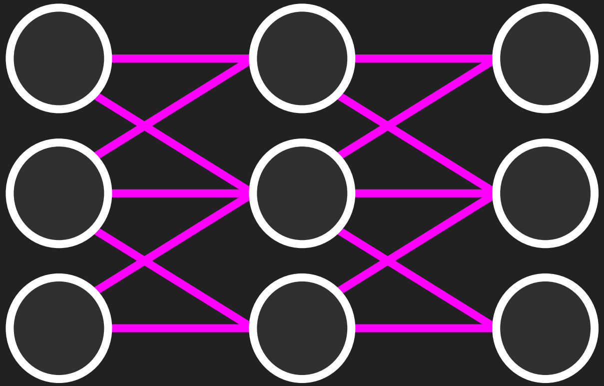
Comparing The Effectiveness Of Machine Learning Against Traditional Optimisation Algorithms For Path-finding.

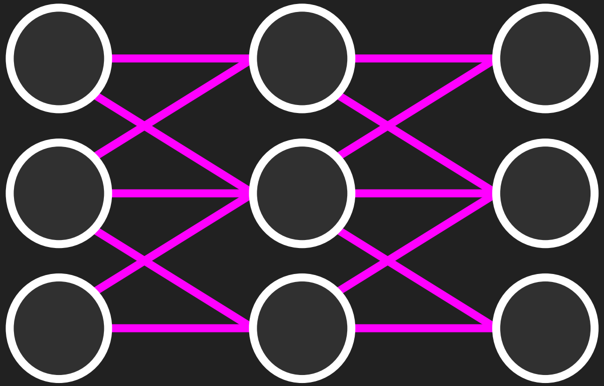
What does that mean?

What does that mean?



$$\mathcal{F}(\mathbf{x}):$$

What does that mean?



Learn

$$\mathscr{F}(\mathbf{x}):$$

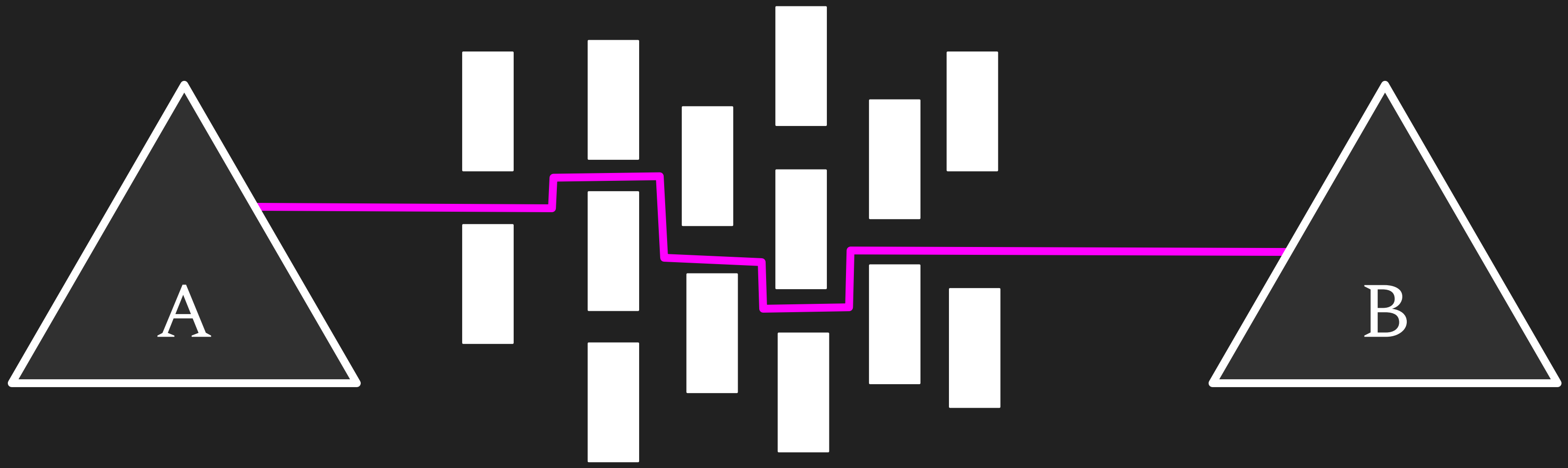
Compute

What does that mean?

Pathfinding - Finding the (Shortest) Path

What?

Pathfinding - Finding the (Shortest) Path



Why?

Why?

- It’s Big News
  - New Speeds
  - More Data

Why?

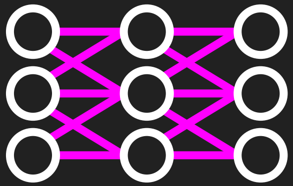
- It's Big News
  - New Speeds
  - More Data
- Real World Uses
  - Self-Driving Cars
  - Facial Recognition

Why?

- It’s Big News
  - New Speeds
  - More Data
- Real World Uses
  - Self-Driving Cars
  - Facial Recognition
- I’m curious
  - ‘Intelligence’
  - How close to Skynet are we?

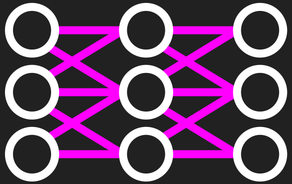
The Plan

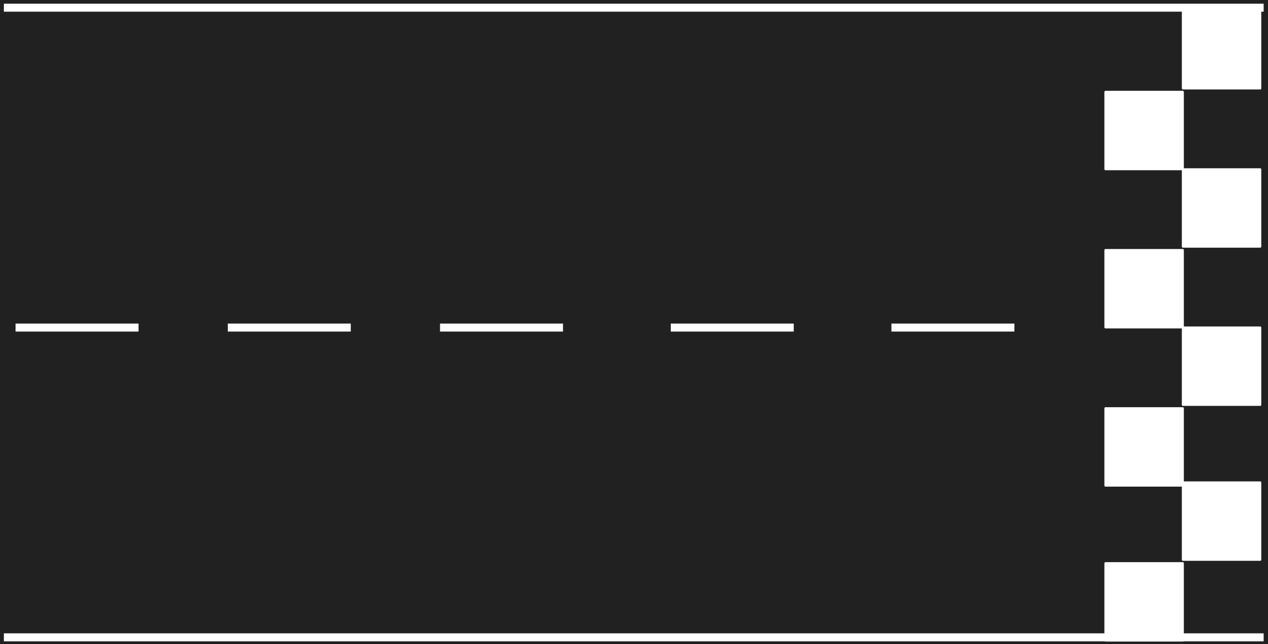
The Plan



$\mathscr{F}(\mathrm{x})$:

The Plan



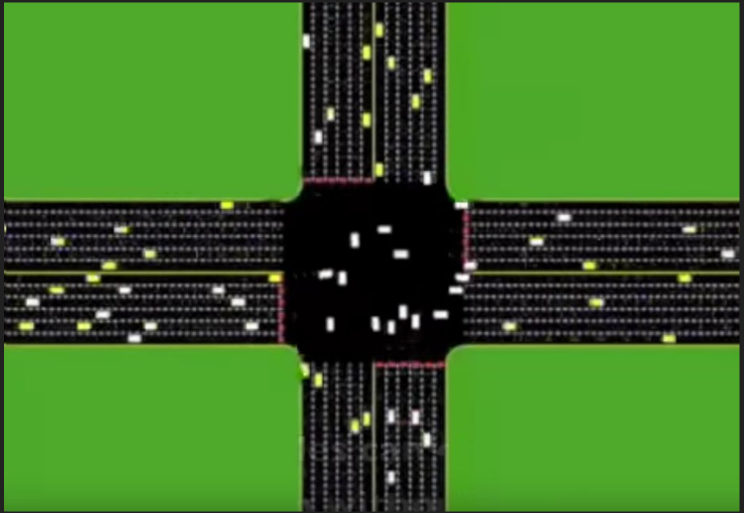


$\mathscr{F}(\mathbf{x})$:

The Plan

The Plan

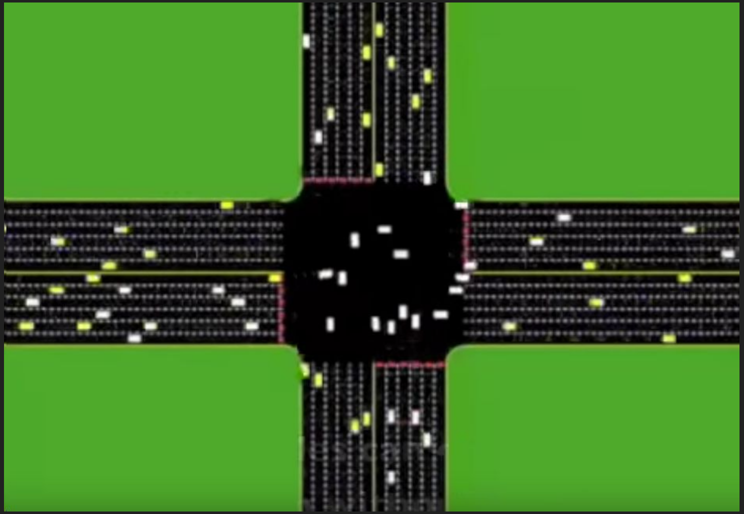
Navigation

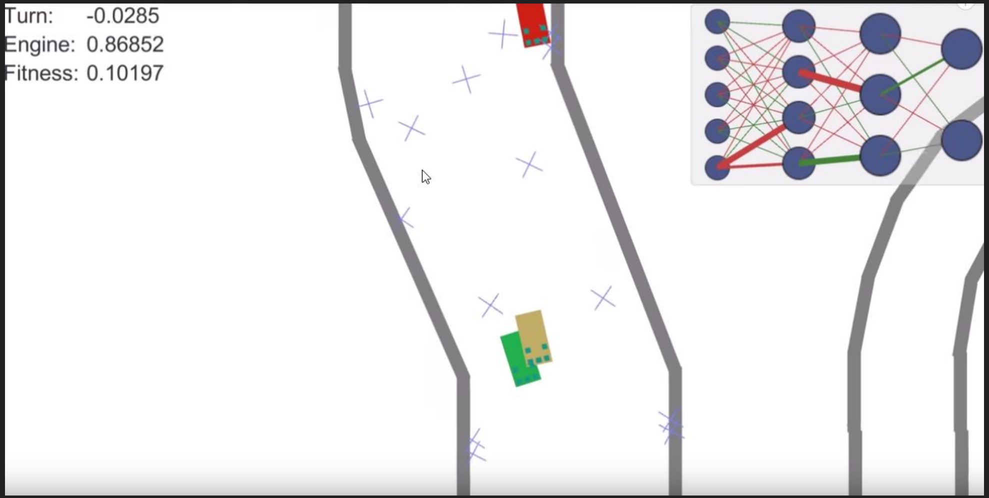


The Plan

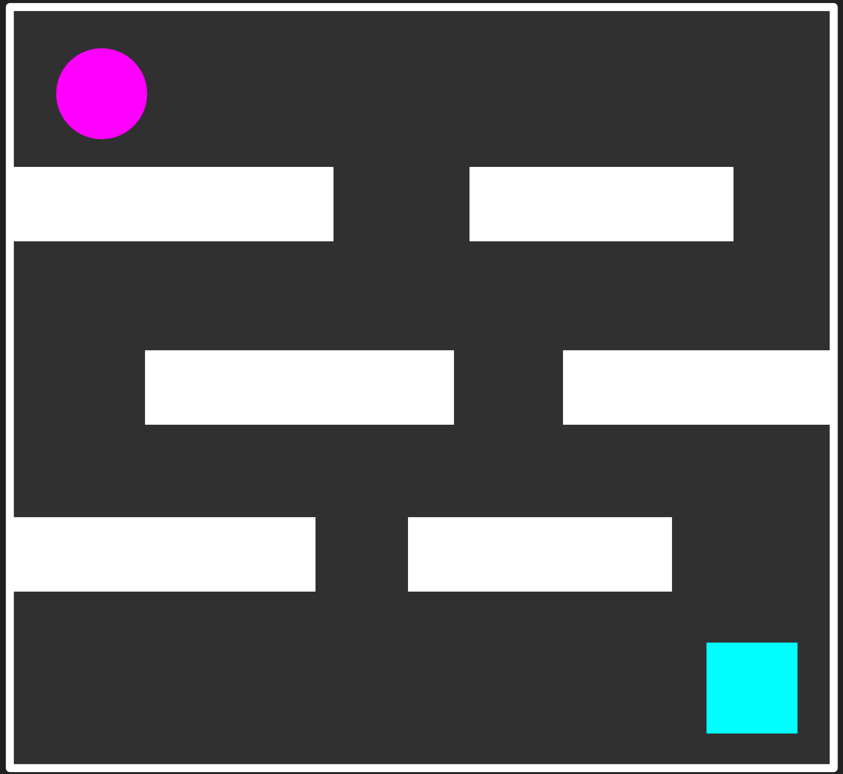
Navigation

Driving Cars





Maze Solving



Algorithm Options

- Broad topic

Algorithm Options

- Broad topic
    - Travelling salesman

Algorithm Options

- Broad topic
  - Travelling salesman
  - Optimal network configurations

Algorithm Options

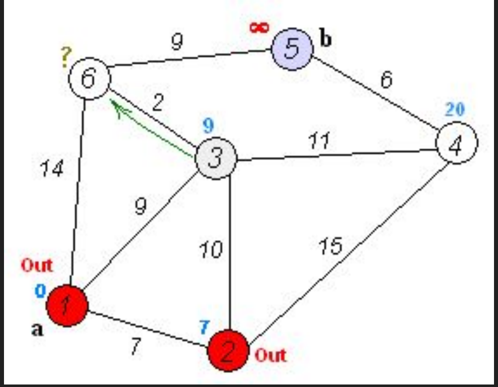
- Broad topic
  - Travelling salesman
  - Optimal network configurations
  - Live Mapping Apps - Google Maps etc.

Algorithm Options

- Broad topic
    - Travelling salesman
    - Optimal network configurations
    - Live Mapping Apps - Google Maps etc.
    - Video game AI
        - Racing games
        - Pac-man style maze games

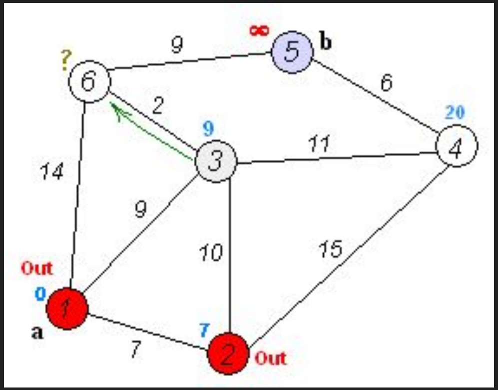
Algorithm Options

- Dijkstra's Algorithm



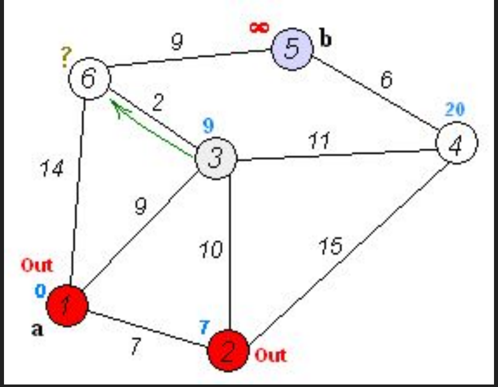
Algorithm Options

- Dijkstra's Algorithm
  - Used to find the shortest path between *a* and *b*.



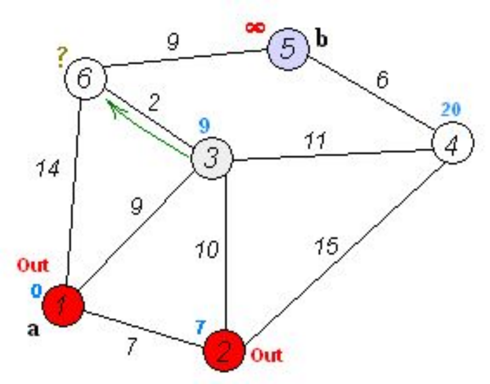
Algorithm Options

- Dijkstra's Algorithm
  - Used to find the shortest path between *a* and *b*.
  - It picks the unvisited vertex with the lowest distance



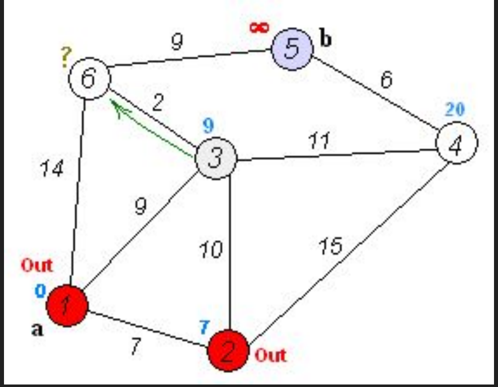
Algorithm Options

- Dijkstra's Algorithm
  - Used to find the shortest path between *a* and *b*.
  - It picks the unvisited vertex with the lowest distance
  - calculates the distance through it to each unvisited neighbor



Algorithm Options

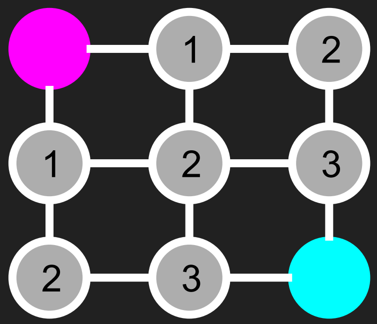
- Dijkstra’s Algorithm
  - Used to find the shortest path between *a* and *b*.
  - It picks the unvisited vertex with the lowest distance
  - calculates the distance through it to each unvisited neighbor
  - Updates the neighbor's distance if smaller.



Dijkstra’s Algorithm

- Ineffective when distance between nodes is uniform

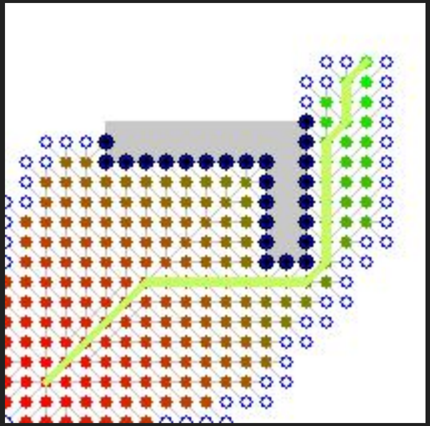
- All nodes will be checked resulting in a large sub-graph



A* Algorithm

- Generalisation of Dijkstra's algorithm
  - Faster as less nodes are considered

- Only if additional information is available that provides a "distance" to the target.



Machine Learning in Brief

- A method of predicting results

X	Y
1	5
2	10
3	?
4	20

Machine Learning in Brief

- A method of predicting results

- Bayes' Theorem: $$P(A \mid B) = \frac{P(B \mid A)\, P(A)}{P(B)}$$

X	Y
1	5
2	10
3	?
4	20

Machine Learning in Brief

- A method of predicting results

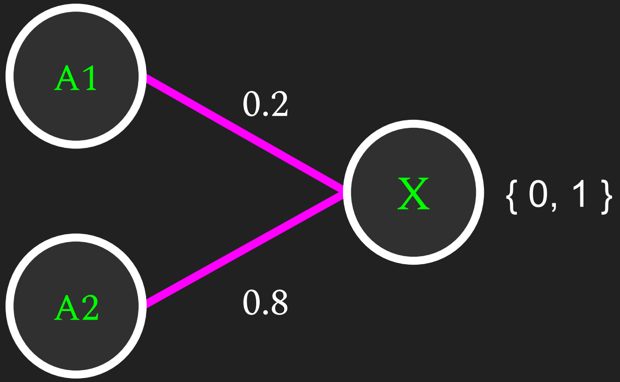
- Bayes' Theorem: $$P(A \mid B) = \frac{P(B \mid A)\,P(A)}{P(B)}$$

- Linear and Non-Linear Regression

X	Y
1	5
2	10
3	?
4	20

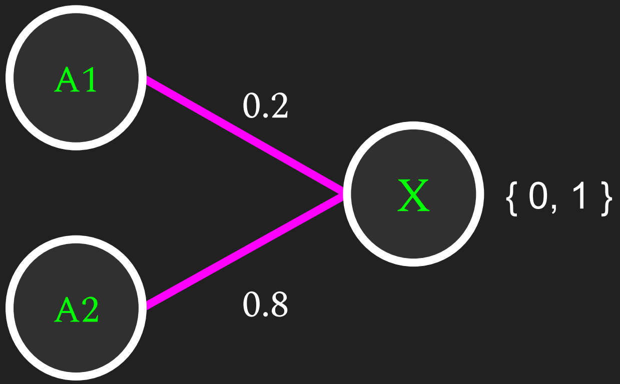
Neural Networks

- Neurons



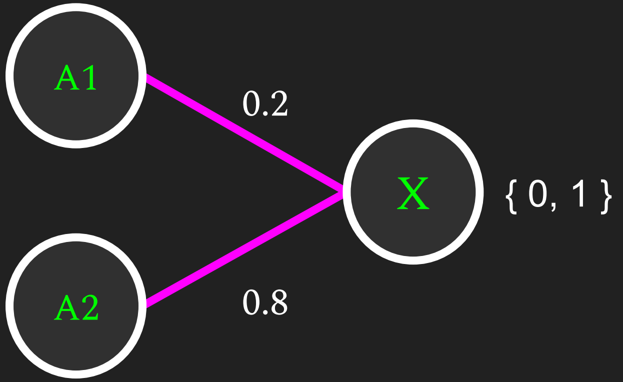
Neural Networks

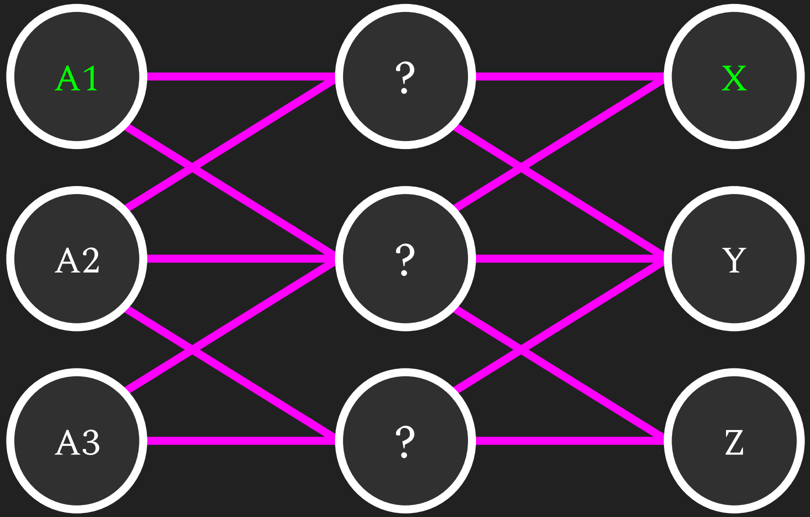
- Neurons
- Connected by weighted bias'



Neural Networks

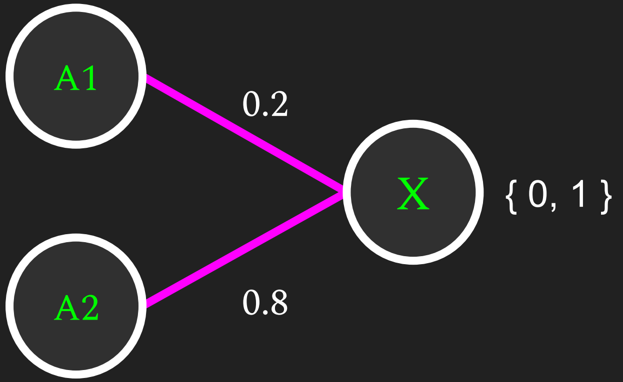
- Neurons
- Connected by weighted bias'
- Build up of Layers

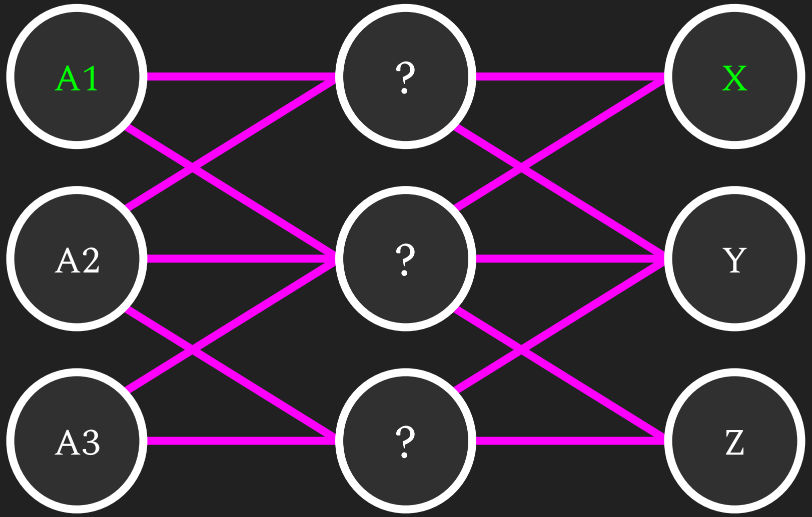




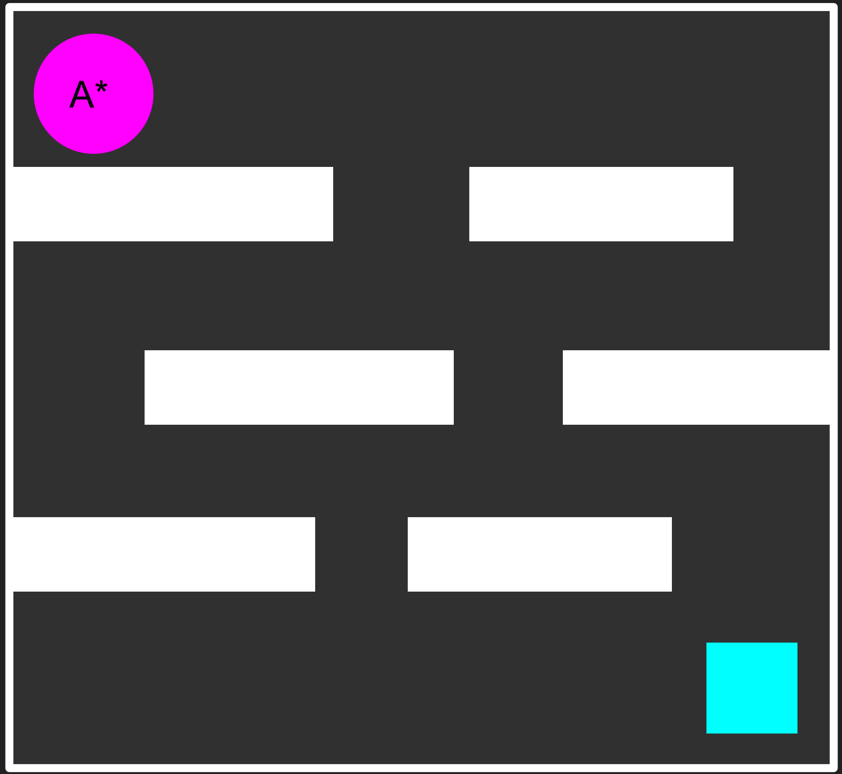
Neural Networks

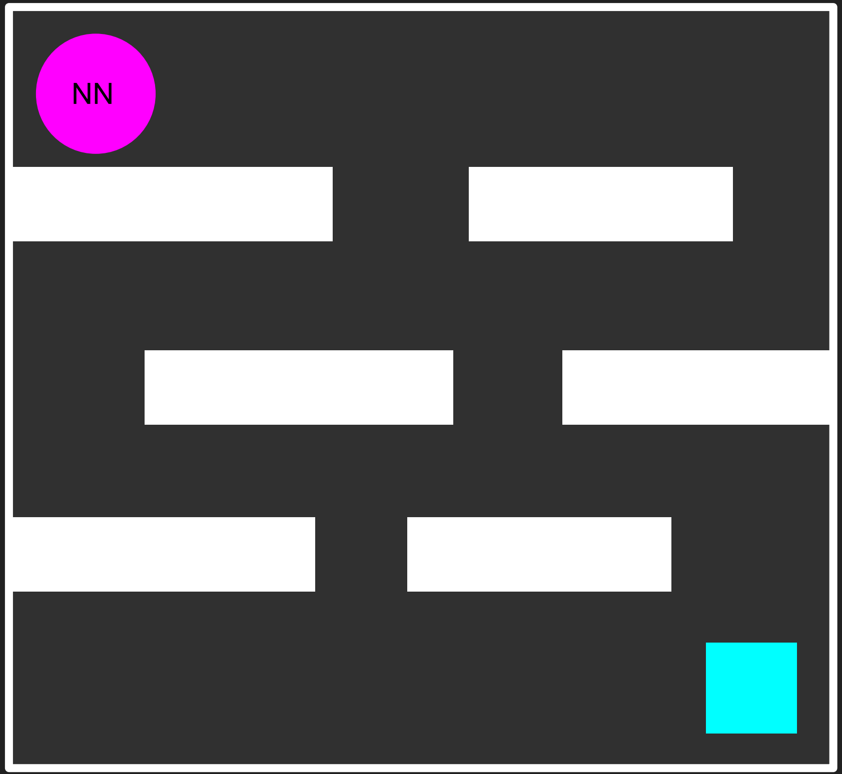
- Neurons
- Connected by weighted bias’
- Build up of Layers
- This bias informs the probability of X occurring if An occurs





Full Scope





The Problem with Static Mazes

- One maze -> One Solution

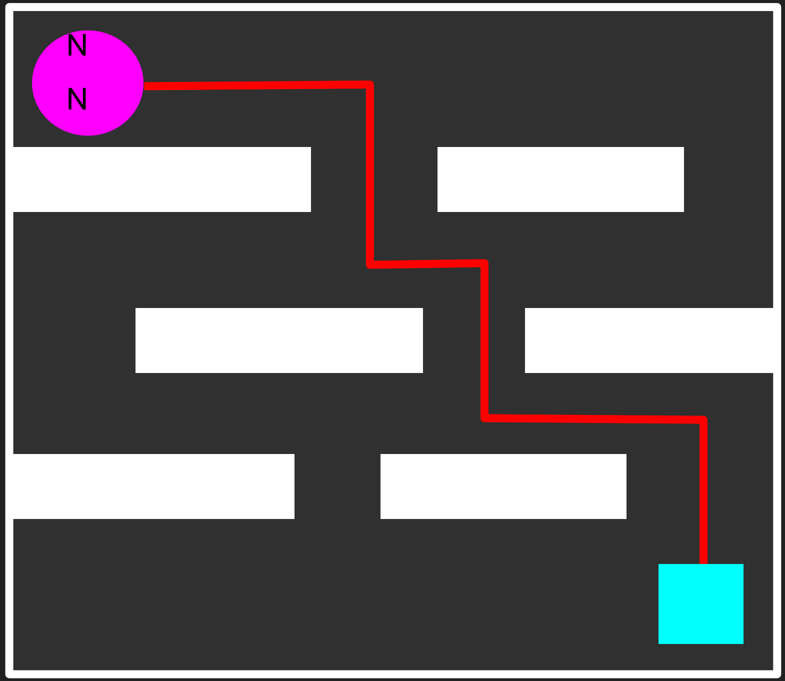
The Problem with Static Mazes

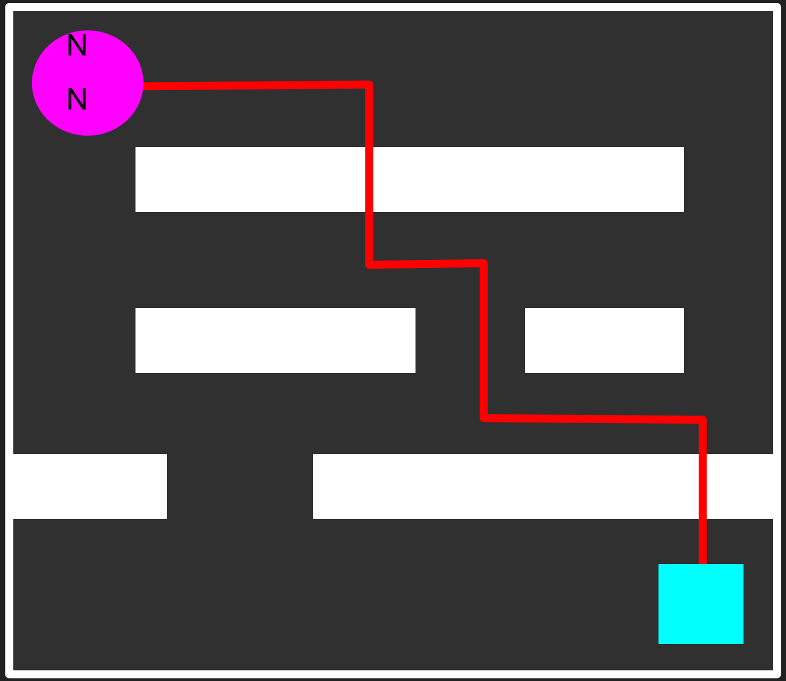
- One maze -> One Solution
- Perfectly fine for functions

The Problem with Static Mazes

- One maze -> One Solution
- Perfectly fine for functions
- Neural Net will only learn a set of instructions

The Problem with Static Mazes





Solution: Random Maze Generator

Implemented Prim's algorithm

Solution: Random Maze Generator

Implemented Prim's algorithm

Modified to work with a grid

Solution: Random Maze Generator

Implemented Prim's algorithm

Start with a grid full of walls.

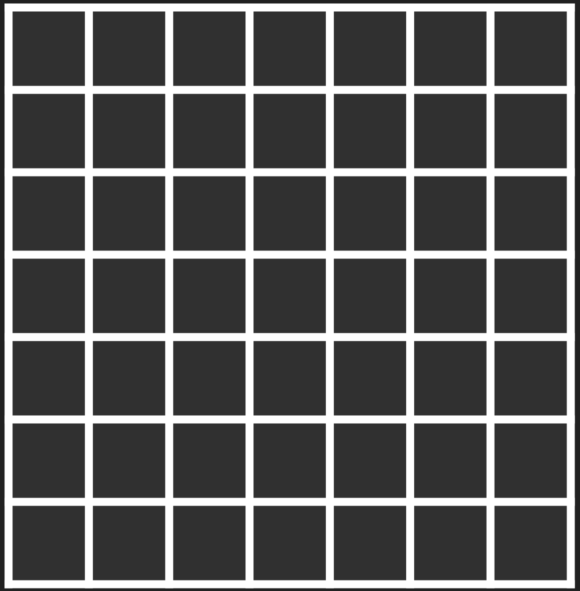
Pick a cell, mark it as part of the maze. Add the walls of the cell to the wall list.

While there are walls in the list:

1. Pick a random wall from the list. If only one of the two cells that the wall divides is visited, then:
   a. Make the wall a passage and mark the unvisited cell as part of the maze.
   b. Add the neighboring walls of the cell to the wall list.
2. Remove the wall from the list.

Solution: Random Maze Generator

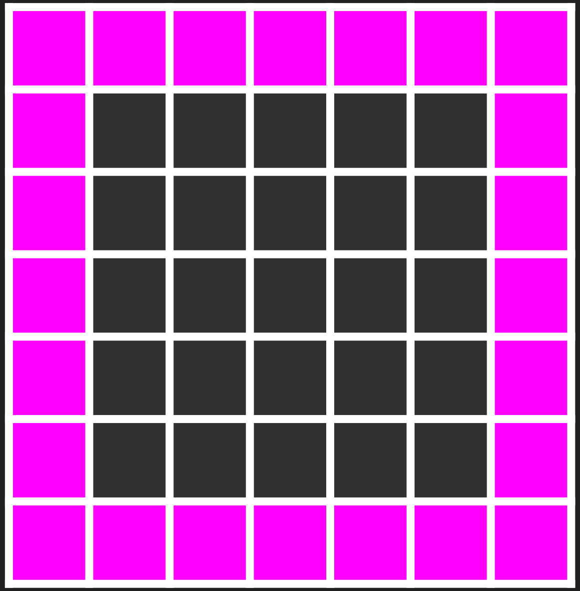
Using Numpy to generate
a Matrix of Booleans



```python
# Build empty grid
Z = numpy.zeros(shape, dtype=bool)
```

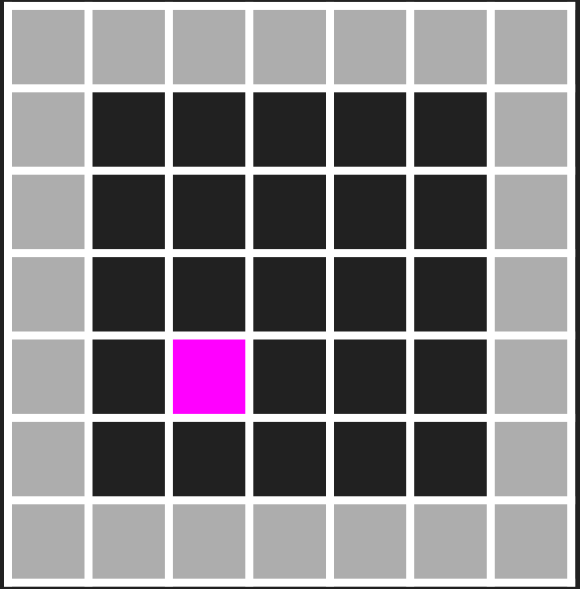
Solution: Random Maze Generator

Next to initialise the edges of my maze



```
# Fill borders
Z[0, :] = Z[-1, :] = 1
Z[:, 0] = Z[:, -1] = 1
```

Solution: Random Maze Generator

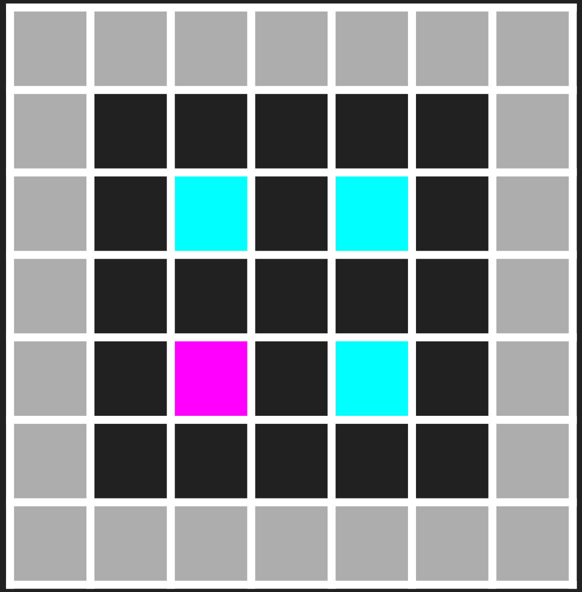


```
for i in range(density):
  # pick a random (even) position
    x = rand.randint(0, shape[1] // 2) * 2
    y = rand.randint(0, shape[0] // 2) * 2

    Z[y, x] = 1
```

Solution: Random Maze Generator

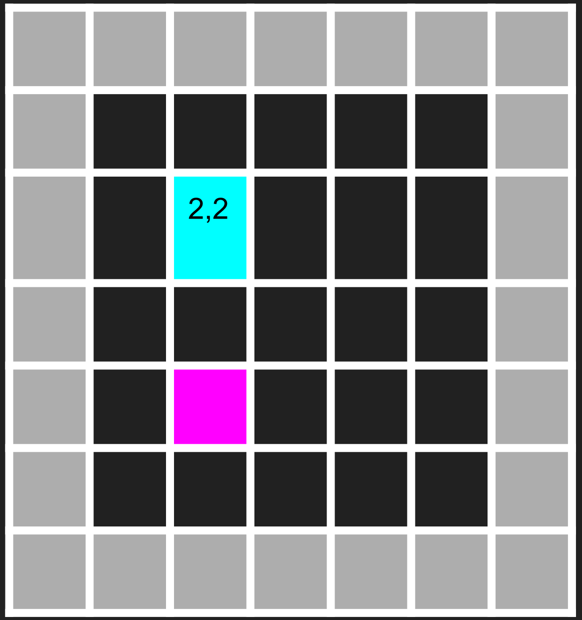
Build a list of neighbouring points
Leaving a gap of 1 between



```
neighbours = []
if x > 1:
    neighbours.append((y, x - 2))
if x < shape[1] - 2:
    neighbours.append((y, x + 2))
if y > 1:
    neighbours.append((y - 2, x))
if y < shape[0] - 2:
    neighbours.append((y + 2, x))
```

Solution: Random Maze Generator

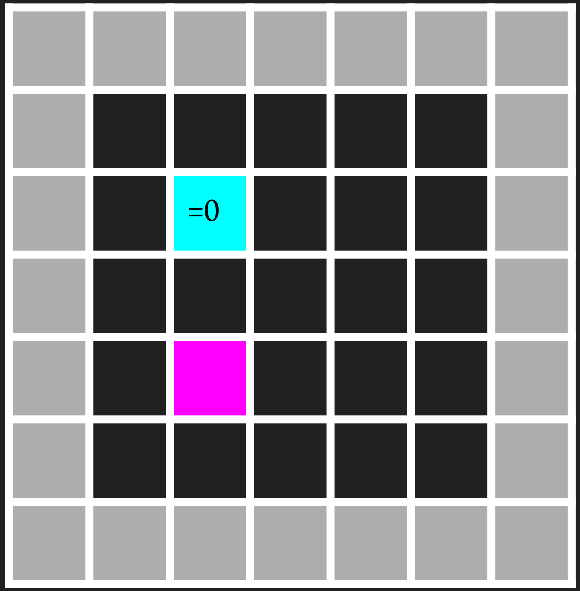
Pick a random neighbour




```python
# If has neighbours, pick at random
if len(neighbours):
    y_, x_ = neighbours[rand(0,
                 len(neighbours) - 1)]
    if Z[y_, x_] == 0:
    Z[y_, x_] = 1
    Z[y_ + (y - y_) // 2,
      x_ + (x - x_) // 2] = 1
    x, y = x_, y_
```

Solution: Random Maze Generator

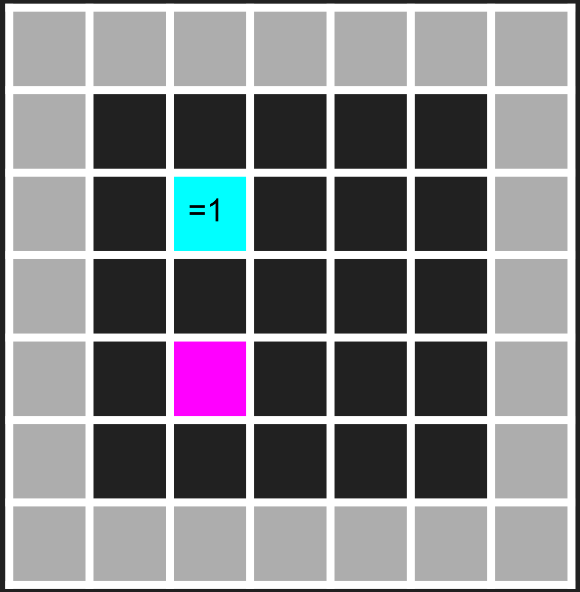
If that neighbour is empty:




```python
if Z[y_, x_] == 0:
    Z[y_, x_] = 1
    Z[y_ + (y - y_) // 2,
       x_ + (x - x_) // 2]= 1
    x, y = x_, y_
```

Solution: Random Maze Generator

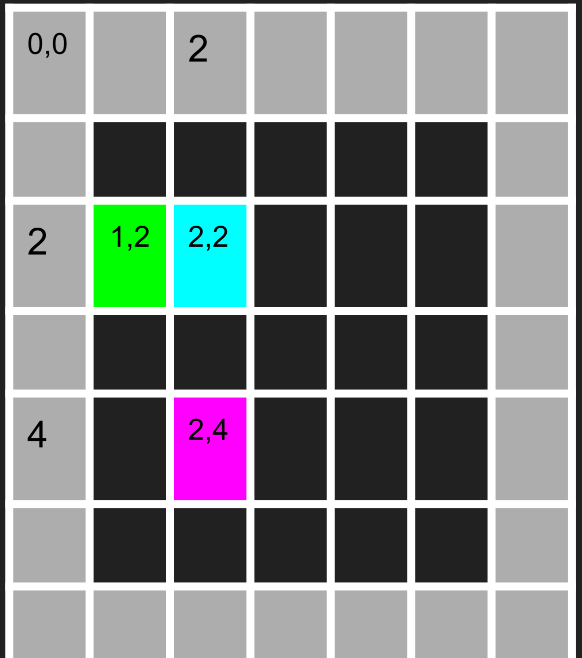
Set it to full




```python
if Z[y_, x_] == 0:
    Z[y_, x_] = 1
    Z[y_ + (y - y_) // 2,
      x_ + (x - x_) // 2]= 1
    x, y = x_, y_
```

Solution: Random Maze Generator

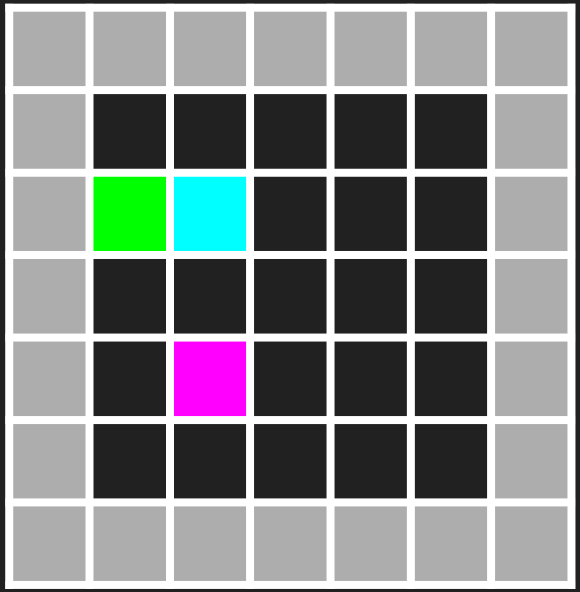
Also set a local neighbour depending on position of the first




```
if Z[y_, x_] == 0:
    Z[y_, x_] = 1
    Z[y_ + (y - y_) // 2,
      x_ + (x - x_) // 2]= 1
    x, y = x_, y_
```

Solution: Random Maze Generator

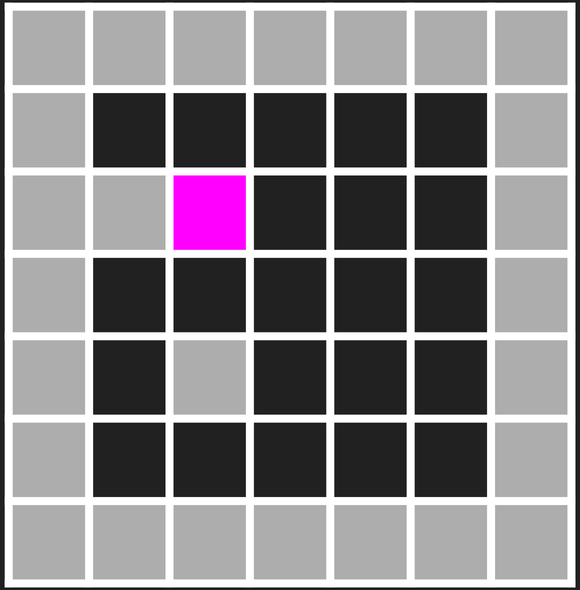
Also set a local neighbour depending on position of the first



```
if Z[y_, x_] == 0:
    Z[y_, x_] = 1
    Z[y_ + (y - y_) // 2,
      x_ + (x - x_) // 2]= 1
    x, y = x_, y_
```

Solution: Random Maze Generator using

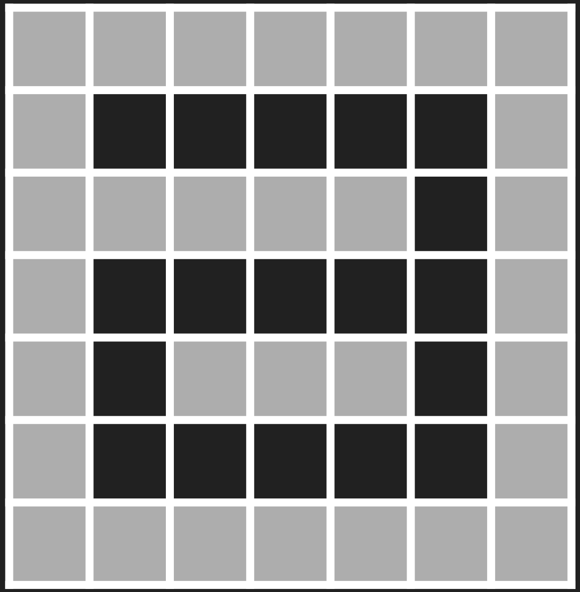
Set neighbour to new start and recursively run until map is complete



```
if Z[y_, x_] == 0:
    Z[y_, x_] = 1
    Z[y_ + (y - y_) // 2,
      x_ + (x - x_) // 2]= 1
    x, y = x_, y_
```

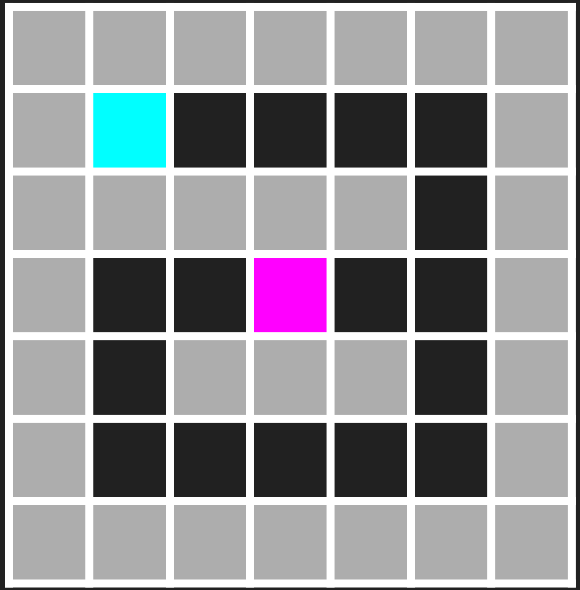
Solution: Random Maze Generator using

Set neighbour to new start and recursively run until map is complete



```
# If has neighbours, pick at random
if len(neighbours):
    y_, x_ = neighbours[rand(0,
                len(neighbours) - 1)]
    if Z[y_, x_] == 0:
    Z[y_, x_] = 1
    Z[y_ + (y - y_) // 2,
      x_ + (x - x_) // 2] = 1
    x, y = x_, y_
```

Solution: Random Maze Generator using



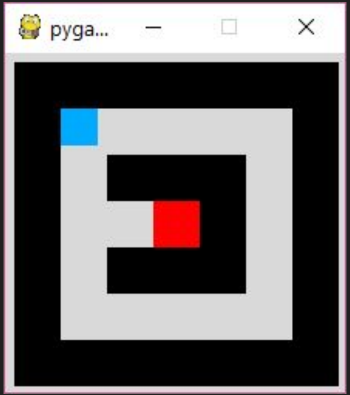
Set a start point (1, 1)

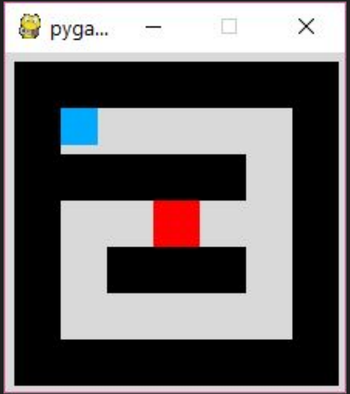
And a goal (3, 3)

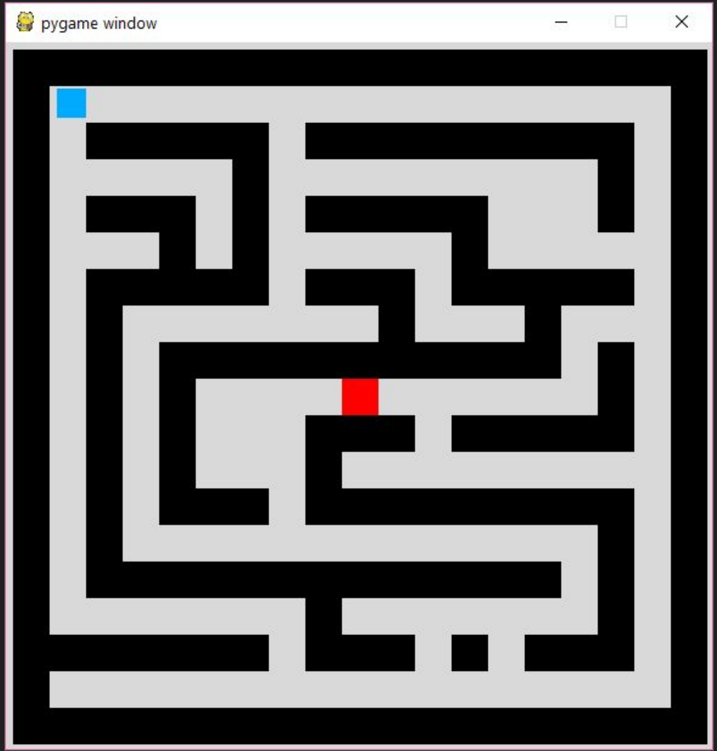
Current Prototype Build

Current Prototype Build

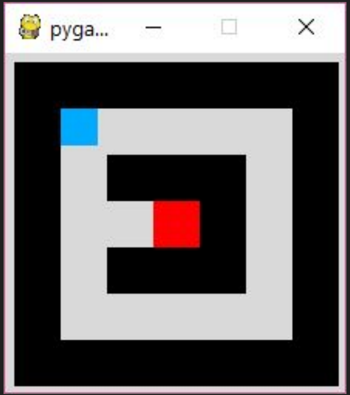
Current Prototype Build

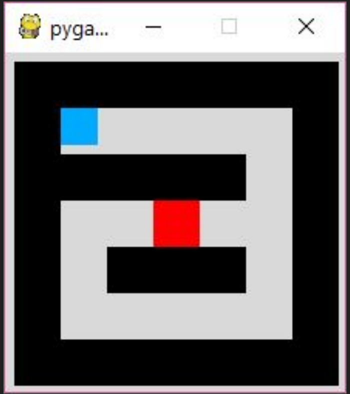


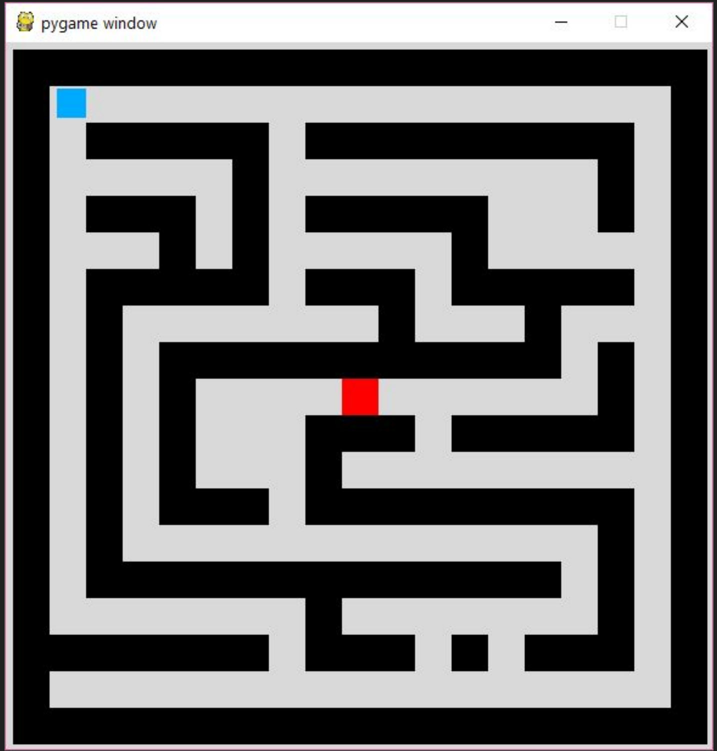


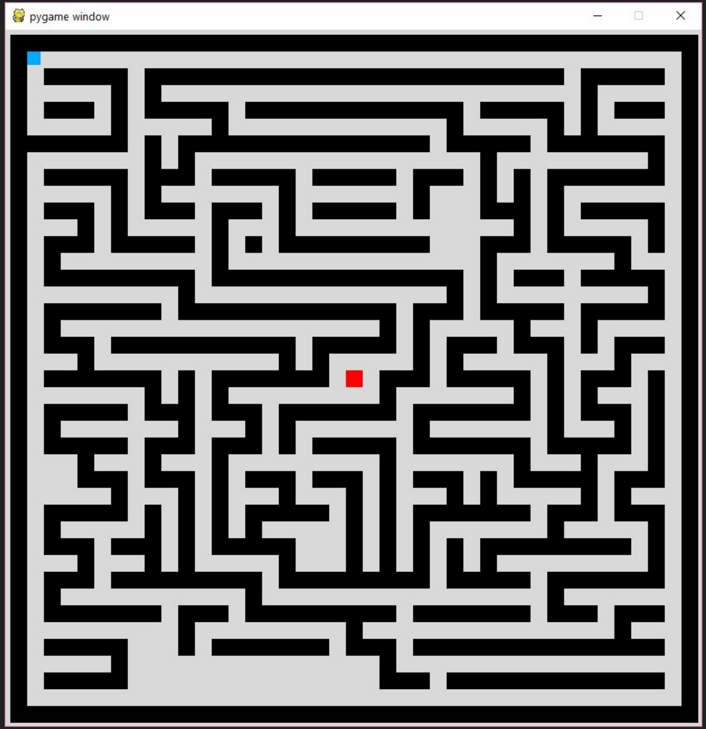


Current Prototype Build









Next Steps

Next Steps

- Implement A* algorithm
- Implement Neural Net
- Game logic
- Analyse Results

What I’ve learnt so far

What I’ve learnt so far

- Time it takes to learn a new package
  - Flexibility of Python
- How many algorithms are out there
  - Don’t reinvent the wheel
  - Standing on the shoulders of those before me
- Time Management is essential

Questions