OPGAVE 1: RIVER CROSSING

Zoals besproken in het college: een boer heeft een wolf, een geit en een kool. Hij moet zichzelf met beide dieren en de kool naar de andere oever brengen. Er is een kleine boot waarin hij slechts één ding tegelijk kan meenemen. Echter, als de wolf en de geit alleen gelaten worden, eet de wolf de geit op. Als de geit en de kool alleen gelaten worden, eet de geit de kool op.

Schrijf een python programma dat alle oplossingen geeft. Het kan in minder dan 80 regels. Je hoeft geen GUI te maken, het mag gewoon met een simpele CLI.

Enkele aanwijzingen. Een toestand (state) is een weergave van linker- en rechteroever. Bijvoorbeeld, 'FGCW|' is de begin-toestand en '|FGCW' de eindtoestand. Een toestand kan overgaan in een volgende toestand doordat de boer met een item de rivier oversteekt. Wanneer we de toestanden zien als knooppunten en de overgangen als takken krijgen we een boomstructuur (tree) die we kunnen doorzoeken met DFS. Zie verder ook de sheets van het college.

Vraag: wat is de tijdcomplexiteit van je oplossing?

OPGAVE 2: BOGGLE

Boggle is en woordspel waarbij zoveel mogelijk woorden moeten worden gevonden in een bord van NxN letters. De woorden kunnen worden gemaakt door reeksen te maken van aanliggende letters. In deze opgave is aanliggend: horizontaal of verticaal (dus niet diagonaal), en met de mogelijkheid om door te gaan van laatste naar eerste kolom, en van de laatste naar de eerste rij (en omgekeerd).

In het onderstaande 4x4 bord kunnen bijvoorbeeld de woorden HUIS, MAT, TAS en EI gevonden worden.

P	I	E	T
G	A	A	T
A	T	M	S
H	U	I	S

Schrijf een programma voor een NxN bord, waarbij N kan variëren. Het kan in minder dan 50 regels. Je hoeft geen GUI te maken, het mag gewoon met een CLI en gebaseerd op tekst.

De file words.txt (te vinden op Blackboard) bevat een lijst met woorden die je kan gebruiken om te 'matchen'. Het is handig om hiervan eerst een lijst te maken in het geheugen met alle mogelijk prefixen (kies hiervoor een geschikte datastructuur). Prefixen van HUIS zijn bijvoorbeeld H, HU en HUI. Een 'geldige' toestand is dan een reeks van letters die overeenkomt met een prefix.

Op basis van DFS kun je alle mogelijkheden aflopen. Een pad stopt wanneer de reeks letters niet voorkomt in de lijst met prefixen.

Vraag: wat is de tijdcomplexiteit van je oplossing?

OPGAVE 3: NUMBRIX

Deze opgave lijkt enigszins op de vorige opgave. Numbrix is een puzzel op een NxN bord waarbij het eerste nummer (1) en het laatste nummer (N*N) gegeven zijn. Er moet een pad gevonden worden van 1 naar het laatste nummer, maar dit pad moet wel lopen langs/over de gegeven tussenliggende nummers (‘sleutels’ of ‘clues’ geheten). Hierbij moeten de nummers overeenkomen met het aantal stappen, dus vakje '3' moet bereikt worden in stap 3. Een goede Numbrix puzzel heeft precies één oplossing.

In onderstaand voorbeeld is de opdracht om een route te vinden van 1 naar 81, zodanig dat onderweg de gegeven nummers worden bezocht. De route mag alleen horizontaal of verticaal gaan (dus niet diagonaal; dit heet soms ook een Manhattan path).

7		3		1		59		81
			33	34	57			
9		31				63		79
	29						65	
11	12			39			66	77
	13						67	
15		23				69		75
			43	42	49			
19		21		45		47		73

Zie ook Hidato op wiki en Alex Bellos voor meer achtergrond informatie.

Schrijf een programma in Python dat de oplossing vindt voor een NxN bord, gegeven een lijst van clues (1, N×N, en enkele tussenliggende waarden). Het kan in minder dan 80 regels. Je hoeft geen GUI te maken, het mag gewoon met een CLI.

Test dit programma met een bord van 9x9, met de volgende waarden, het getal 0 representeert een lege cel:

```
s = """
0  0  0  0  0  0  0  0 81
0  0 46 45  0 55 74  0  0
0 38  0  0 43  0  0 78  0
0 35  0  0  0  0  0 71  0
0  0 33  0  0  0 59  0  0
0 17  0  0  0  0  0 67  0
0 18  0  0 11  0  0 64  0
0  0 24 21  0  1  2  0  0
0  0  0  0  0  0  0  0  0 """
```

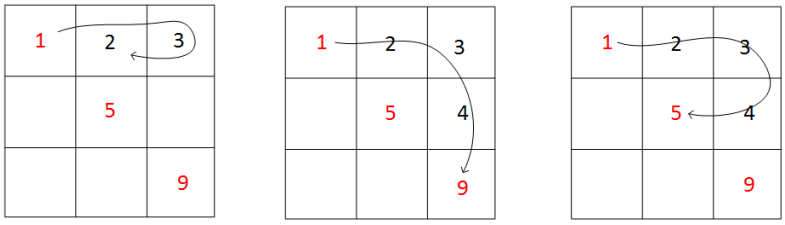
De volgende zaken heb je in elk geval nodig:

- een representatie van het bord (of rooster), hierbij moet ook worden nagedacht hoe je eenvoudig de buren van een cel kan vinden en hoe een lege cel wordt weergegeven;
- een functie die de buren van een cel teruggeeft;
- een functie die het bord (in tekst) afdrukt op het scherm;
- een functie die de oplossing vindt (een pad van 1 naar NxN via de clues).

Strategie
Op basis van DFS kunnen we alle mogelijke paden afgaan. Het is hierbij handig om onderweg de lege cellen te vullen met een ‘stappenteller’. Wanneer we teruggaan (=backtrack), dan moeten we de waarden van deze stappenteller weer aanpassen (maar niet de clues!).

Het lastige hierbij is te bepalen wanneer een pad ongeldig is. Eenzelfde cel twee keer bezoeken is in elk geval ongeldig. Een pad is ook ongeldig wanneer je een cel met een clue bezoekt, en de waarde van de clue is ongelijk aan die van de ‘stappenteller’.

Als voorbeeld, in onderstaande roosters zijn de clue-waarden 1, 5 en 9 gegeven. De eerste 2 paden zijn ongeledig, het laaste pad is wel geldig.



invalid path invalid path valid path

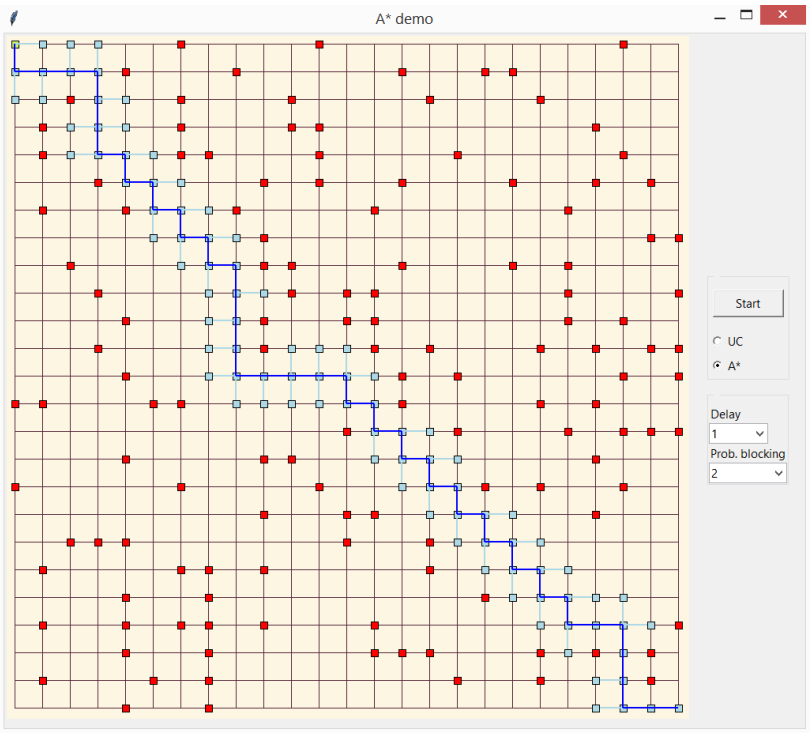
Een mogelijk algoritme gebaseerd op DFS (in pseudo code) is als volgt:

```
1  # using DFS to traverse all paths
2  def solve(position, stepcount, next_clue_in_list):
3
4      can this be a valid path?
5      if yes: update the board and continue
6      if no: return
7
8      if found the last clue nxn
9          # we're done, found a solution
10         display the path
11         return
12     for n in neighbors(position):
13         # try all neigbours, increase the stepcount
14         solve(n, stepcount+1, next_clue_in_list)
15
16 solve(first_position, 1, next_clue_in_list)
```

Vraag: wat is de tijdcomplexiteit van je oplossing?

OPGAVE 4: HET OPTIMALE PAD VINDEN MET UCS EN A*

Er moet een pad gevonden worden op basis van het A* algoritme van links boven (start) naar rechts onder (goal). De rood gemarkeerde punten zijn obstakels, hier kan het pad dus *niet* langs gaan. Het gevonden pad moet blauw worden weergegeven. De punten die onderweg bezocht worden (d.w.z. in de queue komen) moeten lichtblauw worden getoond.



De PriorityQueue klasse uit model.py kan worden gebruikt in het A* algoritme. Je zult ook moeten nadenken over een geschikte heuristiek om de afstand tot de goal node te schatten. Hierbij moet je er op letten dat de deze schatting steeds optimistisch is (d.w.z. 'consistent').

Je moet het verschil kunnen laten zien tussen UCS en A*, zodat de meerwaarde van een heuristiek duidelijk wordt. Verder moeten de delay en de kans op een blocking node kunnen worden aangepast met de combo boxen.

Op Blackboard is de file a_star_start.zip te vinden met 3 Python files. De functie search(app, start, goal) moet nog worden aangepast. Dit kan in ca. 50 regels.

OPGAVE 5: SCHUIFPUZZEL OPLOSSEN MET A*

De schuifpuzzel is een puzzel op een bord van (meestal) 15 vierkante tegels en een leeg veld. In de standaardvariant staan op de 15 tegels de getallen 1 tot en met 15. Deze puzzel was een rage in Amerika en Europa rond 1880. De puzzel bestaat ook in andere varianten, het meest bekend is de 8-puzzel met 3 x 3 tegels.

Steeds kan een tegel naar het lege veld worden geschoven, waarmee de plaats van de tegel en die van het lege veld dus worden verwisseld. De N x N variant is een bekend wiskundig probleem, en wordt soms gebruikt om diverse heuristische functies te onderzoeken.

Deze opgave lijkt op de vorige opgave, en kan ook worden opgelost met A*. De uitdaging zit hem vooral in het vinden van een geschikte (niet te ingewikkelde) heuristische functie om de afstand van een willekeurig bord (start state) tot het doel-bord (goal state) te schatten. Maak je programma zodanig dat het heuristische functie kan worden uitgezet, zodat het verschil in performance tussen UCS en A* getoond kan worden.

Het doel-bord is als volgt:

1	2	3
4	5	6
7	8	0

Hierin representeert ‘0’ het lege veld.

Niet elk start-bord heeft en oplossing; een bekend start-bord is:

8	6	7
2	5	4
3	0	1

Maak het programma zo dat het niet alleen werkt voor bovenstaande 3x3 borden, maar voor elk NxN bord, waarbij N kan variëren. Op Blackboard is de file start-sliding-puzzle.py te vinden die je op weg kan helpen. Het kan in minder dan 80 regels. Je hoeft geen GUI te maken, het mag gewoon met een CLI. Een mogelijk (begindeel van) de output is als volgt:

```
D:\Jacob\Python\games\Sliding puzzle>python eight_sliding_puzzle_astar.py
the start>  867 254 301
the goal>   123 456 780

nr states visited: 31
867
254
301

867
254
310

867
250
314

867
205
314
```