 Uitwerking doolhof generatie

6.1 Recursieve Backtracking

Doordat deze methode gebruik maakt van cellen moet het bord eerste opgedeeld worden. Elke cel zal omringd worden door vier muren die al dan niet zullen verwijderd worden door het algoritme. Het bord zal eruitzien zoals in Figuur 6-1. In principe krijgt elke cel maar 2 muren hierdoor moeten er minder muren geplaatst worden en is het programma performanter.



Figuur 6-1: Cellen voor het doolhof

De eerste cel rechtsonder krijgt de coördinaten [0,0]. Aan de hand van dit coördinaat kunnen de omliggende cellen opgevraagd worden voor te confirmeren of ze wel of niet bezocht zijn. Het algoritme zal random een aanliggende cel kiezen volgens de regels en de muur tussen beide zal worden verwijderd. De muren zijn geïdentificeerd aan hun coördinaten. De coördinaten van de muur die moet worden verwijderd kan dus berekend worden aan de hand van de cel-coördinaten en de richting naar waar het algoritme wilt verder gaan.

Het algoritme is geïmplementeerd in C# (Codefragment 6-1). De GetRandomDirection zal nagaan of de currentCell nog onbezochte aanliggende cellen heeft en een random directie meegeven.
De cases voor de andere richtingen zijn ook geïmplanteerd volgens hun DirectionType.

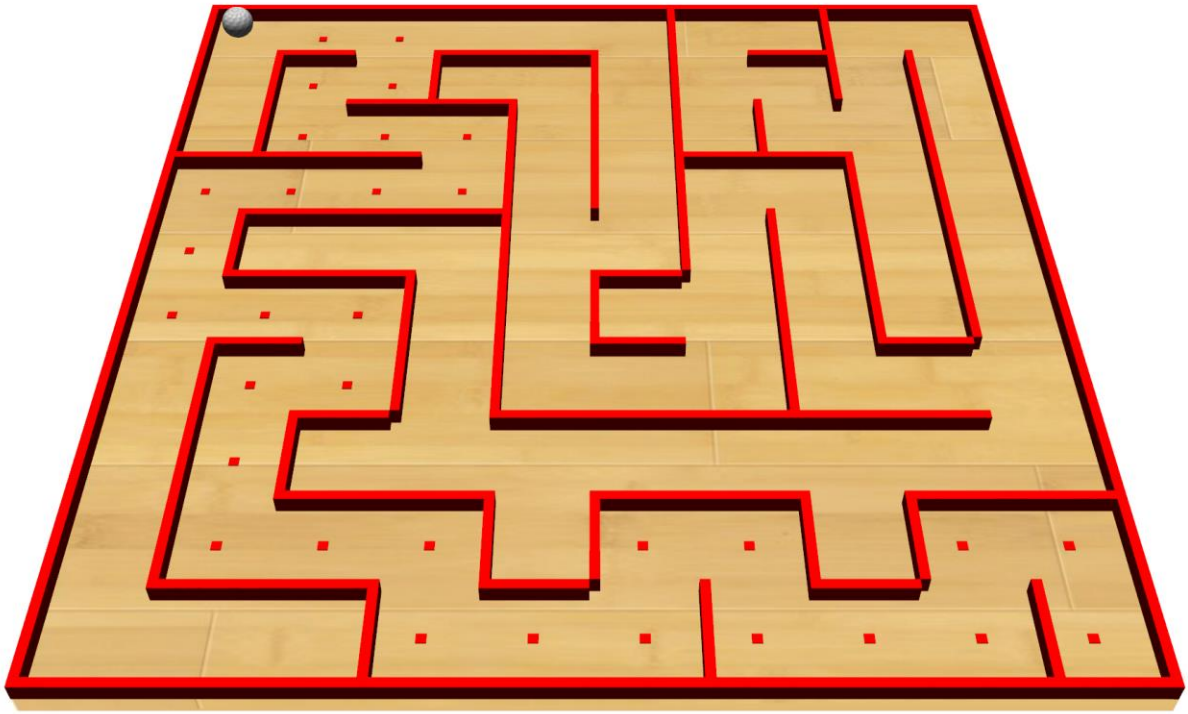
```
while (CellsToDo > 0)
{
	currentCell.Visited = true;

	switch (GetRandomDirection(currentCell))
	{
		case DirectionType.stay:
			if (stack.Count > 0) currentCell = stack.Pop();
			else CellsToDo -= 1;
			break;
		case DirectionType.up: //go up, remove wall from current
			stack.Push(currentCell);
			RemoveWall(currentCell, DirectionType.up);
			currentCell = cells[currentCell.X + "," + (currentCell.Z + 1)];
			CellsToDo -= 1;
			break;

			…

	}

}
```

Codefragment 6-1: Recursieve backtracking

Elke cel is eigenlijk maar geassocieerd met 2 muren, boven en links. Hierdoor moet er aan de hand van de richting eerst de currentCell veranderen naar de volgende cel of eerst de muur worden verwijderd. Dit is zo geïmplementeerd zodat er minder resources gebruikt worden het programma dus sneller kan werken. Als dit niet zo is zal elke cel vier muren hebben en moet er telkens twee worden verwijderd om een doorgang te creëren.

6.2 Kortste pad

Bij recursieve backtracking is er telkens maar 1 oplossing mogelijk wat dus ook het kortste pad zal zijn. Wanneer de cel gemarkeerd als einde is bereikt zal voor elke cel in de Stack een aanduiding gecreëerd worden. Dit visualiseert dan de weg van begin naar einde. Het uiteindelijk resultaat met de oplossing van het doolhof is te zien in Figuur 6-2.



Figuur 6-2: Doolhof met oplossing

Besluit

Uit het onderzoek is gebleken dat WPF veel functionaliteiten heeft, voor zowel 2D als 3D toepassingen. Helaas is er geen vooraf gedefinieerde code voor werkelijke 3D games te ontwerpen. Het is dus noodzakelijk om gameloops en fysica zelf toe te passen. Dit heeft natuurlijk een voordeel doordat er geen onnodige code moet draaien en de fysica specifiek ontworpen is voor de toepassing. Dit maakt de applicatie kleiner en met minder overhead. De code van WPF geeft een goed inzicht op de basis onderdelen die nodig zijn om een volledig werkende 3D applicatie te verkrijgen. Dit vooral doordat gewerkt wordt met driehoeken om een Mesh te creëren. Recursieve backtracking is een veel gebruikt algoritme om doolhoven te genereren. Dit algoritme kan zeer snel een random doolhof generen en het uiteindelijk resultaat hangt af van de gekozen richtingen tijdens het uitvoeren. Het is een zeer eenvoudig proces, om te verstaan maar ook om te implementeren. Het enige nadeel is dat er veel geheugen gebruikt wordt indien er veel cellen in de stack terecht komen.

Literatuurlijst

[1] Wikipedia, „Windows Presentation Foundation,” 31 3 2018. [Online]. Available: https://en.wikipedia.org/wiki/Windows_Presentation_Foundation.

[2] Wikipedia, „DirectX,” 9 4 2018. [Online]. Available: https://en.wikipedia.org/wiki/DirectX.

[3] M. James, „WPF The Easy 3D Way,” 10 6 2015. [Online]. Available: http://www.i-programmer.info/projects/38-windows/273-easy-3d.html?start=1.

[4] C. Moser, „Introduction to WPF 3D,” 4 7 209. [Online]. Available: http://wpftutorial.net/IntroductionTo3D.html.

[5] MathIsFun, „Vertices,” 2016. [Online]. Available: https://www.mathsisfun.com/geometry/vertices-faces-edges.html.

[6] S. V. Impe, „Game Loops,” 24 11 2015. [Online]. Available: http://svanimpe.be/blog/game-loops-fx.

[7] AmazingThew, „Fixed time step vs. variable time step,” Reddit, 2014. [Online]. Available: https://www.reddit.com/r/gamedev/comments/22k6pl/fixed_time_step_vs_variable_time_step/.

[8] Wikipedia, „Wrijving,” 2017 10 31. [Online]. Available: https://nl.wikipedia.org/wiki/Wrijving.

[9] Wikipedia, „Botsing,” 24 1 2018. [Online]. Available: https://nl.wikipedia.org/wiki/Botsing_(natuurkunde).

[10] B. Gideon, „Do C# Timers elapse on a separate thread?,” 20 11 2010. [Online]. Available: https://stackoverflow.com/questions/1435876/do-c-sharp-timers-elapse-on-a-separate-thread/1436331#1436331.

[11] S. P. Dinkgreve, „Ontbinden van krachten,” 2018. [Online]. Available: https://wetenschapsschool.nl/chapter/Kracht_5_Ontbinden+van+krachten.html.

[12] Wikipedia, „Maze Generation Algorithm,” 27 4 2018. [Online]. Available: https://en.wikipedia.org/wiki/Maze_generation_algorithm. [Geopend 7 5 2018].

[13] J. Buck, „Maze Generation: Hunt-and-Kill algorithm,” 24 1 20111. [Online]. Available: http://weblog.jamisbuck.org/2011/1/24/maze-generation-hunt-and-kill-algorithm. [Geopend 7 5 2018].

[14] J. Buck, „Maze Generation: Eller's Algorithm,” 29 12 2010. [Online]. Available: http://weblog.jamisbuck.org/2010/12/29/maze-generation-eller-s-algorithm. [Geopend 7 5 2018].