27 april 2018

Dwight Van der Velpen

Odisee

Advanced aplied programming

Inhoudsopgave

[1. Deel 1- 3D WPF Doolhof: 2](#_Toc515055921)

[1.1. Onderzoek en probleemstelling 2](#_Toc515055922)

[1.1.1. Textures 2](#_Toc515055923)

[1.1.2. Indices en corners 3](#_Toc515055924)

[1.1.3. Mazedata 4](#_Toc515055925)

[1.2. Opbouw Project en schema 5](#_Toc515055926)

[1.2.1. GUI 5](#_Toc515055927)

[1.2.2. Datastructuur.cs 5](#_Toc515055928)

[1.2.3. Shape, Block en Sphere 5](#_Toc515055929)

[1.2.4. Schema 5](#_Toc515055930)

[1.3. Problemen en vooruitgang 6](#_Toc515055931)

[2. Deel 2 – Physics 9](#_Toc515055932)

[2.1. Onderzoek en probleemstelling 9](#_Toc515055933)

[2.1.1. De benodigde fysica wetten 9](#_Toc515055934)

[2.2. Opbouw project en schema 10](#_Toc515055935)

[2.2.1. GUI 10](#_Toc515055936)

[2.2.2. Physics.cs 10](#_Toc515055937)

[3. Deel 3 – Maze generation 11](#_Toc515055938)

[3.1. Onderzoek en probleemstelling 11](#_Toc515055939)

[3.1.1. Recursief zoeken 11](#_Toc515055940)

[3.1.2. Eller’s 11](#_Toc515055941)

[3.1.3. Prim’s 11](#_Toc515055942)

[3.2. Opbouw project en schema 12](#_Toc515055943)

[3.2.1. Gui 12](#_Toc515055944)

[Bibliography 14](#_Toc515055945)

# Deel 1- 3D WPF Doolhof:

## Onderzoek en probleemstelling

De opdracht van dit deelproject is het namaken van een houten doolhof spel. In dit deel focusen we ons echter enkel op het visueel gedeelte: het voorstellen van een doolhof volledig opgebouwd uit blokken gecodeerd in C# en een bal hierin weergeven met een kleine animatie.

Om hieraan te beginnen kregen we echter een startproject met daarin een WPF toepassing, een cubus in WPF en een driehoek in WPF. Wegens ik al zelf een kleine ervaring had met WPF was de eerste dan ook niet zo speciaal voor mij. De driehoek was een interresante om te bekijken. Het toont echter hoe een indice opgesteld moest worden, hoe je een kleur aanbrengt enz.

Als laatste werd er gekeken naar de kubus. Deze was echter zeer informatief om een 3D kubus te tekenen en een kleine rotatie op te tonen. Echter was er een fout in de texures van deze. Enkel op de voor- en achterkant werd deze toegepast op een correcte manier. Het eerste onderzoek begon dus welliswaar naar het “corrigeren” van de voorbeeld code. Na enkele bronnen te bekijken kwamen er enkele nuttige boven gedoken.

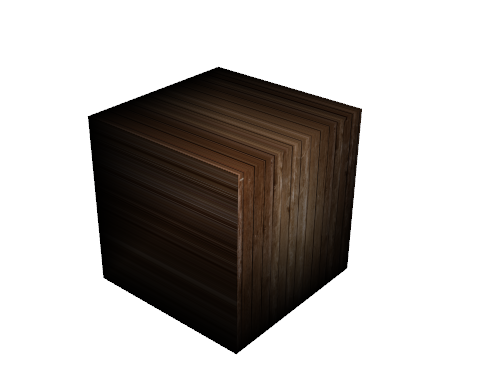
## Textures

Wat blijkt is dat de textuur-punten gelinkt zijn aan de hoeken van de kubus. Dit wilt zeggen dat elk hoekpunt een vaste coördinaat heeft in de textuur. Als we dan proberen om op elk vlak ieder van de 4 hoekpunten een unieke coördinaat te geven merken we snel dat dit onmogelijk is. Er zal altijd 2 vlakken vervormd zijn. Wegens 2 vlakken minder is dan 4 vervormde vlakken zou dit dus al een verbetering zijn op de voorbeeld code maar het kan nog beter.

Uit de vorige conclusie kunnen we dus 2 richtingen op gaan. Oftewel maken we extra hoeken aan, wel op dezelfde coördinaten, en geven we deze dan andere textuur-punten. Als we voor deze optie kiezen zullen we ook de indices van deze 2 vlakken moeten veranderen naar deze nieuwe punten.

Een andere optie is om een volledig nieuwe mesh te maken voor deze 2 vlakken en dan de 2 meshes samen bekijken als 1 object.

Wegens de 2e mogelijkheid zeer onhandig is, is er gekozen voor de 1e manier.

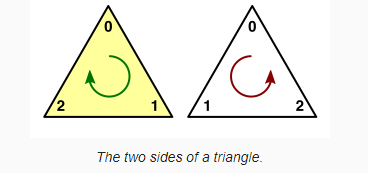
Na implementatie, en verandering van de belichting, bekomen we het volgend resultaat:

## Indices en corners

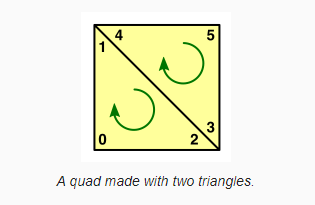
Elk 3D figuur in WPF bestaat uit 2 hoofdelementen, de punten die het beschrijven en de manier waarop deze verbonden zijn met elkaar voor de oppervlakte te beschrijven.

Deze hoekpunten zijn het simpelst voor te stellen als men denkt aan een kubus. Elk hoekpunt van de kubus kan men voorstellen als een dergelijk punt. Het is echter wel belangrijk in te zien dat **elk** 3D figuur bestaat hieruit. Bij bogen is het echter een benadering van de boog die nagebootst wordt doormiddel van verschillende punten. Hoe meer punten er beschreven worden hoe “ronder” de boog zal lijken. Zo zullen we ook in dit project een bal maken, als we hieraan komen zal de benadering dieper besproken worden.

Indices zijn de driehoeken die getrokken worden tussen ieder punt. Deze duiden niet enkel de oppervlakte aan maar ook de richting van deze. Deze driehoeken zijn enkel zichtbaar van 1 richting. In de onderstaande afbeelding word een driehoek getoond waarin de indice 0-1-2 gedefinieerd is. Merk op dat enkel op de kant waarin deze **kloksgewijs** georiënteerd zijn er een driehoek getekend is.

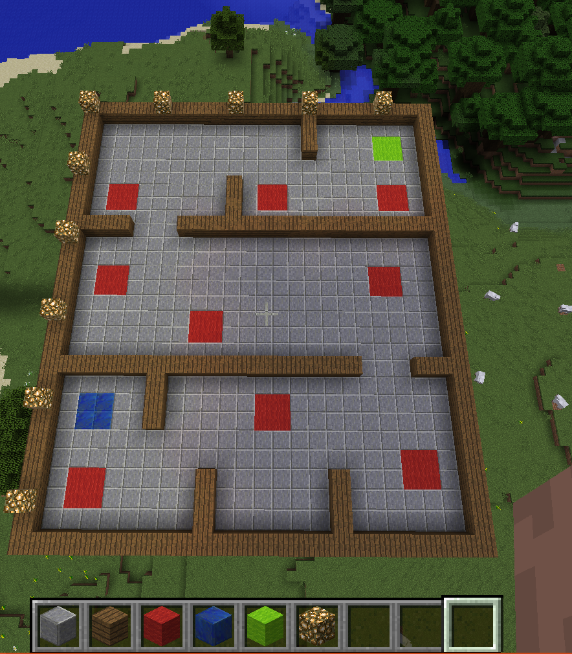


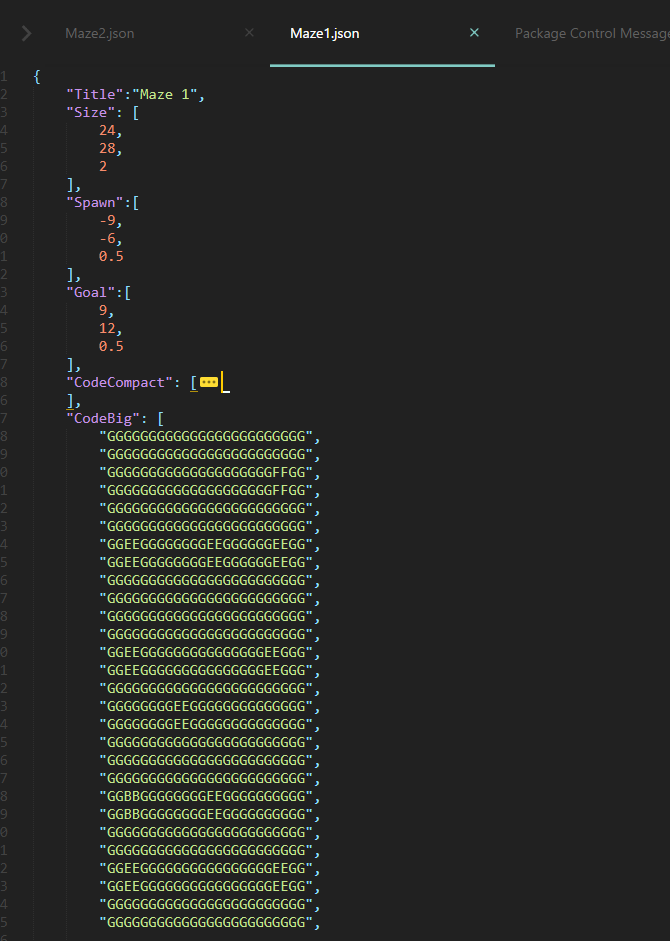
Het is dus de kunst om elk oppervlak te definiëren in verschillende indices. Zo kan een vierkant gedefinieerd worden in de samenstelling van 2 indices.



Een kubus zal dus bestaan uit 12 van deze indices.

## Mazedata

Nu we al de basis van 3D WPF hebben besproken kan het idee achter Mazedata uitgelegd worden. Toen er een doolhof ontworpen moest worden is er gekozen, puur uit gemakzucht, om deze snel op te bouwen in een 3D spel gebaseerd op blokken, Minecraft. Na het ontwerpen van 2 doolhoven, een makkelijk voor snelle tests en een iets uitgebreidere, kwam de vraag boven hoe ik deze kon implementeren in C#.

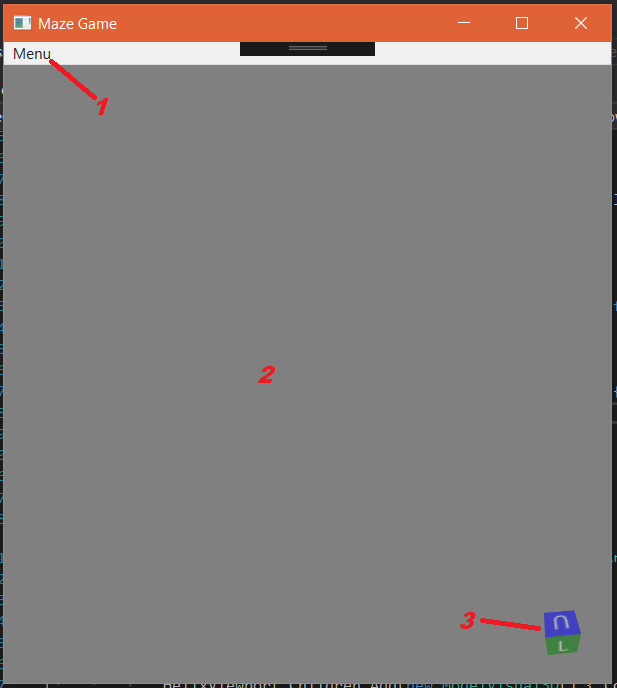
Elk van deze doolhoven bestaat echter uit enkele honderden blokken met de leegtes meegerekend. Er werd gekozen om de doolhoven op te slaan in een JSON bestand met daarin verschillende andere velden zoals waar de start en finish zijn, wat voor blokken er waar gebruikt moeten worden en dergelijke.

## Opbouw Project en schema

## GUI

In je Gui hebben we een interface met de volgende elementen:

1. Een menu knop waarin een JSON file voor een doolhof kan ingelezen worden.
2. Een HelixViewport, hierin word het doolhof weergegeven.
3. De HelixViewport camera controller.



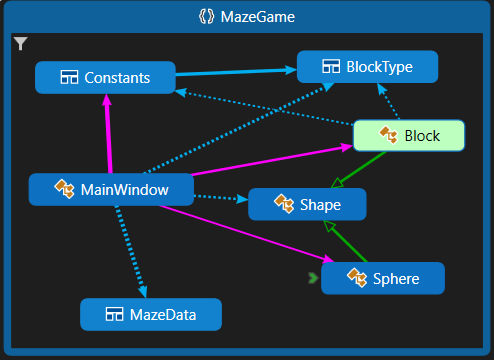
## Datastructuur.cs

Hierin worden de verschillende structs opgesteld die gebruikt worden doorheen de code. Zo is er bijvoorbeeld een constants struct met daarin alle aanpasbare variabelen. Een struct genaamd Mazedata die de JSON file voorstelt. Een blocktype struct om de verschillende blocktypen onder te brengen.

## Shape, Block en Sphere

Er is gekozen om een abstracte basisklasse Shape te maken. Hierin worden enkele gemeenschappelijke velden (zoals de Geometrymodel en dergelijke) gedefinieerd en gelinkt aan elkaar. Zo kan er in Block en Sphere gefocust worden op enkel het specifieke voor elk kubus en bal object.

## Schema



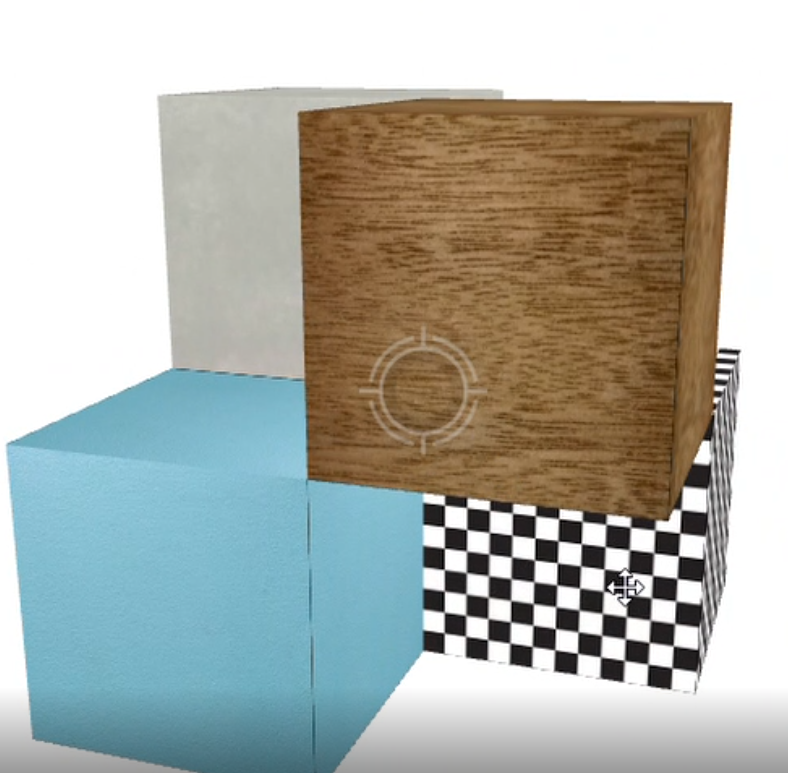
## Problemen en vooruitgang

Wat nu volgt is het “dagboek” van hoe er gewerkt werd deze opdracht.

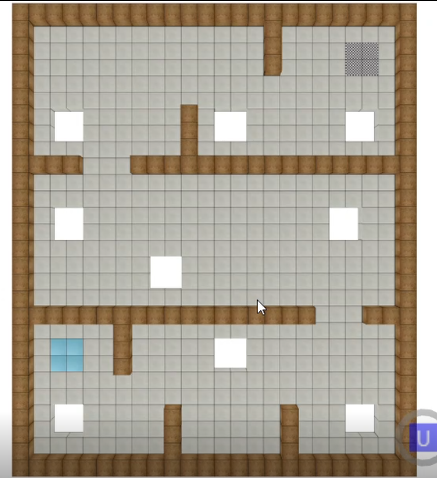
* Ik heb reeds enige voorkennis van WPF dus er word geen vooronderzoek gedaan.
* Ik bekijk de voorbeeld code en kopieer grotendeels de kubus code.
* Probeer deze te “objectificeren”.
* **De kubus is niet zichtbaar!**
  + Het was wegens de Canvas niet gelinkt was met de modelgroup, (de collectie van alle visuele objecten. Hierdoor was er geen licht of kubus aan het visuele.)
* Merkt op dat Cube textuur verkeerd is
* Ook het geval in Demo code
  + Na een uitbundige google-trip ondervonden dat deze gelinkt zijn aan de hoekpunten en het dus onmogelijk is om alles correcte textuur te geven met enkel 8 punten.
  + Zie eerder besproken deel om oplossing te zien.



* Experimenteert met licht en merkt dat ambient goed is voor een algemene belichting, directioneel het toch iets meer “3D” laat uit zien.
* Speelt wat met de camera en bekijkt enkele video’s hierrond om de verschillende properties en soorten te begrijpen.
  + Orthografische camera is raar en word vermeden.
  + Perspectief camera is de gebruikte camera.
    - Field of view is de hoek dat de camera kan zien.
    - View direction is een vector die aantoont waar de camera naar kijkt.
    - Up direction is een vector die aantoont welke richting de bovenkant is van de camera.
    - Near & far plane bepalen de culling afstanden.
* Ik besluit om de animatie weg te laten voor nu.
* Ik Google een Maze voorbeeld om na te bouwen.
  + Ik bouw de mazes in Minecraft puur uit gebruiksvriendelijkheid.
  + Elk blokje krijgt een letter en deze worden opgeslagen in een JSON file.
* Verzamelt enkele texturen voor de verschillende blokken
* Maakt de CTOR voor de kubus zodat deze gemaakt kan worden met maar 1 gegeven punt (p6) de linker-onder-achter hoek.
  + De indices zijn binnen i.p.v. buiten getekend; dus deze omgedraaid.
* Implementeerd de String to Cubes (Json naar maze)
  + Ik vergat een counter te gebruiken dus enkel de eerste blok werd getekend



* De volledige JSON ingelezen zonder probleem



* Ik maakte een reset methode die de volledige maze ook “ontladen”
  + - Puur het model resetten werkte niet. Je moet de Child elementen effectief verwijderen.
  + Voegde da mazedata toe
  + Probeerde de Hotswapping uit zonder problemen
  + Events voor keypresses geïmplementeerd
    - De Helixviewport gebruikt echter WASD en de pijltjes knoppen al voor camera besturing.
    - Er word gekozen voor de numpad “pijltjes”
    - Aan de hand van welke knoppen ingeduwd zijn word een boolean array geüpdatet [1] [2].
    - Deze array bepaalt via een timer hoe het bord kantelt. Deze timer word x aantal keren per seconde getriggerd en maakt zo een “frame” aan.
  + Na enkele uren te zoeken over een bal te maken in WPF een duidelijke gevonden.
    - Implementatie verloopt vlekkeloos en uitleg was voldoende om te begrijpen.
    - Het principe is dat je 2 velden, phi en theta, meegeeft en deze bepalen in hoeveel punten de bal word opgedeeld. De phi deelt de bal op van boven naar onder in verschillende “lagen”. Hoe hoger de phi, hoe beter de **verticale** rondheid van de bal. De theta bepaalt dan per laag in hoeveel delen deze onderverdeeld mag worden (volledige cirkel). De samenwerking van de 2 bepalen dus het aantal indices over de oppervlakte.
* Cube en Sphere gebaseerd op een abstracte klasse genaamd shape om de overlappende code weg te werken.
* Sphere werkt perfect.
* Een blog opgezocht voor optimalisatie van punten voor sphere, wat leid naar nog betere rondheid.
* Heel, heel lang gezocht op de bal animatie.
  + - Besloten om de bal gewoon te verplaatsten evenredig met de tilt.
    - De center van de bal aanpassen werkt niet, je krijgt dan een slang vormig figuur.
    - Echter de Transform property aanpassen werkt vlekkeloos.

# Deel 2 – Physics

## Onderzoek en probleemstelling

De opdracht van dit deelproject is het simuleren van real-life fysica. In dit deel focussen we ons dus enkel op het fysisch gedeelte: het bewegen van de speelbal met verschillende mogelijke fricties en het botsen van de bal tegen muren. Dit wilt dan ook zeggen dat er een soort gamestate geïmplementeerd zal worden om aan te tonen opdat de speler al dan niet de finish heeft bereikt.

Als startpunt werd er simpelweg deel 1 gekopieerd en onderzoek gedaan naar alle nodige fysica wetten.

### De benodigde fysica wetten

Voor de toepassing niet ingewikkelder te maken dan nodig is werd de speler aanschouwd als een blok op een helling. Hiervoor kon er dan in enkele stappen een robuuste basis aan fysica wetten onderzocht worden.

Als startpunt onderzoeken we de verplaatsings formule. Deze formule helpt ons bij een probleemstelling waarin de versnelling, beginsnelheid en duratie gekend is, en er word ons dan een verpatste afstand gegeven.

Hierna kunnen we de eindsnelheidsformule ook onderzoeken:

Met deze 2 formules kunnen we per “fysica-loop” de verplaatsing en snelheids verandering berekenen als we de versnelling weten.

Om de versnelling te bepalen gaan we echter gebruik moeten maken van de helling’s opstelling. Deze combineerd de wrijvingswetten, de kracht’sonbindingswetten en de 2e wet van Newton om ons zo een versnelling te geven.

Vooraleer we deze kunnen opsommen enkele kleine aandachtspunten:

* De versnelling wordt veroorzaakt door de zwaartekracht die op de speler inwerkt. Als deze op een helling staat zal deze ontbinden in een kracht loodrecht op de helling en een kracht parallel ermee. De kracht die loodrecht staat wordt de normaal kracht genoemd en zullen we gebruiken voor de wrijving te bepalen. De andere is de “pure” kracht die de blok op de helling duwt.
* Wrijving bestaat uit 2 delen, een statische wrijvingsfactor en een kinetische. Beide zijn factoren tussen de 1 en 0 gelegen en deze tonen aan hoeveel kracht de zwaartekracht de beweging zal verhinderen.
* De Statische wrijvingskracht is de kracht die overkomen moet worden bij stilstand om een beweging te verwekken.
* De Kinetische wrijvingskracht is de kracht die de op de blok inwerkt indien de speler beweegt. Deze is per definitie altijd kleiner of gelijk aan de statische factor. Het is echter moeilijker om een voorwerp te doen bewegen dan om hem in beweging te houden.
* Een “negatieve” versnelling kan aangezien worden als een vertraging.

Alle formules die nodig zijn om de huidige versnelling te bepalen zijn

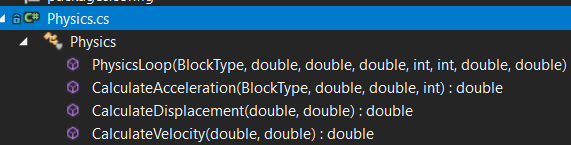
Als allerlaatste wet is de weerkaatsing formule nodig. We kunnen deze versimpeld aanzien als dat elk statisch voorwerp een weerkaatsingfactor heeft en als er een kinetisch voorwerp (de speler) mee in aanraking komt zijn snelheid verlaagd word met deze factor en van richting veranderd.

## Opbouw project en schema

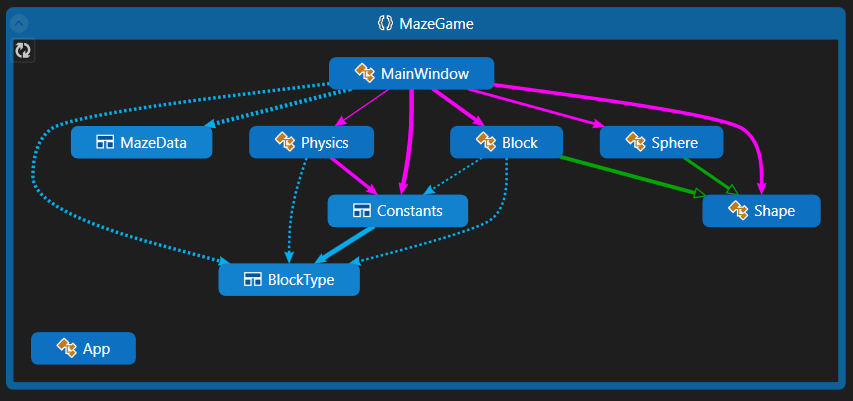
### GUI

Ten opzichte van het vorige deel is er aan de gui niet veel veranderd. Het enige dat onmerkbaar is, is dat de 0-toets een pauze functie heeft en de plus-toets als een herstart functie dient.

### Physics.cs

Wegens alle weten van de fysica onveranderlijk zijn, zijn deze dan ook in een statische methode ondergebracht. Deze bestaat uit 1 hoofd methode die verschillende sub-methoden zal oproepen voor de verschillende onderdelen. 

De loop zelf word aangeroepen op de zelfde plek waar het tilten van het bord gebeurd.



# Deel 3 – Maze generation

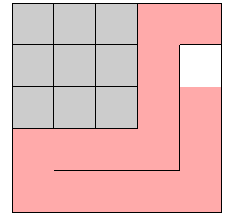
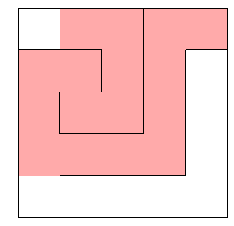
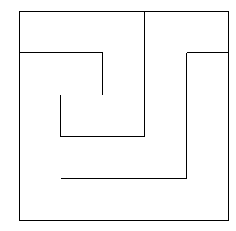
## Onderzoek en probleemstelling

De opdracht van dit deelproject is het genereren van doolhoven. In dit deel focussen we ons dus enkel op het creëren van JSON bestanden die de doolhof data bevatten.

Er werd als suggestie enkele verschillende algoritmen gesuggereerd en in dit deel gaan we elke stap per stap uitleggen. Elk van deze word op een doolhof toegepast met een vooraf gedefinieerde grootte.

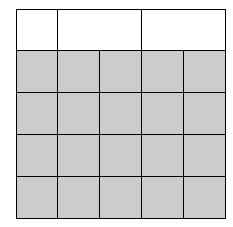
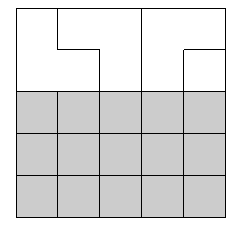
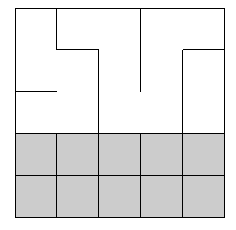
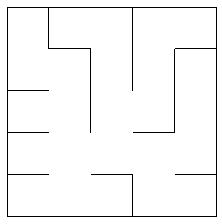
## Recursief zoeken

Deze is de meest van zichzelf sprekende manier. Er wordt een willekeurig startpunt gekozen en van daaruit word er alsmaar een random weg gemaakt doorheen het doolhof tot deze op een doodlopend einde komt. Hierna zal er via backtracking een punt gezocht worden waar dat er wel nog een opening beschikbaar is en word er vandaar random een weg gemaakt tot deze doodloopt. Dit herhalen we tot we weer bij het begin punt uitkomen met de backtracking.

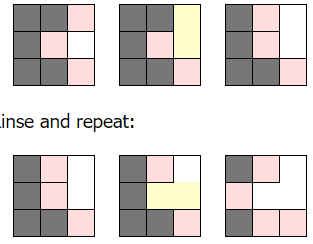
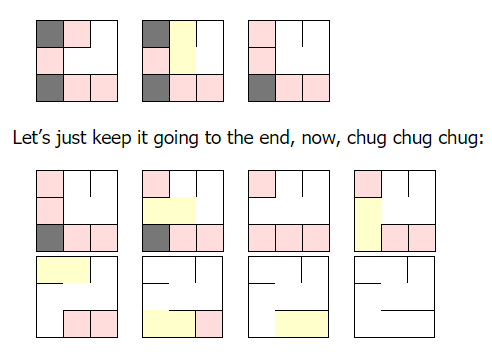
### Eller’s

Eller’s alogaritme bekijkt de maze van top naar bottom en deelt deze telkens op in verschillende random sub-divisies. Door laag per laag zo het doolhof af te zakken en nooit een divisie aan zich zelf te verbinden op een lagere laag, zal er uiteindelijk een perfect doolhof bekomen worden.

### Prim’s

Prim’s alogaritme lijkt sterk op de backtracking maar gaat iets anders te werk. Deze kiest ook een random startpunt maar duid alle “omliggende” cellen aan als een front. Daarna zal deze 1 van fronts willekeurig kiezen en aan een willekeurig “ingevulde cel” linken. Hierbij worden alle omliggende cellen ook een front. Dit word herhaald tot er geen enkel front overblijft.

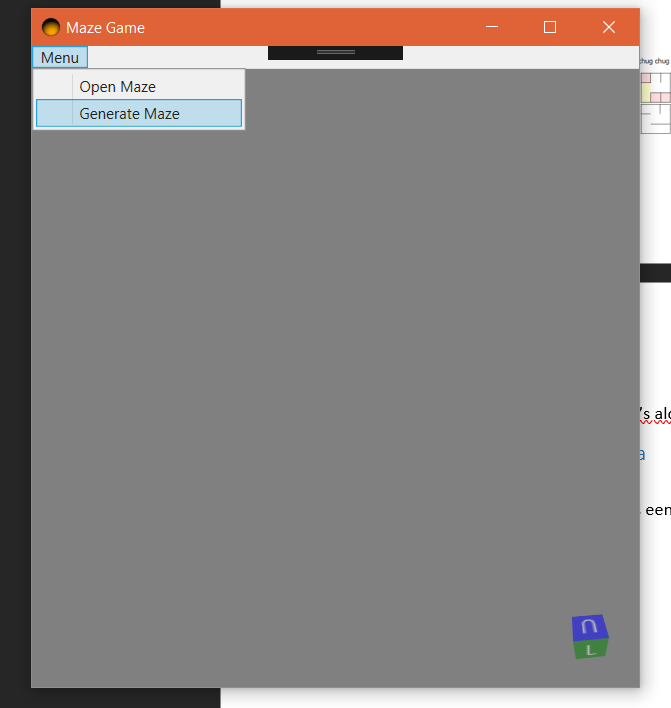
 

Er werd gekozen voor als uitdaging Prim’s alogaritme te implementeren.

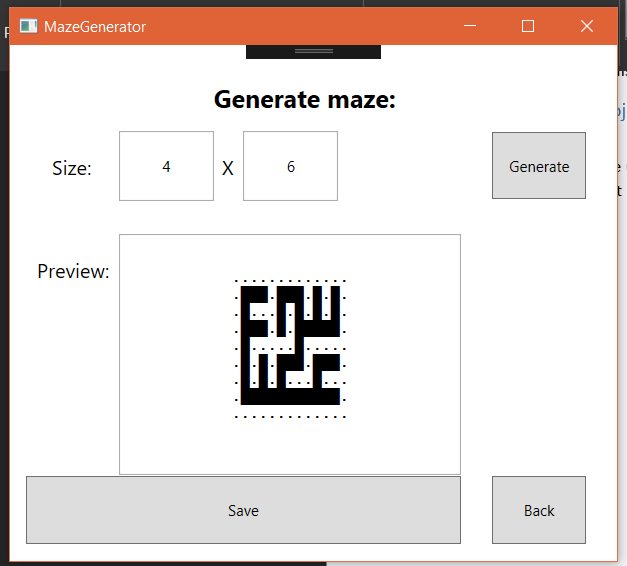
## Opbouw project en schema

### Gui

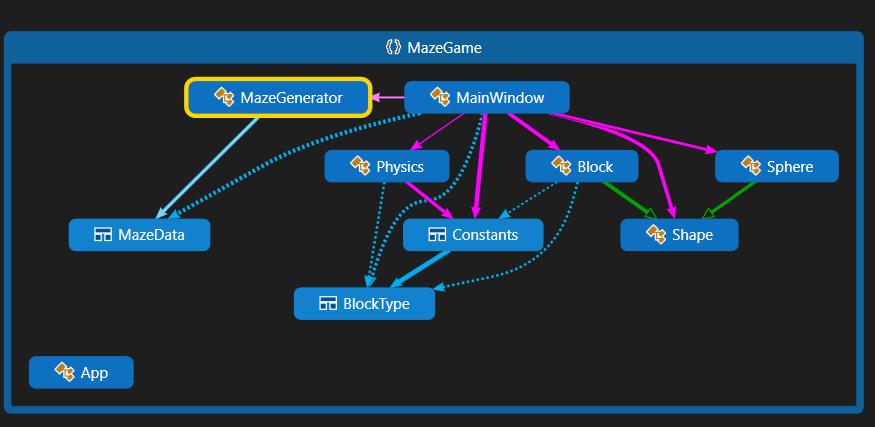
De GUI krijgt een kleine uitbreiding: er is een nieuw menu element om een maze te doen genereren. Als deze gekozen wordt opent er een nieuw venster.



Het nieuwe venster bevat een veld om de grootte in te stellen en daarna een knop om een random doolhof te genereren. Er wordt daarna een voorbeeld getoond van het doolhof en een optie getoond om deze als een JSON bestand op te slaan.



Hierna kan deze ge-opend worden en bespeeld worden.



# Bibliography

|  |  |
| --- | --- |
| [1] | “Hout textuur,” [Online]. Available: https://www.jukeboxprint.com/images/Walnut\_Enlarged.jpg. |
| [2] | “Verschillen tussen lichtbronnen,” [Online]. Available: https://msdn.microsoft.com/en-us/library/system.windows.media.media3d.light(v=vs.110).aspx. |
| [3] | “Hoe texturen met apparte vlakken,” [Online]. Available: http://csharphelper.com/blog/2014/10/make-a-3d-cube-with-pictures-on-its-sides-with-xaml-and-c/. |
| [4] | “Textures uitleg,” [Online]. Available: https://msdn.microsoft.com/en-us/library/system.windows.media.media3d.meshgeometry3d.texturecoordinates(v=vs.110).aspx. |
| [5] | “Camera properties,” [Online]. Available: https://msdn.microsoft.com/en-us/library/system.windows.media.media3d.projectioncamera(v=vs.110).aspx. |
| [6] | “Verschil tussen Ortho en Perspectief camera,” [Online]. Available: https://www.youtube.com/watch?v=c\_WVffzc6GU. |
| [7] | “Maze inspiratie,” [Online]. Available: https://lh3.googleusercontent.com/SFmD2hYPJgXPOZPORieG8Kozu2urdXGD7r3qpWvhgiNQKuUcd6Eo5IgJDAl0EU1nz-DT=h310. |
| [8] | “Concrete texture,” [Online]. Available: http://www.grespania.com/cmsupload/catalogo-despiece/concrete-gris-1x1\_rgb.preview.jpg. |
| [9] | “Finish textuur,” [Online]. Available: https://www.gettyimages.be/detail/illustratie/vector-finish-flag-royalty-free-illustraties/671382102?esource=SEO\_GIS\_CDN\_Redirect. |
| [10] | “Base textuur,” [Online]. Available: https://www.gettyimages.be/detail/foto/blue-wall-background-royalty-free-beeld/531109151?esource=SEO\_GIS\_CDN\_Redirect. |
| [11] | “Helix toolkit,” [Online]. Available: https://github.com/helix-toolkit/helix-toolkit/wiki/Getting-started-with-WPF-3D. |
| [12] | “Menu balk,” [Online]. Available: https://stackoverflow.com/questions/9936796/create-a-menu-bar-in-wpf. |
| [13] | “Grid uitleg WPF,” [Online]. Available: https://www.wpftutorial.net/GridLayout.html. |
| [14] | “structs,” [Online]. Available: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/using-structs. |
| [15] | “Button events,” [Online]. Available: https://social.msdn.microsoft.com/Forums/vstudio/en-US/9e1a598d-1ce6-4d43-9b25-3bebb513b21b/touch-long-press-on-buttons-in-wpf?forum=wpf. |
| [16] | “Rotaties enz,” [Online]. Available: https://docs.microsoft.com/en-us/dotnet/framework/wpf/graphics-multimedia/how-to-apply-multiple-transformations-to-a-3-d-model. |
| [17] | “Sphere,” [Online]. Available: http://csharphelper.com/blog/2015/04/draw-spheres-using-wpf-and-c/. |
| [18] | “Sphere deel 2,” [Online]. Available: http://csharphelper.com/blog/2015/04/draw-smooth-spheres-using-wpf-and-c/. |
| [19] | “Indices,” [Online]. Available: http://catlikecoding.com/unity/tutorials/procedural-grid/. |
| [20] | “Hoe texture points werken,” [Online]. Available: https://stackoverflow.com/questions/18776406/wpf-3d-understanding-texture-coordinates. |
| [21] | “Textures in WPF hulp,” [Online]. Available: http://csharphelper.com/blog/2014/10/make-a-3d-cube-with-pictures-on-its-sides-with-xaml-and-c/. |
| [22] | “Frictie wetten oefeningen,” [Online]. Available: https://physics.info/friction/practice.shtml. |
| [23] | “Acceleratie op een heuvel,” [Online]. Available: http://www.physicsclassroom.com/class/newtlaws/Lesson-3/Finding-Acceleration. |
| [24] | “Helling fysica,” [Online]. Available: http://physics.bu.edu/~duffy/semester1/c5\_incline.html. |
| [25] | “Hellings wetten,” [Online]. Available: http://www.physicsclassroom.com/class/vectors/Lesson-3/Inclined-Planes. |
| [26] | “Jsons opslaan,” [Online]. Available: https://www.newtonsoft.com/json/help/html/SerializeWithJsonSerializerToFile.htm. |
| [27] | “Uitleg Prim,” [Online]. Available: http://weblog.jamisbuck.org/2011/1/10/maze-generation-prim-s-algorithm.html. |