# Maze Generator

Naam: Gino van de Graaf.

| Version | Decription |
|---|---|
| 0.1 | Opzet en hoofdvraaag en subvragen. Eerste uitwerking. |
| 0.2 | Verbetering van de design vraag en toevoegen op prototype. |
| | |
| | |

# Inhoud

# Voorwoord

Deze onderzoek gaat gepaard met een devlog om beter te kunnen demonstreren hoe bepaalden doolhoven generators werken.

Dit doolhof procedurele generator wordt gebruikt in de VR game zenuwspiraal dolhof. Maar het probleem is hoe ga ik een doolhof maken. Welken soorten generators zijn er? Welke is het beste om te gebruiken en kan snel een doolhof maken. Dit document zal al mijn antwoorden geven op deze vraag.

# Design vraag

Design een doolhof generator die tot in staat is om mijn te helpen om te leren van procudual genration.

# Hoofdvraag

Welke doolhof generator is het beste voor mijn zenuwspiraal game?

## Sub vragen

Welke doolhof generators zijn er? (Libary Literature study)

Welke doolhof generator voelt het beste aan voor de elektrisch doolhof? (Libary SWOT analysis)

Het maken van een prototype doolhof in code.

Het testen/feedback van ander of de doolhof generator goed werkt.

Verdere stappen.

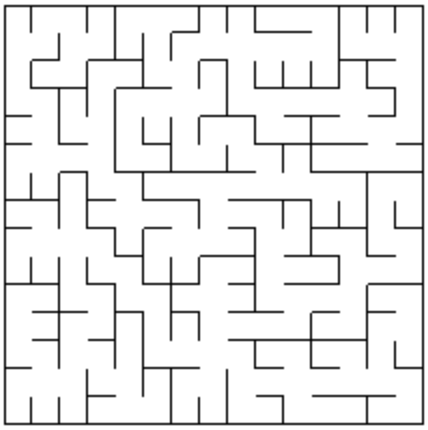
# Welke doolhof generators zijn er?

Er zijn verschillende soorten doolhof generators die ik kan gebruiken en allenmaal hebben ze hun eigen manier van genereren. Dus heb wat regels voor mijzelf opgesteld waar het aan moet voldoen.

1. Een doolhof generator moet snel zijn met het generareen. En moet niet complex zijn in het programmeren.
2. Een doolhof generator moet niet voorspelbaar zijn.( dit kan bijvoorbeeld dat de uitgang van de route altijd hetzelfde is)
3. En het moet een beetje natuurlijk aanvoelen. We willen geen grote hobbels hebben waarbij je arm iedere keer moet draaien.
4. Er moeten zo weinig mogelijk dead ends creëren( dit betekent dat je de doolhof niet kan oplossen)

## Heeft de criteria

### Kruskal's Algorithm



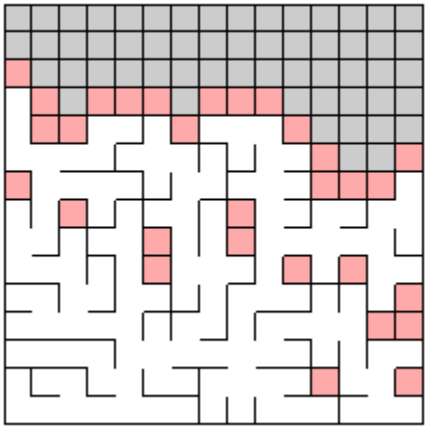
De kruskal algortime is een methode voor het produceren van zetten blokken die op elkaar passen

- Doe alle randen in de grid in een grote verzameling.
- Haal de rand met het laagste gewicht eruit. Als de rand twee afzonderlijke bomen verbindt, verbind dan de bomen.
- Gooi anders die rand weg. Herhaal dit totdat er geen randen meer over zijn.

Dit algoritme is snel met generen van een doolhof. Het probleem van dit algoritme is dat veel korten eindpunten heeft.

## Prim's Algorithm

Prim's benadert het probleem vanuit een andere hoek. In plaats van de hele grid rand voor rand af te werken, begint het bij een punt en groeit het daarvandaan. De standaardversie van het algoritme werkt als volgt:

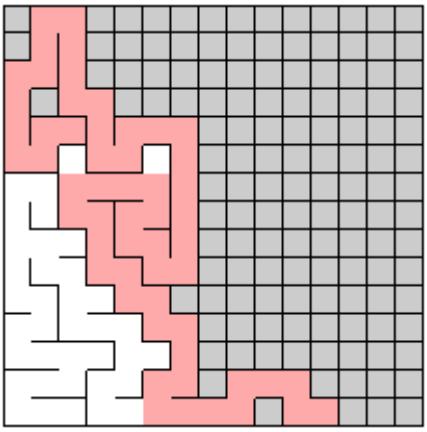


- Kies een willekeurige knoop uit de lijst en voeg deze toe aan een (initieel lege) verzameling.
- Kies de rand met het kleinste gewicht uit de lijst, die een knoop verbindt met een andere knoop die niet in zit.
- Voeg die rand toe aan de minimale spannende boom en voeg de andere knoop van die rand toe aan.
- Herhaal stappen 2 en 3 totdat alle knopen van de lijst bevat.

Het algoritme is best snel omdat hij meerder punten behandelt. Heeft wel hetzelfde probleem als de kruskal. En dat is dat ook korte eindpunten heeft.

## Growing Tree algorithm

Het werkt bijna precies hetzelfde als het algoritme van Prim. Met nog een kleine wijziging kun je doolhoven genereren met eigenschappen van beide.



Zo werkt het:

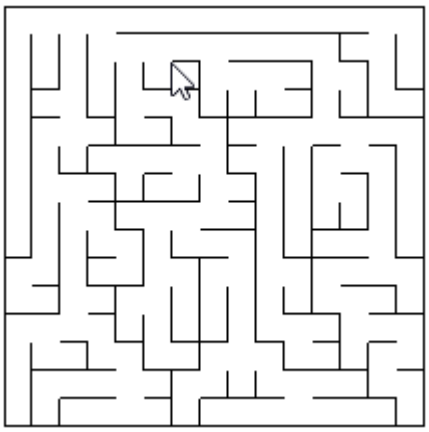
- Laat C een lijst van cellen zijn, aanvankelijk leeg. Voeg één cel toe aan C, willekeurig gekozen.
- Kies een cel uit C en snij een doorgang naar een onbezochte buur van die cel en voeg die buur ook toe aan C. Als er geen onbezochte buren zijn, verwijder dan de cel uit C.
- Herhaal #2 totdat C leeg is.

Is wat langzamer dan de ander 2 algoritmes. Maar is wel de meest gebruikte.

## Haalden niet de criteria

Dit zijn generators die niet haalden vanwege wat regels die ik had. Dat betekent niet dat het slechte algoritmes zijn. Maar ze voldoen niet aan de eisen wat ik wil.

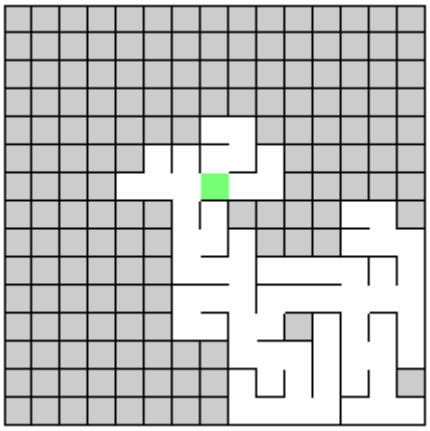
### Sidewinder algorithm



het doolhof maakt het makkelijk om elke keer hetzelfde patroon te maken vooral boven in is een rechte weg die verbind met alle ander locaties. Het algoritme is wel snel met het generen van de maze.

### Aldous-Broder

Leuk om naar te kijken maar het duurt zo lang dat het niet geschikt is. Het duurt zolang dat de prim doolhof 18 keer sneller is met generen.



## Welke doolhof generator voelt het beste aan voor de electric maze?

Hierin doe ik een korte bench mark hoe snel ze zijn met generen van een doolhof. Alle doolhoven worden genereerd in een 16*16 grid. Dus 256 tegels.

| Doolhof algoritme | Snelheid | Overal toegang | Makkelijk op te lossen | Maakt het doolhof natuurlijk gevoel |
|---|---|---|---|---|
| kruskal | 6 seconden | ja | gemiddeld | Nee maakt korte eindpunten maar heeft wel langer opeenvolgende stukken |
| prim | 5 sec | Ja | gemiddeld | |
| Growing tree | 11 seconden | Ja | moeilijker | Meer bochten ook rechte stukken. Begin van generatie is random |
| sidewinder | 7 seconden | Ja | Te makkelijk | Nee een gedeelte blijft hetzelfde met ieder generatie |
| Binary tree | 6 seconden | Ja | Te makkelijk | Nee n gedeelte blijft hetzelfde met ieder generatie |
| Aldous-Broder | 1 min en 28 second | ja | Moeilijker op te lossen | Ieder doolhof is uniek omdat het startpunt random is. |
| | | | | |

## Het maken van een prototype maze in code.

Ik heb gekozen voor de Growing tree algorithme van de doolhof generator deze is snel en makkelijk te maken niet alleen dat het wordt ook veel gebruik van gemaakt.

```
public class GrowingTree : MonoBehaviour
{
    //Current is the node to be checked if it has complete all side go down by one if there is no node left complete
    // the allgorithme
    // complete node list is for all nodes that has been checked
    // Return noeList is all the node getadded in here for later refrence of sides..

    List<NodeGridSystem.NodeGridObject> CurrentNodeList = new List<NodeGridSystem.NodeGridObject>();
    List<NodeGridSystem.NodeGridObject> CompleteNodeList = new List<NodeGridSystem.NodeGridObject>();

    // we need the start node and the grid itself.
    public void StartGrowingTreeAlgortime(NodeGridSystem.NodeGridObject startNode,Grid<NodeGridSystem.NodeGridObject> grid)
    {
        CurrentNodeList.Clear();
        CompleteNodeList.Clear();



        CurrentNodeList.Add(startNode);
        CurrentNodeList[0].UpdateTileChecked(NodeGridSystem.NodeGridObject.TileChecked.Current);
        StartCoroutine(GenerateMaze(grid));
    }
```

*Code figuur 1*

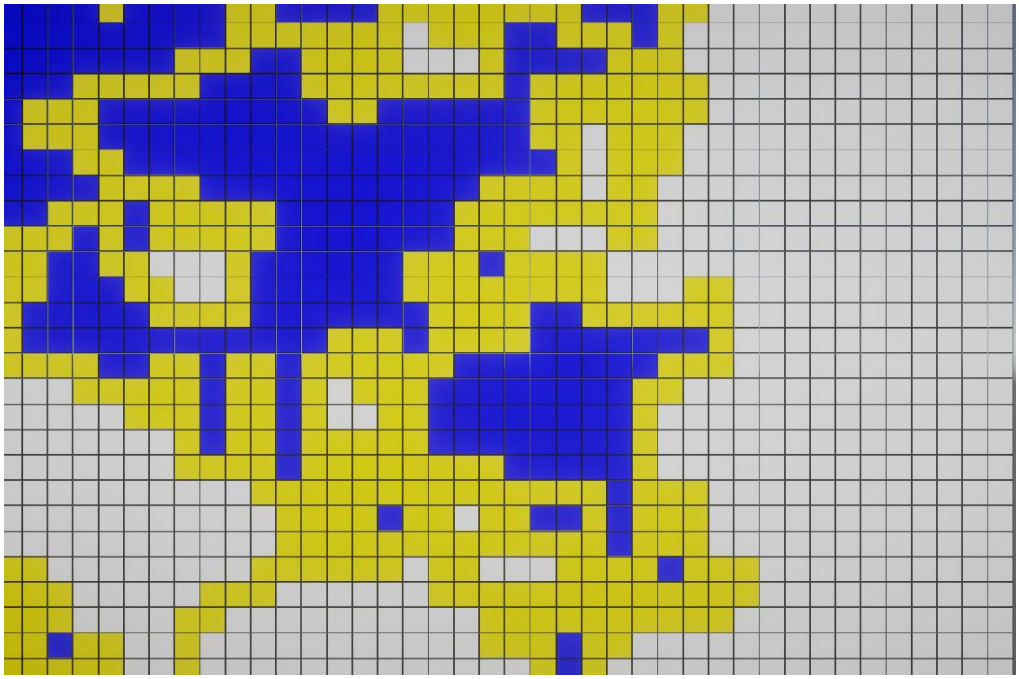
de code bestaat uit 2 delen het eerste gedeelte in *code figuur 1* is gewoon de setup. Hij moet de grid hebben en twee lijsten in lijst is voor de nodes die hij moet checken en de tweede is voor alle nodes die zij gecheckt.

```
IEnumerator GenerateMaze(Grid<NodeGridSystem.NodeGridObject> grid)
{
    int worldSize = grid.GetHeight() * grid.GetWidth();
    while (CompleteNodeList.Count < worldSize)
    {
        NodeGridSystem.NodeGridObject nodeGrid = CurrentNodeList[CurrentNodeList.Count-1];
        //possiable Direction
        List<NodeGridSystem.NodeGridObject> possibleNodeGridObjects = new List<NodeGridSystem.NodeGridObject>();
        GetNode(nodeGrid, grid, 1, 0, possibleNodeGridObjects);
        GetNode(nodeGrid, grid, -1, 0, possibleNodeGridObjects);
        GetNode(nodeGrid, grid, 0, 1, possibleNodeGridObjects);
        GetNode(nodeGrid, grid, 0, -1, possibleNodeGridObjects);

        if(possibleNodeGridObjects.Count>0)
        {
            NodeGridSystem.NodeGridObject chooseNode = possibleNodeGridObjects[Random.Range(0, possibleNodeGridObjects.Count)];
            CurrentNodeList.Add(chooseNode);
            chooseNode.UpdateTileChecked(NodeGridSystem.NodeGridObject.TileChecked.Current);
        }
        else
        {
            CompleteNodeList.Add(CurrentNodeList[CurrentNodeList.Count - 1]);

            CurrentNodeList[CurrentNodeList.Count - 1].UpdateTileChecked(NodeGridSystem.NodeGridObject.TileChecked.Checked);
            CurrentNodeList.RemoveAt(CurrentNodeList.Count - 1);
        }
        yield return new WaitForSeconds(0.00f);
    }
}
```

*code figuur 2*

het 2de gedeelte is de algorithme het zelf het checkt of het alle nodes heeft gedaan zo niet blijft het doorgaan todat alle nodes in de compleet lijst zit. Daarna checkt hij of dat zijn buurmannen in de check lijst zit zo niet voegt hij ze toe. Mocht hij geen nieuwe nodes in de checklijst zetten dan halt hij uit de check lijst en plaats hem in de compleet lijst en een nieuwe node neem het over.

*1 figuur game het algorithm*



Dit is het resultaat. Het blauwe gedeelte is al gecheckt de gele blokken zitten in de lijst van nog het checken en de witte tegels zijn nog niet gecheckt of zitten in de lijst.

## Het testen/feedback van ander of de maze generator goed werkt.

# Links

https://weblog.jamisbuck.org/2011/1/27/maze-generation-growing-tree-algorithm.html

https://www.youtube.com/watch?v=OutlTTOm17M&t=1486s