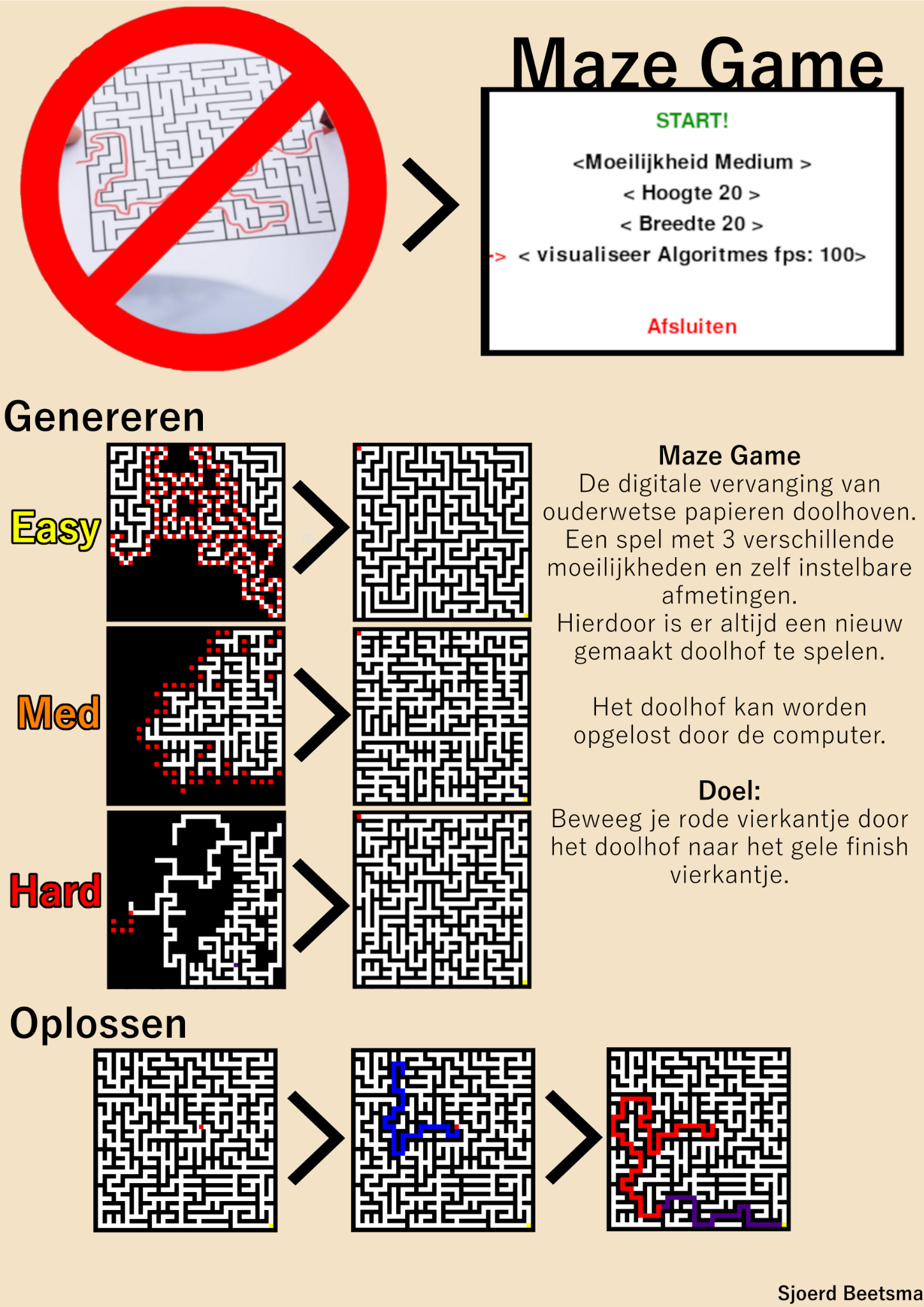
I-PASS Rapportage Maze Game  
 Sjoerd Beetsma 1789293  


[Probleembeschrijving](#_7x0vt5lbd5rr) 3

[Probleemoplossing & eisen](#_99jy8qmhbtw0) 3

[Samenvatting](#_8jrr9jeo5a7k) 3

[Genereer algoritmes](#_h2lgm8mqou1q) 4

[Grafische implementatie](#_l5pgfs1l06nw) 11

[Bronnen](#_wrcjk35p842) 11

# **Probleembeschrijving**

Door de huidige pandemie heerst er meer eenzaamheid en verveling onder de mensen, met name de ouderen. Mijn grootmoeder heeft hier ook erg last van. Ze houdt zich vooral bezig met het oplossen van puzzeltjes via de krant of een puzzelboek. Het nadeel van fysiek puzzel materiaal is dat ze er snel doorheen gaat en vervolgens moet wachten op een nieuwe editie. Om deze reden heb ik een voorstel gedaan voor het ontwikkelen van een digitale oplossing.

# **Probleemoplossing & eisen**

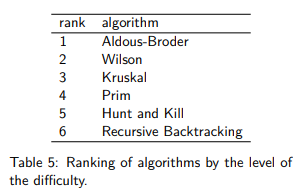
De oplossing voor het probleem van het snel opraken van fysiek puzzel materiaal, is het ontwikkelen van een spel applicatie die labyrint puzzels genereert. De door mijn opdrachtgever gestelde eisen zijn dat het spel makkelijk te starten en besturen is en dat het spel kan worden opgelost door de computer. Aan de hand hiervan zijn een aantal eisen gesteld. Ten eerste moet er een eenvoudig hoofdmenu zijn waarin de afmetingen instelbaar zijn. Daarnaast moet er de mogelijkheid zijn om uit drie verschillende moeilijkheidsgraden te kiezen waarbij elk niveau een ander genereer algoritme vertegenwoordigt. Zo zal er altijd een nieuw doolhof gegenereerd worden. Ook moet er een functionaliteit zijn om de computer het spel te laten oplossen met behulp van een padvinders algoritme. Tot slot is het belangrijk dat het beroepsproduct vloeiend in gebruik is, dus dat de algoritmes geen vertragingen in de spelervaring veroorzaken.

# **Projectsamenvatting**

Er is gestart het implementeren van vijf doolhof genereer algoritmes volgens de moeilijkheid tabel (zie Table 5). Op basis van een efficiëntie analyse (zie Figuur 6,7 en 8) is er een definitieve keuze gemaakt voor drie genereer algoritmes. Er is gekozen voor Depth First Search, Prim en een hybride van Aldous-Broder en Wilson. Vervolgens zijn twee padvinder algoritmes geïmplementeerd om een gegenereerd doolhof te laten oplossen door de computer. Breadth First Search en Depth First Search zijn geïmplementeerd om uiteindelijk één definitieve keuze te maken. Uit de efficiëntie analyse is gebleken dat Depth First Search het efficiëntst werkt en is daarmee de definitieve keuze geworden (zie Figuur 11, 12 en 13).

# **Genereer algoritmes**

Gebaseerd op de moeilijkheid tabel uit het artikel van Gabrovšek (2019) zijn de volgende genereer algoritmes gekozen voor implementatie: Depth First Search, Prim, Aldous Broder, Wilson en een hybride van Aldous Broder en Wilson. In de tabel (zie Table 5) zijn verschillende algoritmes gerangschikt op basis van de volgende factoren: aantal intersecties, aantal doodlopende paden en de tijd voor het oplossen van het doolhof door padvinder algoritmes.

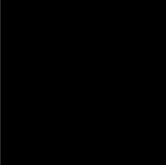
Opmerkelijk is dat blijkt dat Aldous-Broder en Wilson ongeveer gelijke aantallen intersecties en doodlopende paden heeft. Dit komt doordat dit de enige twee algoritmes zijn die een *uniform spanning tree* genereren (Gabrovšek, 2019). Een *uniform spanning tree* houdt in dat elk doolhof in de set van alle mogelijke doolhoven van *« n »* formaat evenveel kans heeft om gegenereerd te worden.

Voor het niveau ‘hard’ zijn zowel Aldous Broder als Wilson geïmplementeerd omdat deze algoritmes beide een *uniform spanning tree* oplevert. De definitieve keuze is gemaakt na een efficiëntie analyse. Voor het testen van de genereer algoritmes is er naast een efficiëntie analyse ook een unit test gemaakt, deze test alle algoritmes op of ze een perfect doolhof genereren.

**Maze object**

Elk genereer algoritme start met een Maze object. Dit object heeft een grid/matrix van een gegeven formaat welke vol start met 1’s (dichte/muur cel). De genereer algoritmes kerven hier een doolhof uit door 1’s in 0’s te veranderen (dichte cel naar open cel). Op het grid zijn de cellen op de oneven kolommen en rijen normale cellen waaruit bewogen wordt en even kolommen en rijen zijn de muur cellen die gesloopt kunnen worden.

**Depth first search (DFS)**

Time complexity O(n)  
Space complexity O(n)  
  
Het Depth First Search algoritme (Kim, 2019) maakt gebruik van een stack waarin het huidige pad wordt opgeslagen. Het algoritme begint bij een willekeurige cel als huidige cel in het grid van cellen en maakt deze open. De computer kiest vanaf dit punt een nog onbezochte buurcel. Tussen de huidige cel en de buurcel wordt de muur gesloopt en worden de stappen herhaald vanaf de buurcel.

Als het algoritme op een cel komt waar geen onbezochte buurcellen aanwezig zijn, ook wel een doodlopend einde, gaat het algoritme een stap terug in de stack.

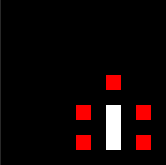
Als alle cellen bezocht zijn, of de stack leeg is, dan is het algoritme klaar.  
Het algoritme heeft een voorkeur om zo diep mogelijke paden te maken totdat er een doodlopend eind is. Om deze reden is er een bias voor diepe doolhof paden.

Voor een demonstratie van het Depth First Search algoritme zie Figuur 1.

**Prim**

Time complexity O(n)

Space complexity O(n)



Het Prim algoritme (Kim, 2019) maakt gebruik van een set ‘frontier’ waarin alle buurcellen zijn opgeslagen van cellen die bezocht zijn.

Het algoritme kiest een willekeurige cel uit het grid van cellen, maakt deze open en voegt zijn buren toe aan de frontier. Vanaf dit punt kiest en verwijdert het algoritme een willekeurige cel uit de frontier, verbindt deze met een cel die al open is en voegt de buurcellen toe aan de frontier. Dit wordt herhaald totdat de frontier leeg is.

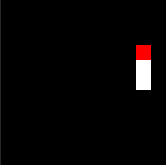
Het algoritme groeit vanaf het startpunt naar buiten in alle richtingen en heeft een bias voor doolhoven met doodlopende einden.

Voor een demonstratie van het Prim algoritme zie Figuur 2.



**Aldous Broder**Time complexity O(∞)

Space complexity O(n)

Het Aldous-Broder algoritme (Kim, 2019) werkt volgens erg eenvoudige regels. 

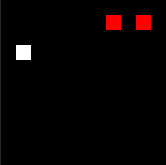
Het algoritme kiest een willekeurige cel als startpunt en maakt deze cel open en bezocht. Vanaf deze cel wordt een willekeurige buurcel gekozen. Als deze buurcel onbezocht is wordt deze opengemaakt en de muur gesloopt. Dit proces herhaalt zich vanaf de buurcel totdat alle cellen bezocht zijn.  
  
Het algoritme beweegt zich willekeurig over het grid heen en kerft een pad uit op punten waar hij nog niet is geweest. Aldous-Broder genereert een *uniform spanning tree* waardoor er geen bias aanwezig is. Door de willekeurigheid van dit algoritme kan het genereren relatief lang duren. Aan het begin van het algoritme is de kans groot dat onbezochte cellen worden bezocht. Naarmate er meer cellen worden bezocht wordt de kans om op een onbezochte cel te stappen steeds kleiner.

Voor een demonstratie van het Aldous Broder algoritme zie Figuur 3.

**Wilson**

Time complexity O(∞)

Space complexity O(n)  


Het Wilson algoritme (Kim, 2019), ook wel ‘loop erased random walk’ genoemd, start met het openen van een willekeurige cel en markeert deze als bezocht. Vanaf een andere willekeurige onbezochte cel zal het Wilson algoritme een willekeurig pad bewandelen. Dit pad wordt onderdeel van het doolhof wanneer het in een cel loopt die al onderdeel is van het doolhof. Er wordt een nieuw startpunt gemaakt om vanaf daar een nieuw willekeurig pad te bewandelen. Deze stappen herhalen zich totdat het doolhof geen onbezochte cellen meer heeft. Als het pad tegen zichzelf aanloopt zal het pad worden afgeknipt vanaf het punt waar die een loep maakt.

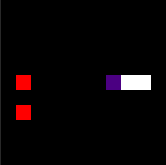
Wilson genereert een *uniform spanning tree*, waardoor er geen bias aanwezig is. Door de willekeurigheid van dit algoritme kan het genereren relatief lang duren. Aan het begin van het algoritme is de kans klein dat een bezochte cel wordt gevonden. Naarmate er meer cellen worden bezocht wordt de kans om op een bezochte cel te stappen steeds kleiner.

Voor een demonstratie van het Wilson algoritme zie Figuur 4.

**Wilson & Aldous broder hybride**

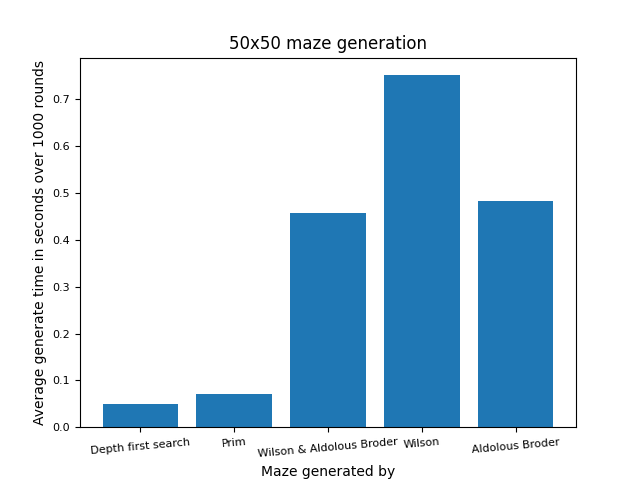
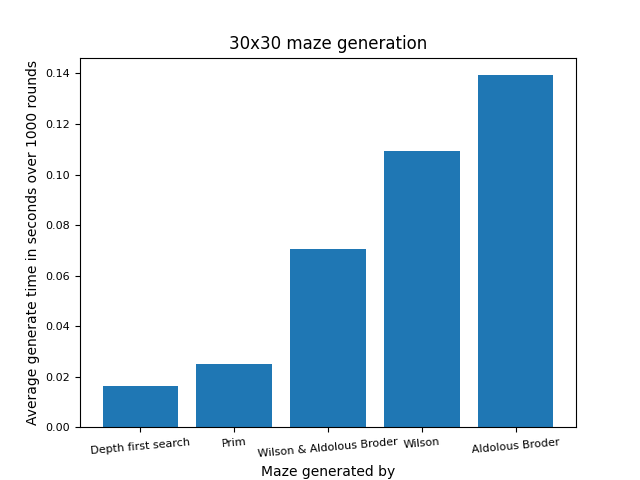
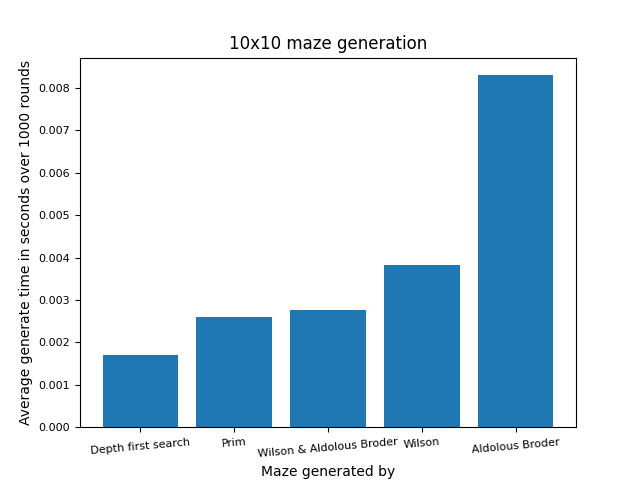
Time complexity O(∞)

Space complexity O(n)

.   
Het algoritme is geïmplementeerd als extensie op Wilson, met toevoeging dat op het eerste willekeurige punt in Wilson een random walker wordt gebruikt. Hierdoor wordt het gebied van onbezochte cellen vanaf het begin vergroot volgens de regels van Aldous-Broder. Daarnaast geldt de regel om het pad van Wilson te vermijden. Hiermee is het voordeel van Aldous-Broder dat er vanaf het begin cellen worden open gemaakt. Tevens is het voordeel van Wilson dat hoe meer cellen er bezocht zijn hoe groter de kans wordt dat er weer cellen/paden worden toegevoegd aan het pad. 

Voor een demonstratie van het Wilson & Aldous Broder hybride algoritme zie Figuur 5.

**Tijd analyse en keuze:**

De definitieve keuze van de genereer algoritmes voor de verschillende moeilijkheidsniveaus zijn gemaakt op basis van een generatietijd test (zie Figuur 6, 7 en 8). Zoals aangetoond in de bar grafieken zijn Depth First Search en Prim efficiënte algoritmes en zijn deze daarom definitief gesteld voor de niveaus ‘easy’ en ‘medium’. Er is met name gekeken naar het verschil tussen Wilson, Aldous Broder en het hybride algoritme. Uit de analyse (zie Figuur 6, 7 en 8) blijkt dat de hybride versie een grotere efficiëntie laat zien dan de andere algoritmes die een *uniform spanning tree* vormen.   
Het verschil tussen Aldous-Broder en de hybride versie wordt kleiner naarmate het doolhof groter wordt. Dit is een resultaat van de regel waarbij het pad van Aldous-Broder in de hybride niet door het pad van Wilson kan gaan. Het hybride algoritme is definitief gemaakt voor het niveau ‘hard’.

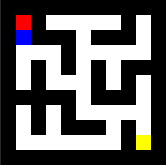
**Padvinder algoritmes**

**Maze object (update):**

Voor het implementeren van de padvinder algoritmes is er een extra functionaliteit aan het Maze object toegevoegd om zichzelf vanuit een grid te kunnen representeren als een geadjungeerde lijst. Vanuit een geadjungeerde lijst is het eenvoudig om een padvinder algoritme te implementeren. Breadth First Search is geïmplementeerd voor het vinden van de oplossing. Dit algoritme werkt gelijk aan Dijkstra’s algoritme op een ongewogen graaf.  
Na het implementeren van het eerste algoritme is er ook besloten om een Depth First Search (DFS) algoritme te implementeren. Dit algoritme vindt geen kortste pad per definitie. Echter is dit geen probleem omdat alle genereer algoritmes perfecte doolhoven generen. Een perfect doolhof heeft maar één oplossing, en die oplossing is tevens dus het kortste pad.  
  
**Breadth first search (BFS)**

Time complexity O(n)

Space complexity O(n)  

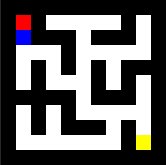

Het Breadth First Search algoritme (Babula, 2009) maakt gebruik van een queue, ook wel een ‘rij’ van cellen in de volgorde van wanneer ze aan de beurt zijn.

Vanaf een gegeven startpositie start het algoritme en gaat door zolang de oplossing niet gevonden is en/of de queue niet leeg is.  
Het algoritme zet de afstand van het startpunt op 0 en voegt zijn buren toe aan de queue.

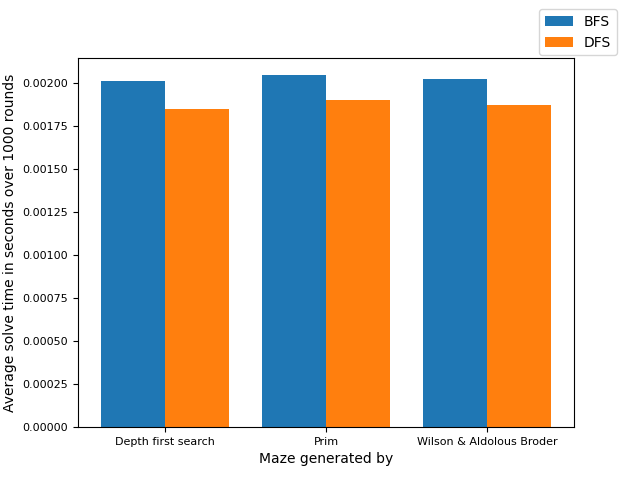
Van elke cel in de queue wordt de afstand geupdate door de afstand van de vorige buurcel te nemen +1, voegt zijn buren toe aan de queue en verwijdert zichzelf uit de queue. Dit herhaalt zich totdat de queue leeg is of het einde gevonden is. Wanneer het eind gevonden is gaat het pad vanaf het eind terug naar het startpunt via de cellen met de kortste afstand, dit pad wordt dan opgeslagen als de oplossing.  
  
**Depth first search (DFS)**

Time complexity O(n)

Space complexity O(n)

Het Depth First Search algoritme (Babula, 2009 ) maakt gebruik van een stack waarin het huidige onderzochte pad in wordt opgeslagen. Vanaf een gegeven startpositie start het algoritme en voegt deze toe aan de stack. Zolang het einde niet gevonden is of de stack niet leeg is wordt de huidige cel het meest recente element in de stack en beweegt het pad zich vanaf deze cel naar een willekeurige nog niet bezochte buurcel.  
Wanneer het pad in een doodlopend eind komt of er geen niet bezochte buren zijn gaat het pad terug in de stack tot een punt waar weer naar een nog niet bezochte buurcel kan worden bewogen. Dit herhaalt zich totdat het eind is gevonden. De huidige stack is het pad naar het eind van het doolhof.  
  


**Tijd tests en keuze**

Om de oplossingstijd van BFS en DFS te vergelijken zijn beide algoritmes getest door 1000 mazes van elk genereer algoritme te laten oplossen door de BFS en DFS. De uitslag is de gemiddelde genereer tijd in seconden.  
Zoals aangetoond in de bar grafieken (zie Figuur 11, 12 en 13) is het DFS in de meeste gevallen sneller dan BFS.

Ook als 2 algoritmes dezelfde time en space complexiteit hebben zal in de meeste gevallen BFS meer geheugen in gebruik nemen dan DFS. Dit omdat BFS alle mogelijke paden van de huidige breedte opslaat, in tegenstelling tot DFS welke één pad tegelijk opslaat.   
Met deze analyse in gedachten is ervoor gekozen om DFS als oplos algoritme te gebruiken in het spel.

# 

# **Grafische implementatie**

Om het Maze object speelbaar te maken en grafisch te representeren is er gebruik gemaakt van de module PyGame.  
Het ontwerp bestaat uit een Player, Game en Menu klassen.

In de Menu klasse is een state machine geïmplementeerd om de instellingen van het game object aan te passen en het spel te kunnen opstarten/afsluiten.

Deze instellingen zijn:

-Moeilijkheid

-Hoogte

-Breedte

-Animeren van de gebruikte algoritmes met instelbare FPS

Met twee knoppen:  
-Start

-Afsluiten  
  
Wanneer er op de start knop wordt gedrukt zal er een doolhof worden gegenereerd met de huidige instellingen. Het gegenereerde doolhof wordt vervolgens in het spel object vertaald naar Pygame objecten en een Player wordt aangemaakt. Elk frame wordt gecontroleerd op botsingen tussen de speler en de muren om te voorkomen dat een speler door muren heen kan lopen. Ook wordt er gecontroleerd op een botsing tussen de speler en finish. Wanneer de speler botst met de finish is het spel over en wordt de state veranderd naar het hoofdmenu.

# **Bronnen**

-Gabrovšek, P (2019). [Analysis of Maze Generating Algorithms](http://ipsitransactions.org/journals/papers/tir/2019jan/p5.pdf), IPSI Transactions on Internet Research, Vol. 15, No. 1  
-Kim, P. (2019). [Intelligent Maze Generation](http://rave.ohiolink.edu/etdc/view?acc_num=osu1563286393237089). (Electronic Thesis or Dissertation).  
-Babula, M. (2009). [Simulated maze solving algorithms through unknown mazes](http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.215.2983&rep=rep1&type=pdf#page=21). Organizing and Program Committee, 13.