ikdoeict.be

# Advanced Applied Programming
## 2017 - 2018

Katja Verbeeck
Joris Maervoet
Tim Vermeulen

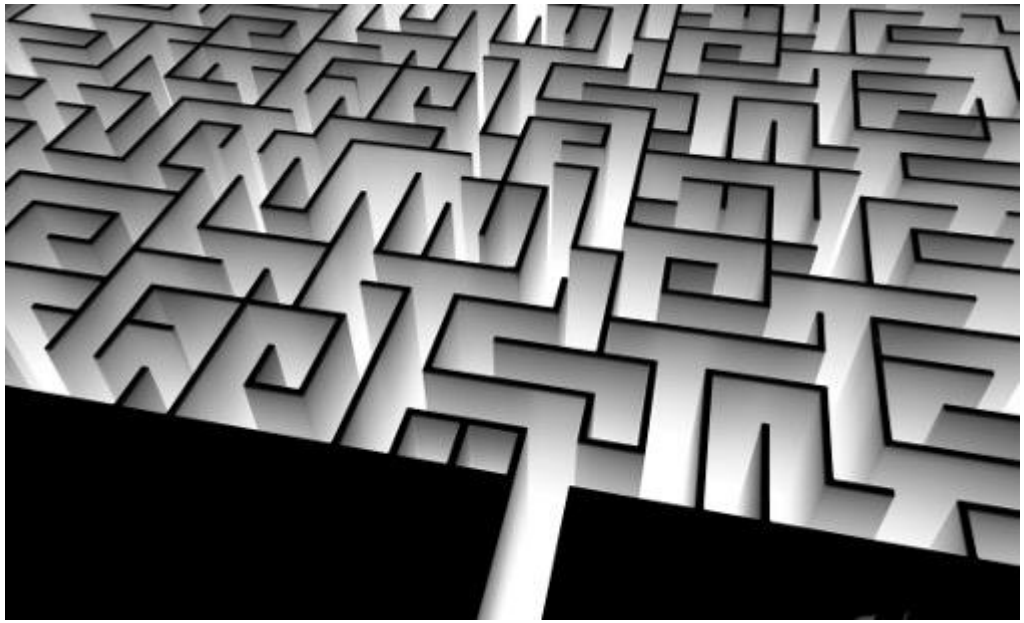
# Game AI

# Inhoud

- **opdracht deel 3**
- **(light) Intro in AI**
- **Game AI :**
  - Intro : rules and states
  - Path planning
  - Graph algo & maze generation

# Opdracht deel 3

Schrijf een algoritme waarmee je telkens opnieuw een ander **perfect 2D labyrinth** kan genereren en visualiseer ook telkens het kortste pad van een start- naar een eindpunt.



# Maze generation

**Voorbeeld :**

https://www.youtube.com/watch?v=6kv5HKPB1XU

https://www.youtube.com/watch?v=NZEoCFrETNU

# Intro Artificial Intelligence

•**Kunstmatige intelligentie (KI) of artificiële intelligentie (AI) is de wetenschap die zich bezighoudt met het creëren van een artefact dat een vorm van intelligentie vertoont.**

•**Wanneer is iets intelligent?**

→ **Turing Test ,** ***Alan Turing,***
***Computing Machinery and Intelligence, 1950***

•**Computationele Intelligentie:**

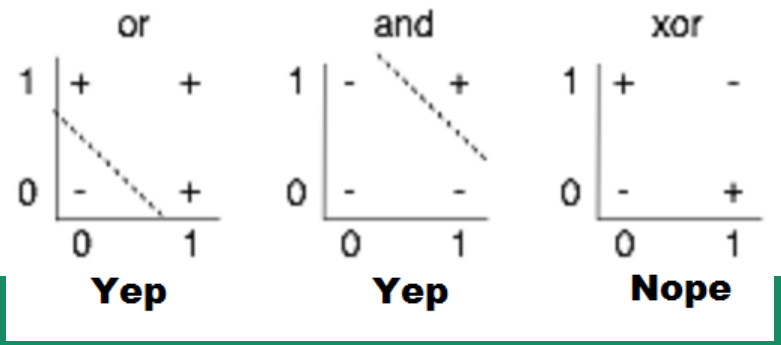
–Computationele technieken gebruiken
**om natuurlijke intelligente**
**phenomenen te modelleren**
→ **machine leertechnieken**

–Intelligente software → **Software Agenten**

# Beetje geschiedenis

- **1940 Programeerbare digitale computers**
- **1943 – 1956 : McCulloch & Pitts : het brein modelleren als een booleaans circuit**
- *Dartmouth Summer Research Project on Artificial Intelligence* (DSRPAI) hosted by John McCarthy and Marvin Minsky in 1956. eerste AI meeting: **ontstaan van de naam AI en presentatie van het eerste AI programma : Logic theorist**
- **1956 – 1974 : de gouden jaren**

  **Oa. general problem solvers and ELIZA (first chatbot!)**
- **AI Winter : de XOR functie kan niet gemodelleerd worden**



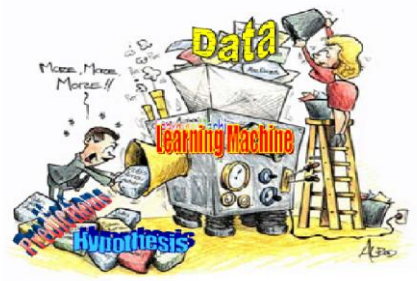

Visualization of the limitations of Perceptrons. Finding a linear function on the inputs X,Y to correctly ouput + or - is equivalent to drawing a line on this 2D graph separating all + cases from - cases; clearly, for the third case this is impossible.

http://www.andreykurenkov.com/writing/a-brief-history-of-neural-nets-and-deep-learning/

# Beetje geschiedenis

- Jaren 80
  - Ontstaan van tak machine learning : Neurale netwerken (met gebruik van hidden layers) worden weer populair
  - Expert systemen
- 1987 – 1993 : tweede AI winter – wachten op computational power
- 1995 AI wordt als aparte wetenschap gezien
  - Integratie van leren, redeneren, kennisrepresentatie
  - Toepassingen in taal, beelden, data mining
- 2003 – 2007 DARPA grand challenge
- 2005 ASIMO
- 2006 Netflix prize
- 2009 google selfdriving car
- 2010 Kinect
- 2010 deep-learning
- 2011 IBM Watson
- 2011 Siri on the iPhone4S
- 2012 google brain learns the concept of a cat
- 2016 AlphaGo &
- 2017 AlphaGoZero (learning by selfplay)

What do you mean? Machines can they learn?




(from Eric Xing lectures on ML)

# Deep-Blue

**1997 Deep Blue de eerste schaak computer die de toenmalige wereld kampioen Garry Kasparov versloeg. Deep Blue kon 200 miljoen posities doorzoeken per seconde, maakt hierbij gebruik van gesofisticeerde evaluatie functies en (undisclosed) zoekmethoden om ongeveer 40 zetten vooraf te 'denken'**



## Mars Exploration Rover

De Mars Exploration Rovers zijn twee onbemande ruimtevaartuigen die in juni 2003 door NASA naar Mars zijn gestuurd en daar in januari 2004 landden. Wikipedia

**Startdatum:** 7 juli 2003

De twee **Mars Exploration Rovers** (Marsverkenningsrobotwagentjes) zijn genaamd de Spirit (of MER-A) en de Opportunity (of MER-B). Het contact met de Spirit is sinds 22 maart 2010 verloren en op 25 mei 2011 heeft NASA deze missie beëindigd. In juli 2014 had de Opportunity 40 kilometer afgelegd.

## Mars Rover

- Directe remote controle is niet mogelijk
  - Het duurt + / - 10 minuten om een bericht te sturen vanop aarde naar Mars
  - Het is niet mogelijk om met voldoende nauwkeurigheid te voorspellen wat de robot zal tegenkomen in zijn omgeving.
  - → De robot moet autonoom kunnen werken!

# DARPA challenge




**Robotic Control , DARPA (URBAN) Challenge**

2004 : the Mojave Desert for 150 miles (240 km) Geen enkel van de automatische voertuigen was in staat om de finish te bereiken. CMU's Red Team kon als enige 11.78 km autonoom afleggen

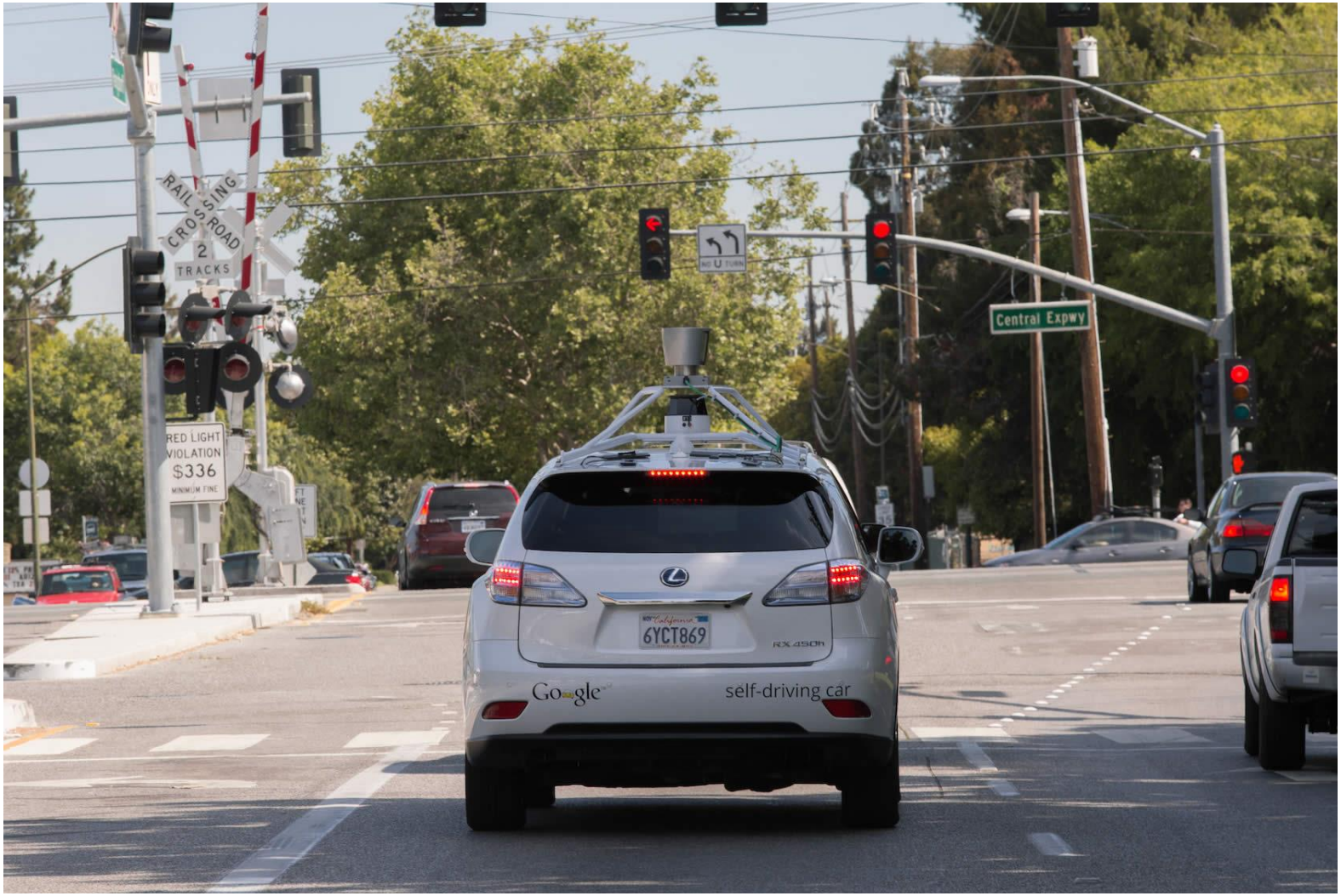
2005 212 km (132 mi) off-road course. Deze keer konden 22 van de 23 finalisten de afstand van 2004 evenaren, 5 wagens konden succesvol het volledige parcours afleggen.

2007 Deze keer moest er 96 kilometer afgelegd worden in een stedelijk gebied in minder dan 6u (60 mi) . De wagens moesten zich houden aan de verkeersregels en onderhandelen met andere voertuigen en obstakels.

# Google CARS

https://www.google.com/selfdrivingcar/


2009 de autonome Toyota Prius op een snelweg in California.




**https://www.google.com/selfdrivingcar/**

2013 de focus ligt nu op stadsverkeer, een veel complexere omgeving dan de snelwegen.

**February 11, 2013 : Google wil autonome voertuigen in de straten op en termijn van 3 a 5 Jaren.**


Proefprojecten  in verschillende steden met een zelf ontworpen voertuig
https://www.google.com/selfdrivingcar/

WAYMO

Journey Technology On the Road Early Riders

# Over 5 million miles self-driven

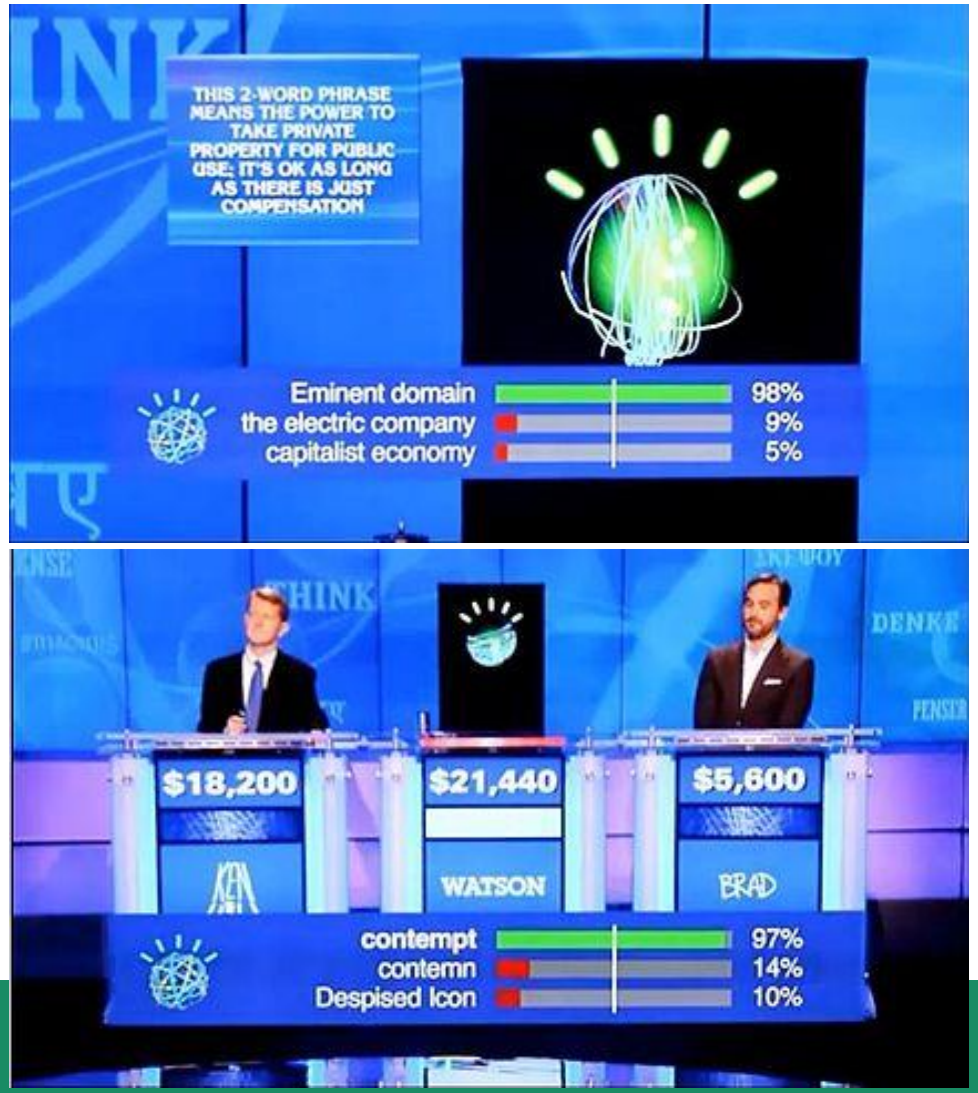
We drive more than 25,000 autonomous miles each week, largely on complex city streets. That's on top of 2.7 billion simulated miles we drove just in 2

Technology Intelligence

## After Uber's fatal accident, is it the end for driverless cars?

share 142

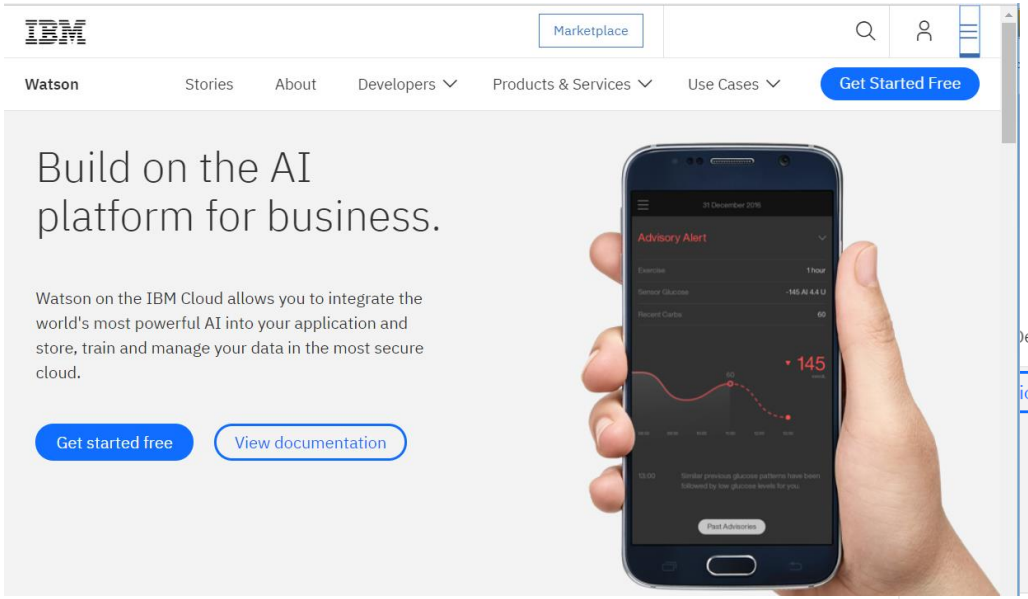
# IBM WATSON

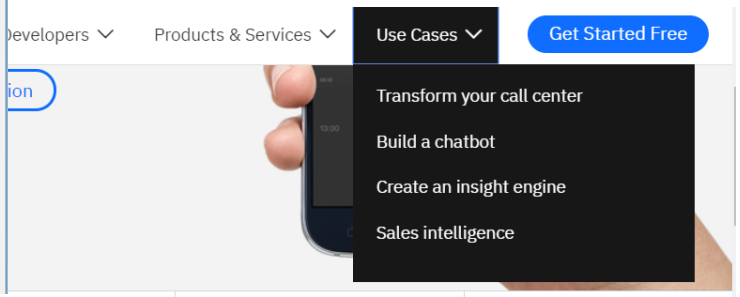



**the Jeopardy!quiz show**

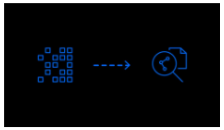
**Het antwoord wordt geprojecteerd, de vraagstelling moet door de kandidaten gevonden worden.**

# IBM Watson



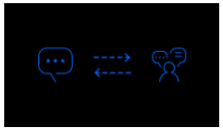




Featured services



Unlock hidden value in data to find answers, monitor trends and surface patterns.

Try Watson Discovery →



Quickly build and deploy chatbots and virtual agents across a variety of channels.

Try Watson Assistant →

Tag and classify visual content using machine learning.

Try Watson Visual Recognition →

# Game AI

## Typisch gebruikt voor :

- Beweging
- Beslissen waarnaartoe te bewegen (path planning)
- Tactische en strategische beslissingen

**“intelligente” simulatie van gedrag (i.e. een uitgedokterde strategie en vooral ook gepaste reacties)**

*The real goal of AI in games is to simulate intelligent behavior, providing the player with a believable challenge that the player can then overcome.*

*[https://software.intel.com/en-us/articles/designing-artificial-intelligence-for-games-part-1/*

# Game AI

- **Ontwikkeld om suboptimaal te zijn**
  - Performantie is niet de hoofdzaak : het problem moet niet optimaal opgelost worden; de heuristische aanpak (vuistregels) is vaak beter en realistischer
- **Ontwikkeld om te entertainen**
  - Het is niet de bedoeling om spelers te frustreren. Goede Game AI moet ook kunnen verliezen, en vooral spelers het gevoel geven dat ze slim zijn en het spel in handen hebben.
- **Ontwikkeld om de illusie van intelligentie op te wekken**
  - Als een speller denkt dat een bot intelligent is, dan is hij dat ook.
  - Is soms simpel : laat een karakter de andere karakters volgen met zijn hoofd
  - Is soms moeilijk: karakters die tegen een muur botsen komen stom over – of karakters die vals spelen (door een muur gaan bv) geven een verkeerde indruk.

# Zoek een systeem om gedrag te representeren en hierin beslissingen te kunnen nemen

## Decision Making Techniques

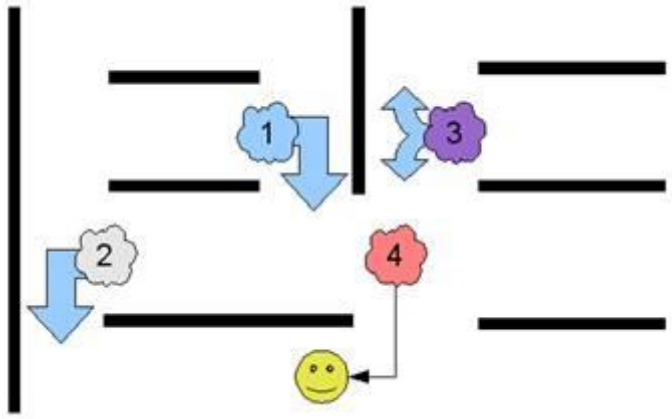
- Decision Trees
- State Machines
- Behavior Trees
- Fuzzy Logic
- Markov System
- Goal-oriented Behaviour
- Rule-Based System
- Blackboard Architecture
- Rule-based System (Scripting)
- ....

## Movement Techniques

- Steering Behaviors (kinematics)
- Path Finding
- Predicting Physics
- Coordinated Motions
- ....

# Rule-based systemen

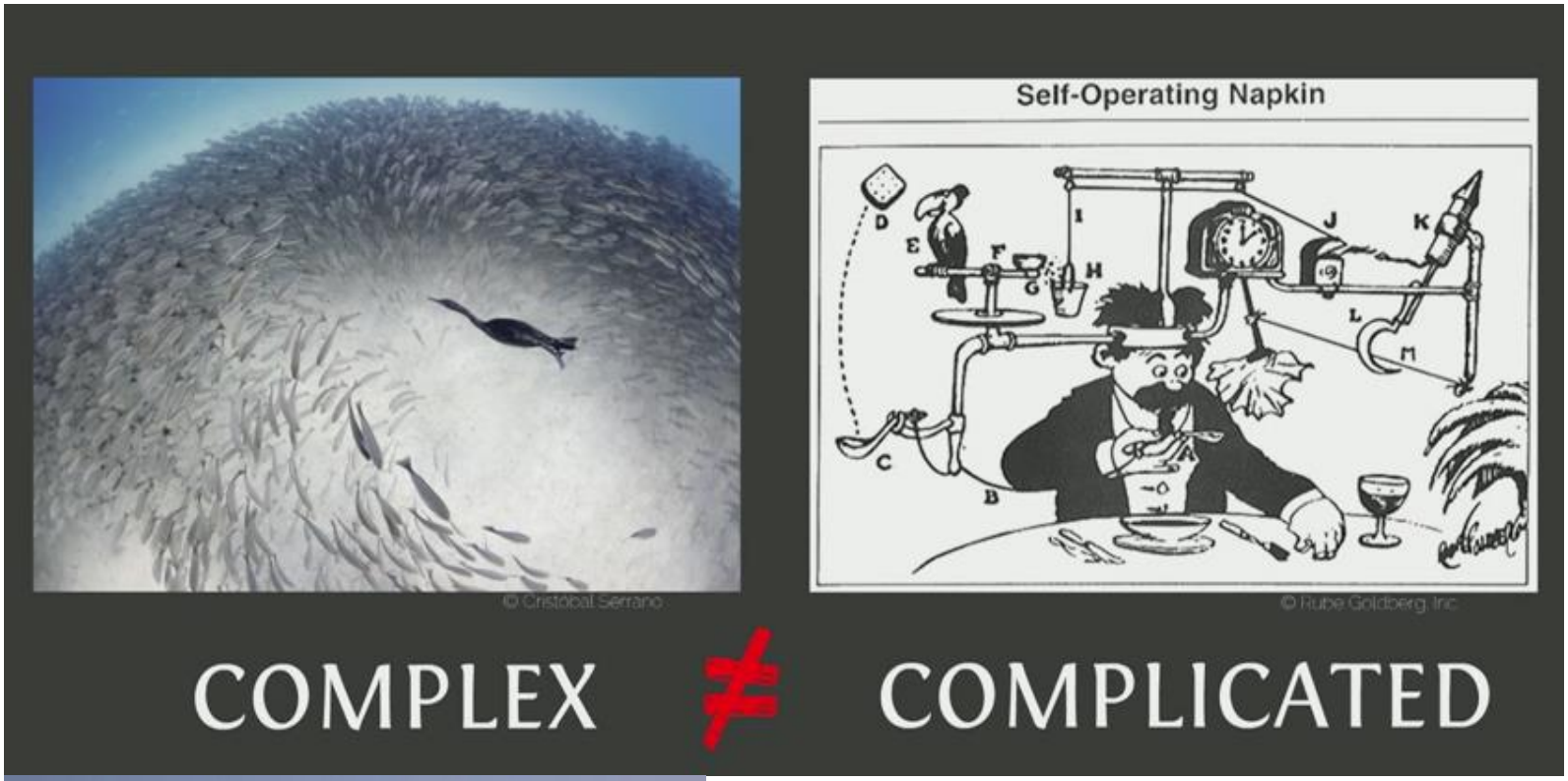
Voorbeeld : de regels gevolgd door *Pac-man ghosts – de pijlen stellen de 'beslissingen' voor die gevolgd worden*




Één spook zal steeds linksafdraaien – een tweede steeds rechtsaf .
Een derde zal een willekeurige richting kiezen.
De vierde beweegt steeds in de richting van de speler.
Elke spook individueel volgt een eenvoudige regel. Maar de groep van spoken lijkt zich complex te gedragen ondanks het feit dat slechts 1 van hen rekening houdt met de positie van de speler.

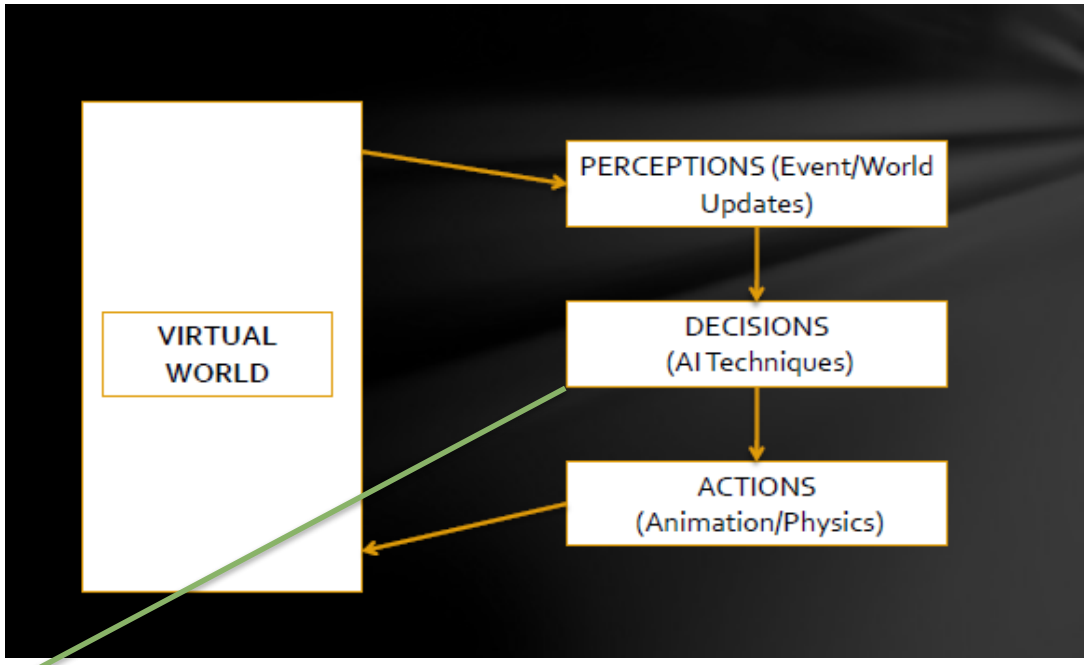
https://software.intel.com/en-us/articles/designing-artificial-intelligence-for-games-part-1/

# Emergentie van complex gedrag




https://social-biz.org/2014/02/10/complex-behavior-emerges-from-simple-rules/

# Model of Game AI




Welke volgende stap moet er gebeuren?

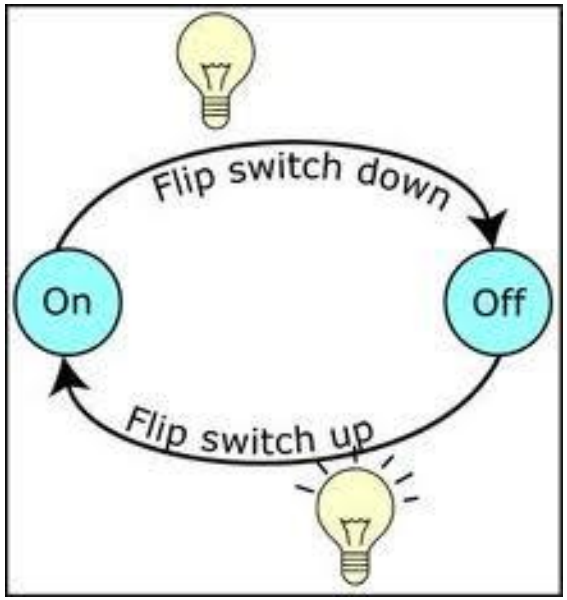
Finite state machines

Hoe daar geraken ?

Path planning

# State Driven agent design - Finite State Machines

**Een eindige toestands-automaat is een model bestaande uit een eindig aantal toestanten waarin een entiteit van het spel zich in kan bevinden op elk moment van het spel. De acties, events of triggers in het spel brengen deze entiteit van de ene toestand in de andere *waarbij een nieuwe actie of output gegenereerd wordt. Op elk moment kan de entiteit zich maar in 1 toestand bevinden. Toestanden kunnen iets fysiek voorstellen maar bvb ook iets emotioneels***





modelleer voorafgedefineerde events van de game

A BABY AS A FINITE STATE MACHINE (PICTURE FROM WWW.ALONGCAMEBABY.CA)

http://jessewarden.com/2012/07/finite-state-machines-in-game-development.html

# Toestanden in voetbalgames


Speler's gedrag kan gedefinieerd worden via toestanden zoals:
***Strike, Dribble, ChaseBall, and MarkPlayer.***

Ook het team zelf kan geimplementeerd worden als een FSM en kan toestanden hebben zoals :
***KickOff, Defend, or WalkOutOnField.***

# Adaptive AI

Niet elke game valt te modeleren met voorafgedefinieerde events.

In dynamische omgevingen gaat men typisch beslissingen uit het verleden bestuderen en deze evalueren om beter te leren beslissingen te nemen in de toekomst (zie bvb decision trees).

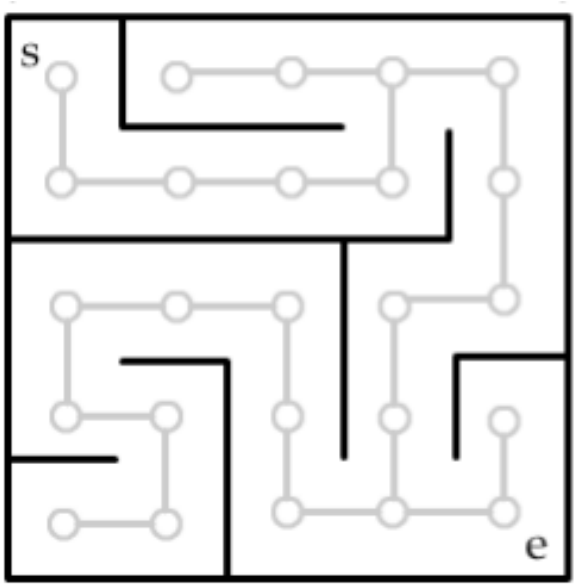
# Path-planning Algorithms

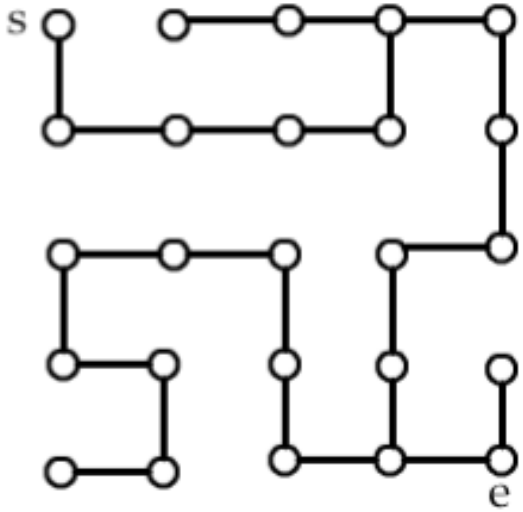
- **path planning =** het zoeken van routes van een startpunt (locatie) naar eindpunt (locatie)
- Hoe spelers van de game naar de juiste positie laten navigeren?
- Heeft uiteraard met beweging te maken maar ook decision making

## 2 types:

- **Map-grid based:**
  - De spelomgeving moet vertaald worden naar een directed niet negatieve gewogen graph : **Dijkstra, A***
- **Sampling gebaseerde path planning** : Rapidly Exploring RandomTrees (RRT) en varianten

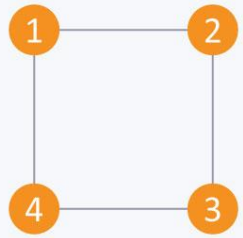
# Grid Graph



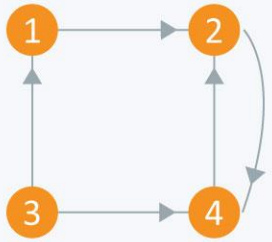




Laat elke cel in de nxn grid (maze) een knoop (vertice) zijn en elke vrije doorgang naar een andere cel een boog (edge). Je bekomt een graph.
Een perfecte maze is eentje zonder loops en waarbij er juist 1 oplossing is van een start naar eindknoop. (in dit geval is de graph gewoon een (spanning) tree)

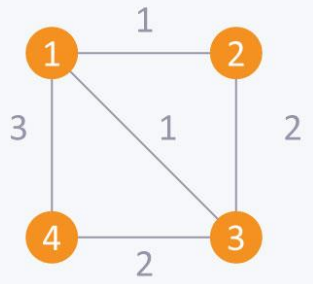
# Graphs



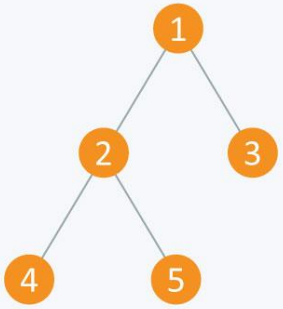

Undirected Graph




Directed Graph

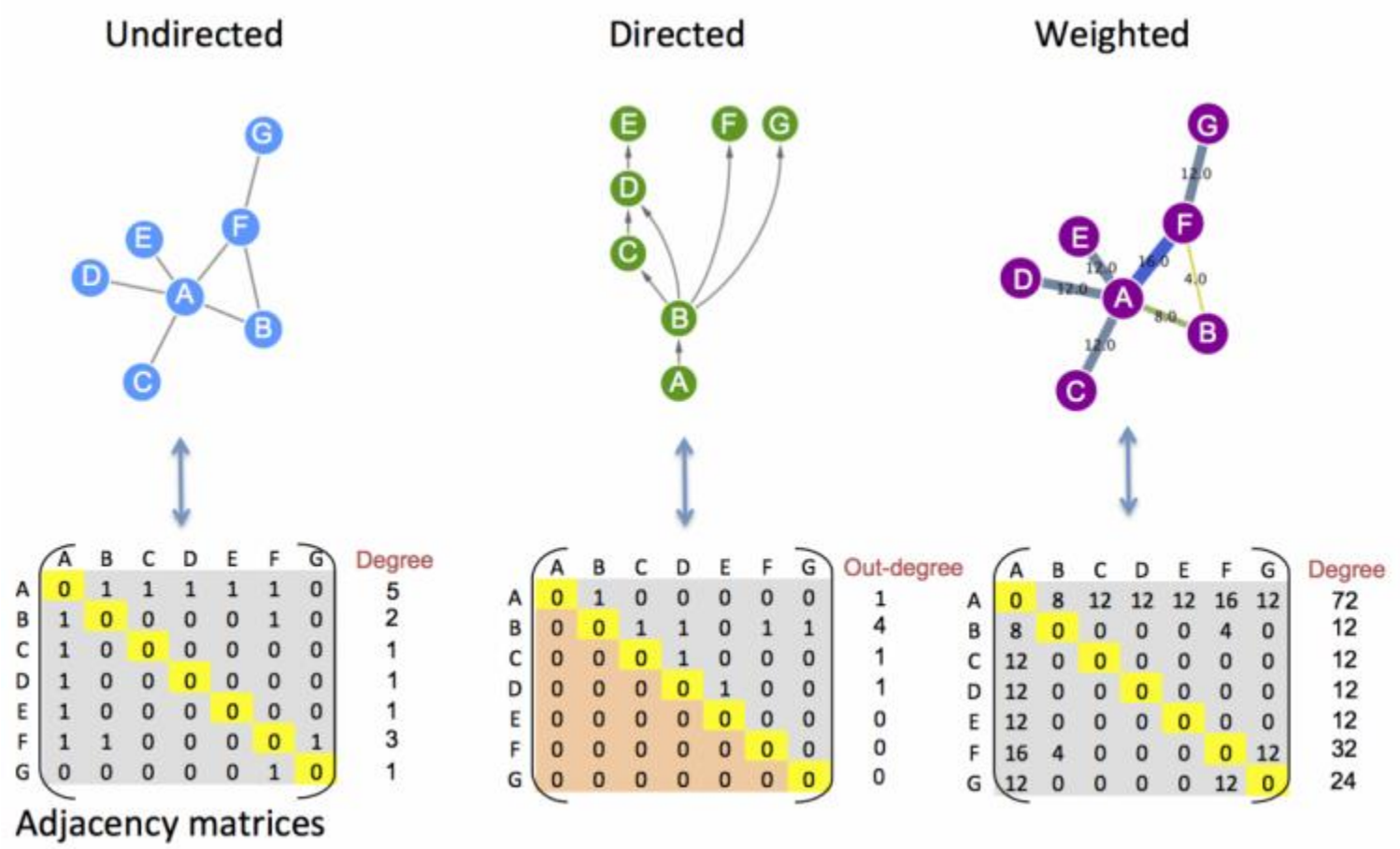



Weighted Graph

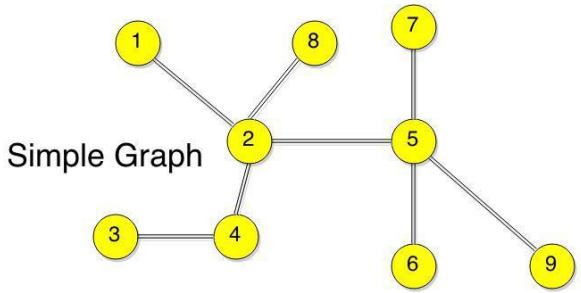



Tree

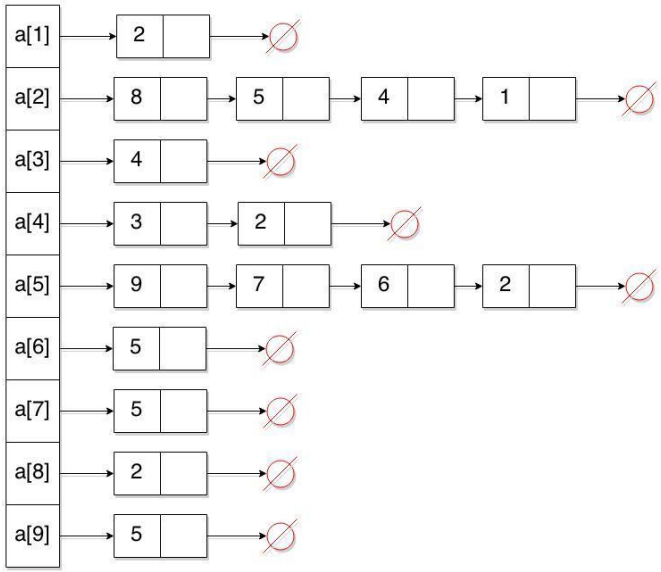
# Implementatie van een graph



# Implementatie van een graph




Adjacency List

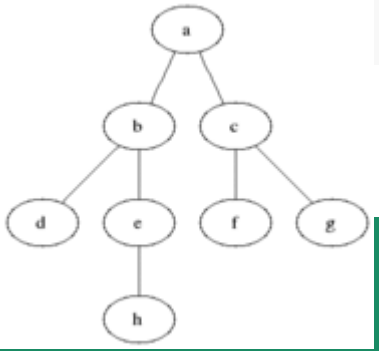


# Doorzoeken van een graph

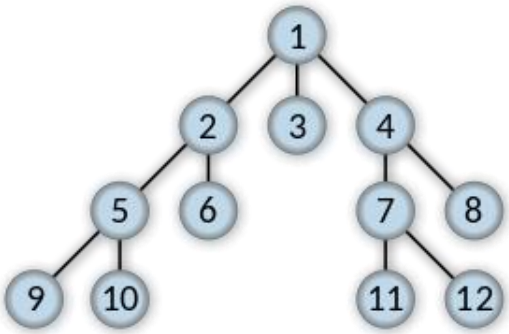
- Elke knoop moet juist 1x bezocht worden
- Elke edge moet juist 1x bezocht worden
- De volgorde waarin knopen/edges bezocht worden is van belang
- Wanneer knopen bezocht worden, worden ze op 1 of andere manier gemarkeerd

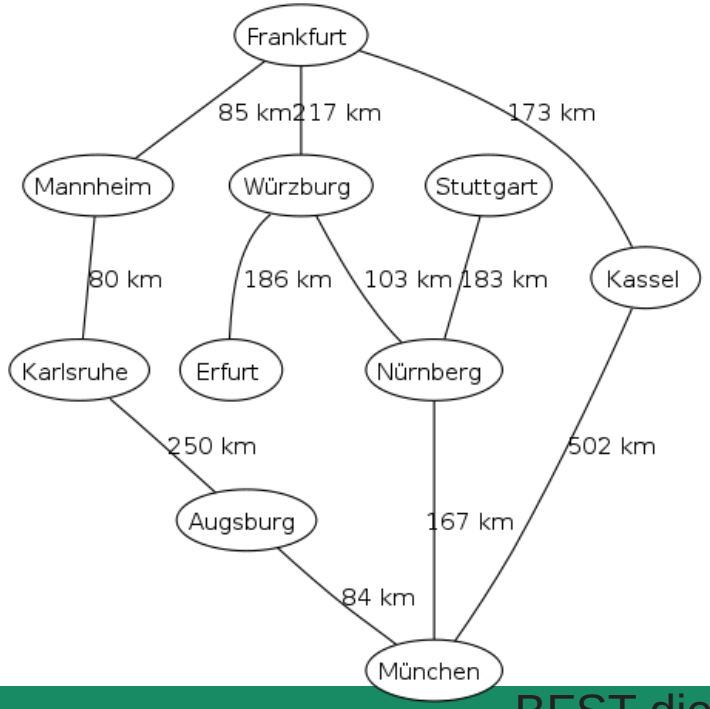
# Doorzoeken van een graph : breadth first

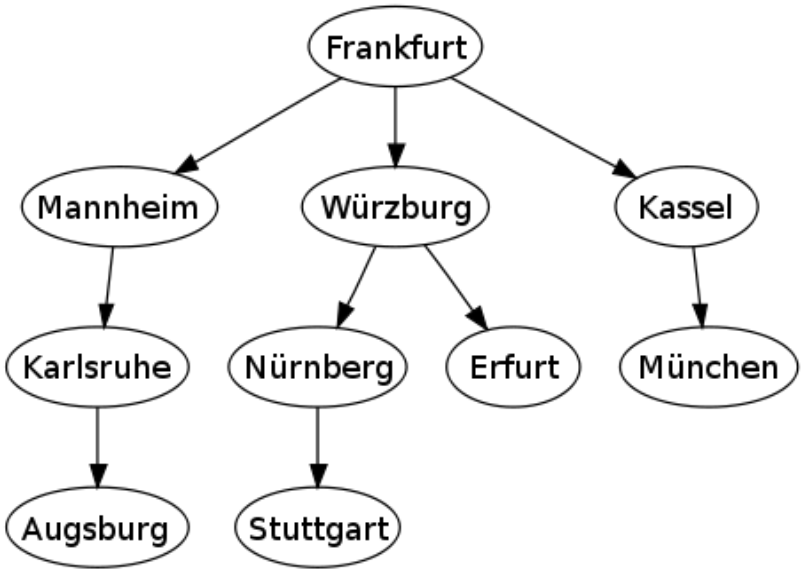
```
BFS (G, s) //Where G is the graph and s is the source node
 let Q be queue.
 Q.enqueue( s ) //Inserting s in queue until all its
                    neighbour vertices are marked.
 mark s as visited.
 while (Q is not empty) //Removing that vertex from queue,whose
                             neighbour will be visited now
       v = Q.dequeue( ) //processing all the neighbours of v
       for all neighbours w of v in Graph G
       if w is not visited
              Q.enqueue( w ) //Stores w in Q to further visit its
                                 neighbour
              mark w as visited.
```



# Depth first tree



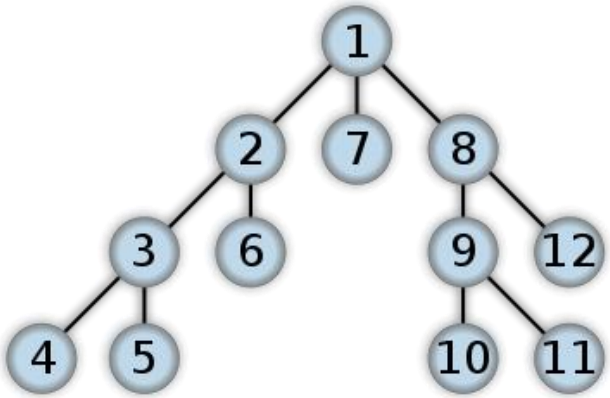



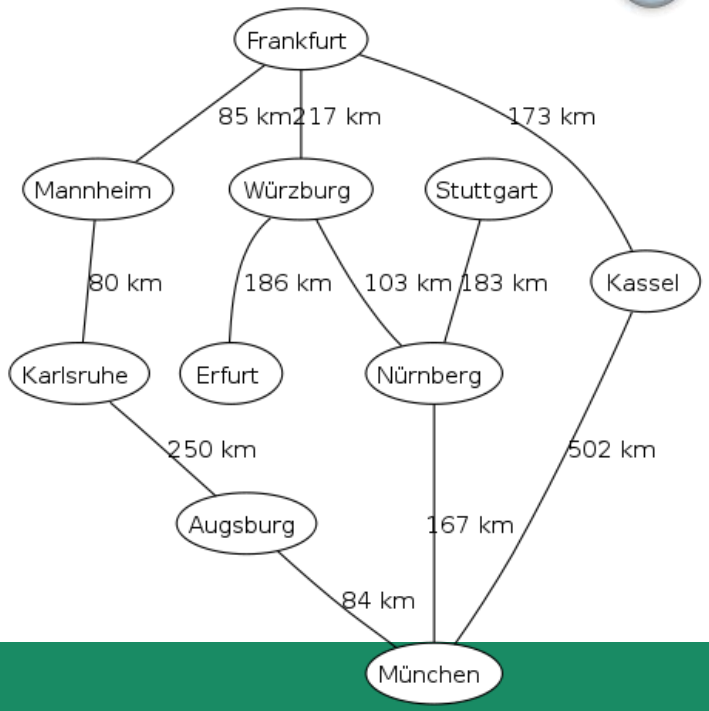

BFST die ontstaat na toepassing van het algoritme op de gegeven kaart met Frankfurt als startknoop.
(https://en.wikipedia.org/wiki/Breadth-first_search)

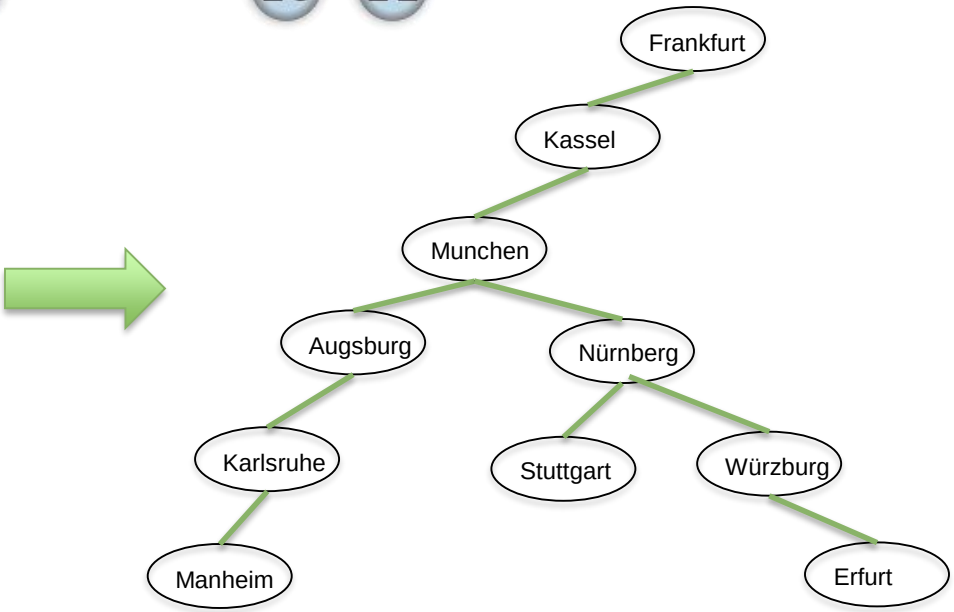
# Doorzoeken van een graph : depth first

```
DFS-iterative (G, s): //Where G is graph and s is source vertex
 let S be stack
   S.push( s ) //Inserting s in stack
   mark s as visited.
   while ( S is not empty): //Pop a vertex from stack to visit
                            next
       v = S.top( )
       S.pop( ) //Push all the neighbours of v in stack that are not
                  visited
       For all neighbours w of v in Graph G:
             if w is not visited :
                   S.push( w )
                   mark w as visited
```

# Depth first tree

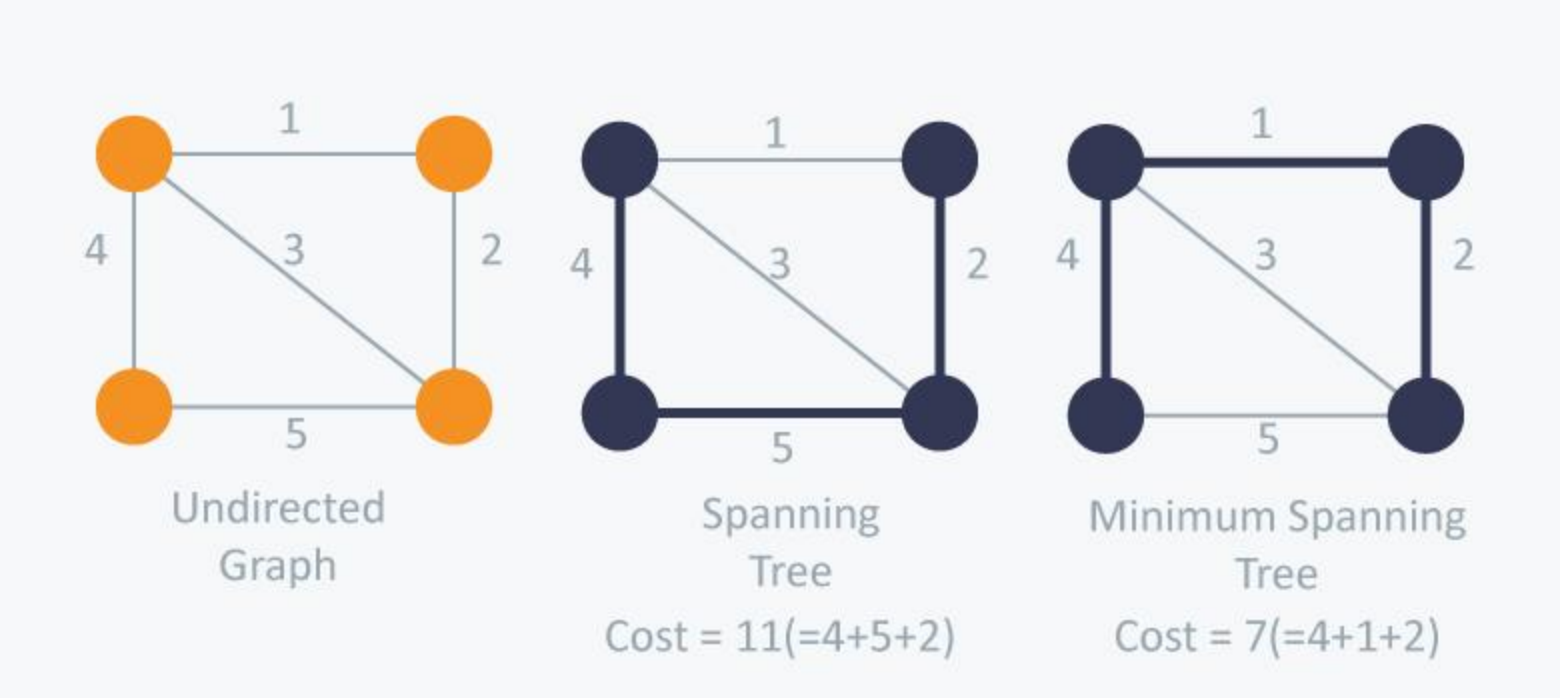


# (minimum) spanning trees

Een spanning tree van een niet gerichte graph is een deelboom uit de oorspronkelijke graph die elke knoop van de graph bevat (elke knoop wordt dus bezocht). Een minimum spanning tree doet dit met een minimale kost (minimum aan edges of som van de gewichten van de edges is minimaal)



# Kruskal's algoritme

edges worden 1 voor 1 toegevoegd aan de spanning tree volgens een greedy manier :

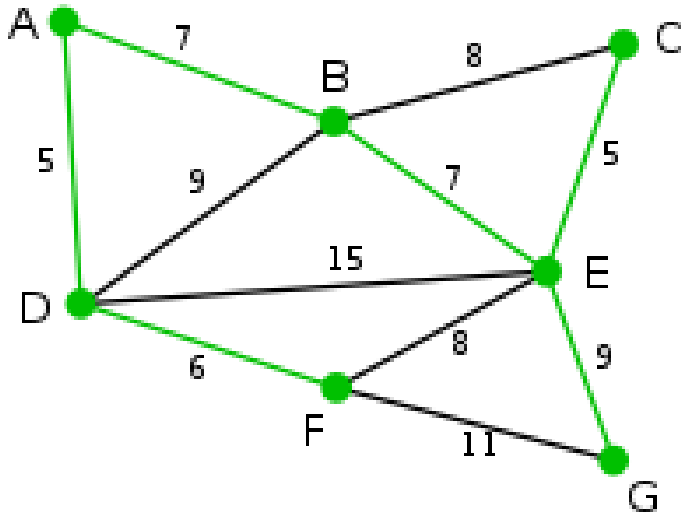
**Algoritme :**

- Sorteer alle edges volgens hun gewicht
- Voeg edges toe aan de MST vertrekkende telkens van deze met het kleinste gewicht
- Let op, alleen edges die een nog niet geconnecteerde knoop toevoegen worden toegelaten (geen loops dus! Werken met disjuncte sets)

# Kruskal's algoritme

```
KRUSKAL(G):
A = ∅
For each vertex v ∈ G.V:
      MAKE-SET(v)
For each edge (u, v) ∈ G.E
      ordered by increasing weight(u, v):
            if FIND-SET(u) ≠ FIND-SET(v):
                  A = A ∪ {(u, v)}
                  UNION(u, v)
return A
```

# Voorbeeld Kruskal




Startknoop D : toegevoegde bogen in volgorde : DA – DF – AB – BE – EC – EG

# Prim's algoritme

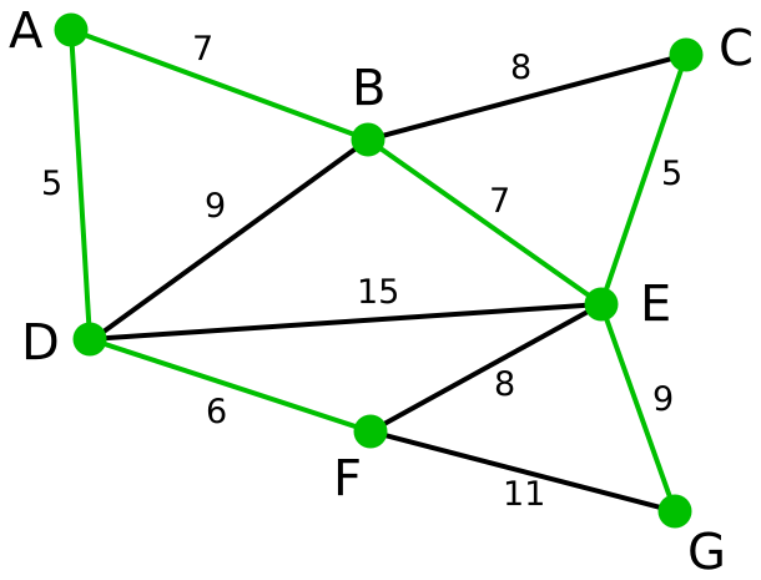
**Ook een greedy algoritme, maar nu worden knopen toegevoegd**

**Algorithm Steps:**

- Werk met 2 disjuncte sets van knopen. De ene set zit al inde MST, de andere nog niet.
- Kies de knoop die op de goedkoopste manier verbonden kan worden met de MST
- Alleen deze knopen die geen cycles veroorzaken kunnen toegevoegd worden.

```
T = ∅; U = { s };
while (U ≠ V)(V is the set of all edges)
       let (u, v) be the lowest cost edge such that
                                        u ∈ U and v ∈ V - U;
              T = T ∪ {(u, v)}
              U = U ∪ {v}
```

# Voorbeeld Prim

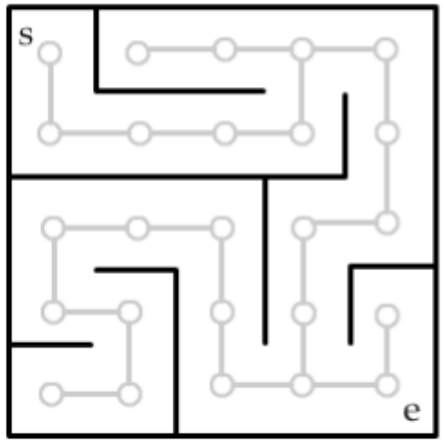


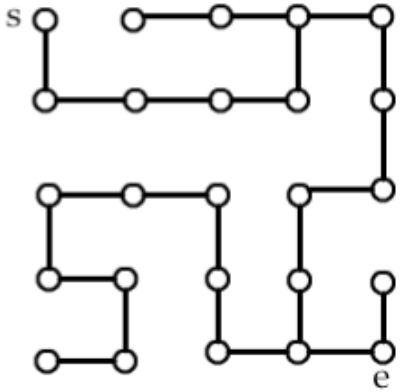

Startknoop D : toegevoegde knopen in de volgorde : A via D – F via D – B via A – E via B – C via E – G via E

# Maze generation

Het resultaat moet een perfecte maze zijn : dit is eentje zonder loops of gesloten circuits of ontoegankelijke gebieden. Vanaf elk punt is er exact één path naar een ander punt.

Een perfecte maze is een spanning tree van de grid



# Mogelijke algoritmen om perfecte mazes te genereren

Recursive backtracker algo

Versies van Kruskal algo

Versies van Prim's algo

Wilson's algo

Hunt and Kill algo

Growing tree algo

Eller's algo

Binary tree mazes

...

**Minstens 1 van deze algoritmen zal je implementeren (bespreken in je verslag) en visualiseren (hoeft geen live visualisatie te zijn zoals in de demo) toon ook het pad tussen start en eindpunt.**

# Reminder

**Deel 1 - Het verslag -**

+ moet een weergave van je onderzoek-stuk zijn –

(hier : hoe interpreteer je het algoritme – welke extra keuzes heb je meegenomen? Heb je algoritmen met elkaar vergeleken?)

+ inzoomen op het uiteindelijke resultaat : d.i.

bespreek je experimenten en de resultaten ervan

+ event. kritische reflectie (future work)

+ Bibliografie en verwijzen vanuit de tekst naar de bibilio

+ maak je tekst leesvriendelijk : werk met extra figuren, tekeningen, een goede structuur, een inhoudstafel, …

# Reminder

**Deel 2 – De Code -**

+ Vervolledig je applicatie in c# - mag in een apart project – maar is uiteraard een plus wanneer je dit kan linken aan je vorige opgaven.

+ Je mag vertrekken van bestaande code maar  niet zondermeer! (verwijzen + begrijpen + je eigen heuristieken erin verwerken)

+ Denk aan de gebruiksvriendelijkheid van je eindresultaat

+ Denk aan de opbouw en structuur van je code (uitbreidbaarheid, herbruikbaarheid, …)

# Reminder

**Deel 3 – De demo –**

**+** De demo vindt plaats tijdens het examen.

**+ Je wordt in groepen ingedeeld en je kiest via tolinto zelf het uur op de juiste dag (alleen binnen de groep waaraan je bent toegewezen)**

**+** Bereid je demo voor ! Denk na over hoe je op een efficiënte manier ons kan overtuigen van je product !

# Deadlines

**Opdracht 1+ 2 : 3D + fysica : vrijdag 27 april**

**Verplichte feedback + demo opdracht 1 :**

**Start week van 30 april (inschrijven via tolinto)**

**Deadline opdracht 3 (+ eventuele update van opdracht 2) : vrijdag 25 mei**

# Referenties


- https://www.hackerearth.com/practice/algorithms/graphs/graph-representation/tutorial/
- http://www.astrolog.org/labyrnth/algrithm.htm
- https://en.wikipedia.org/wiki/Maze_generation_algorithm