Artificial Intelligence II

Lesson 1 - Introduction to AI

First Code
Academy

Today’s Plan

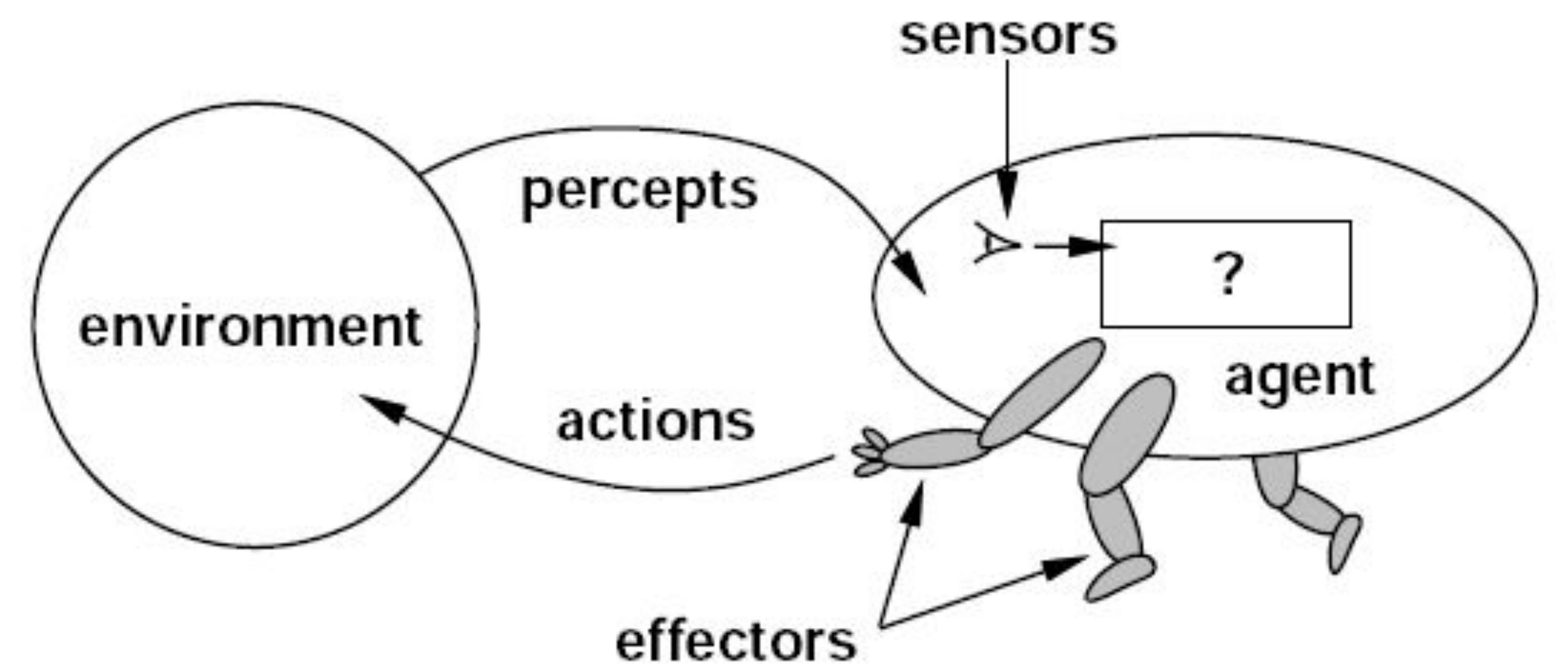
Introduction	00 - 5 min
Review AC260	10 - 20 min
What is AI?	10 - 20 min
Complexity	20 - 30 min
Break	30 - 35 min
Project - Moving Knight on chessboard	35 - 90 min

Introduction

- What is your name?
- How old are you?
- Where do you go to school?
- What do you like the most or find the most interesting about AI?

Review AI Pt. 1

How we think of AI



AI as rational

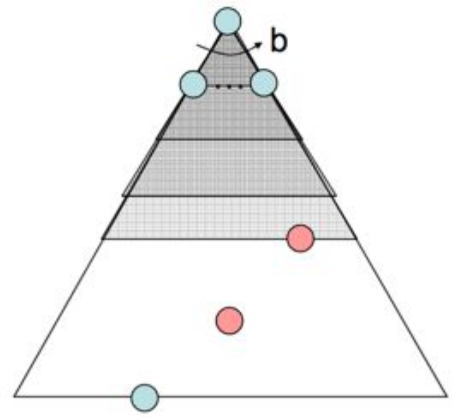
- We want our agents to be **rational** agents
- This means we always try to make the “best” decisions
- How do we search for solutions?

Types of Searches

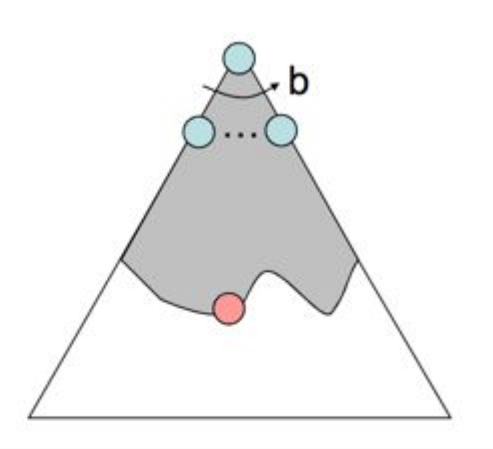
- **Uninformed Searches** don't have any extra information about the environment
- **Informed Searches** take into account how far they have left to go with a heuristic

Identify the Searches

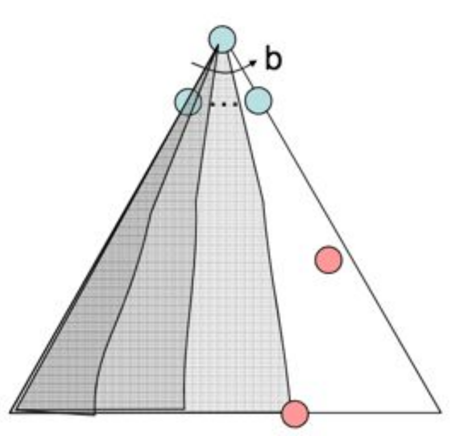
- Can you identify the different searches?



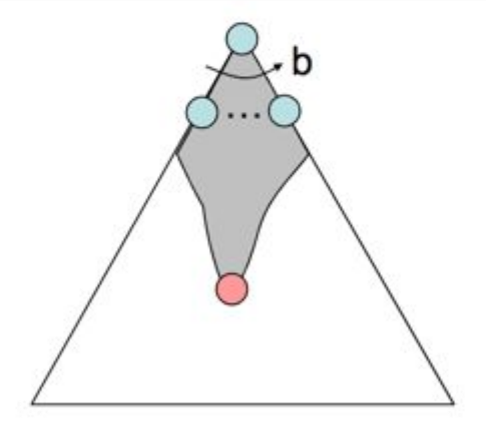
BFS



UCS



DFS



A*

Key Terms

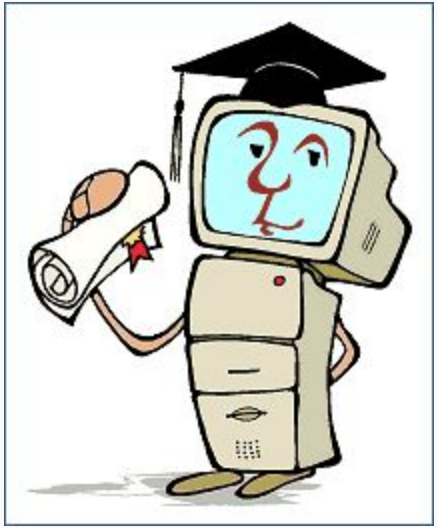
Artificial Intelligence

Complexity

What is AI?

What is AI?

Simple! AI is making the computer smart



Inspiration of AI

- The main goal of AI has always been to imitate human intelligence in a machine
- Begs the question:What is intelligence?
- Known as the problem of intelligence

Types of AI

- **Artificial General Intelligence**: Can do or learn to do anything that a human can
- **Artificial Narrow Intelligence:** Capable of doing only one task well

Types of AI

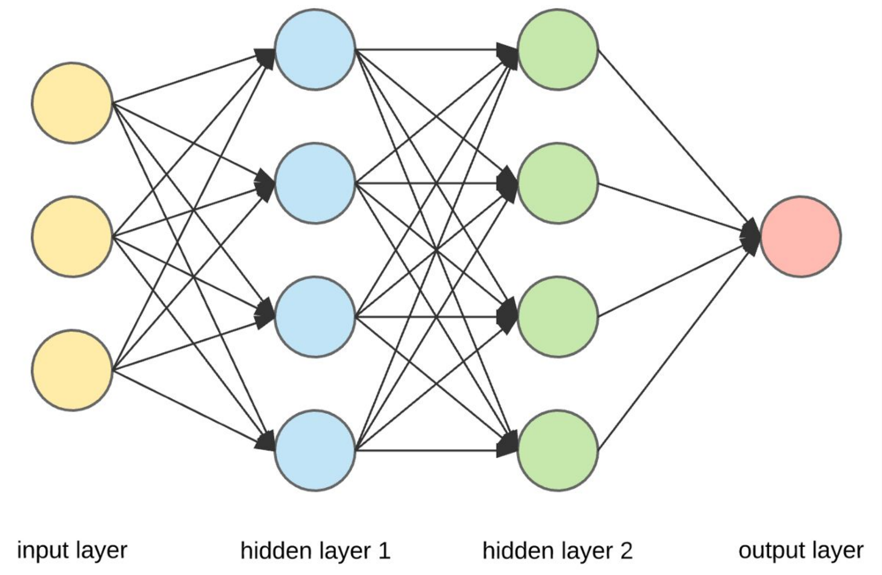
Which ones do you think are general and narrow AI?

History of AI

- 1950’s Turing Test: If an evaluator talks to a machine and a human, can the machine fool the evaluator? If so, the machine passes the test
- 1950s-1970s: AI focused on searching methods and playing games (chess, logic problem solvers, chatbots...)
- 1990s-2000s: With better computers, Machine Learning became the main focus of AI research

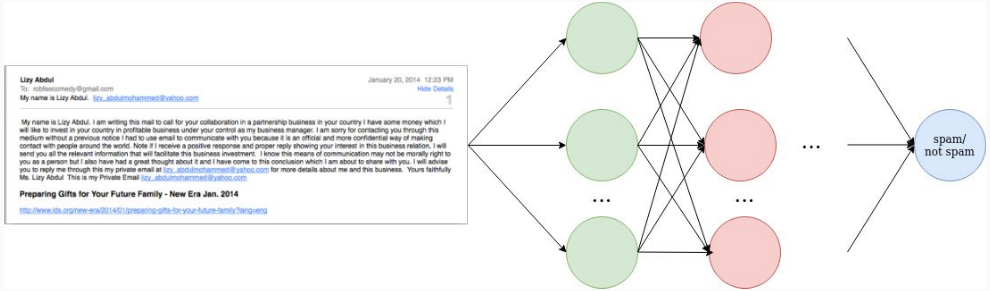
What does AI look like now?

- Mostly focused on different types of Machine Learning
- Able to learn patterns in data the more data we have



Modern AI Examples

Neural Networks classify some information in different categories



Generative Adversarial Networks: Give computer information and have it create completely new samples

Modern AI

- Requires lots of data, which can be hard to find
- Requires lots of computing power, sometimes not easy to run

Complexity

Complexity

- An **algorithm** is a series of steps that accomplishes something
- For example, you could use an algorithm to tie your shoes or to program a video game. Can be almost anything!
- **Complexity** refers to how much resources our algorithm requires to run (time and memory)

Complexity

Example:

How many times do we iterate through Items?

Program 1

```
Items = [1, 2, 3, 4, 5]

for i in items:
    print(i)
```

Program 2

```
Items = [1, 2, 3, 4, 5]

for i in items:
    for j in items:
        print(j)
```

Complexity

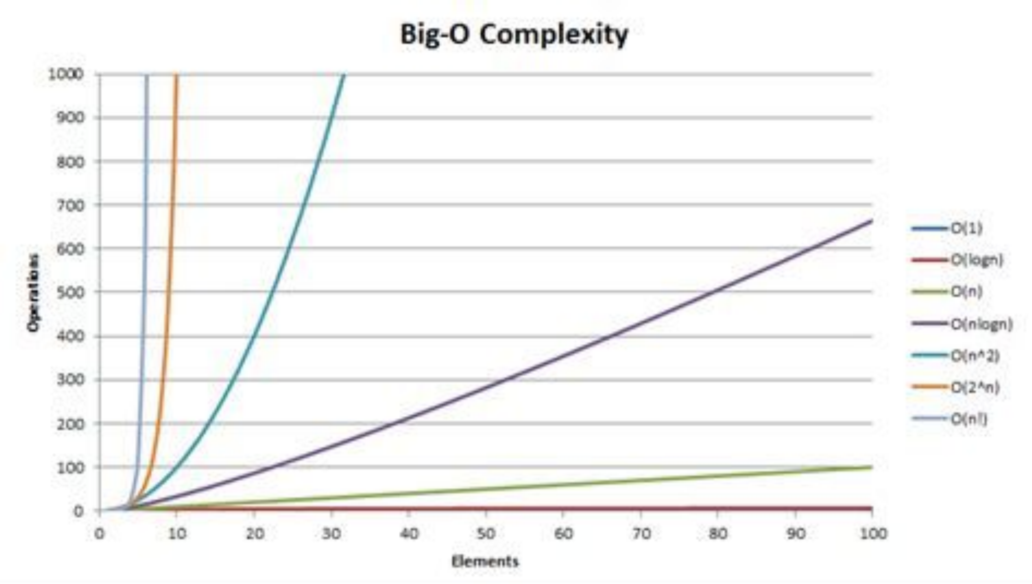
len(Items)	Program 1	Program 2
5	5	25
10	10	100
100	100	10,000
n (any number)	n	n x n = n^2

We would say Program 1 has complexity O(n) and Program 2 O(n^2)

Big-O

If we have input of size *n*, as n gets bigger, how much more time do we need?

- We want a rate that increases the ***slowest***
- Increasing more slowly means a bigger input will run quicker



Complexity Example

Say we wanted to find the sum of all the numbers from 1 to some number n. Which will use less operations for big inputs?

```python
def sum(n):
    count = 0
    for i in range(n):
        count += i
    return count
```

```python
def sum(n):
    return (n * (n - 1)) / 2
```

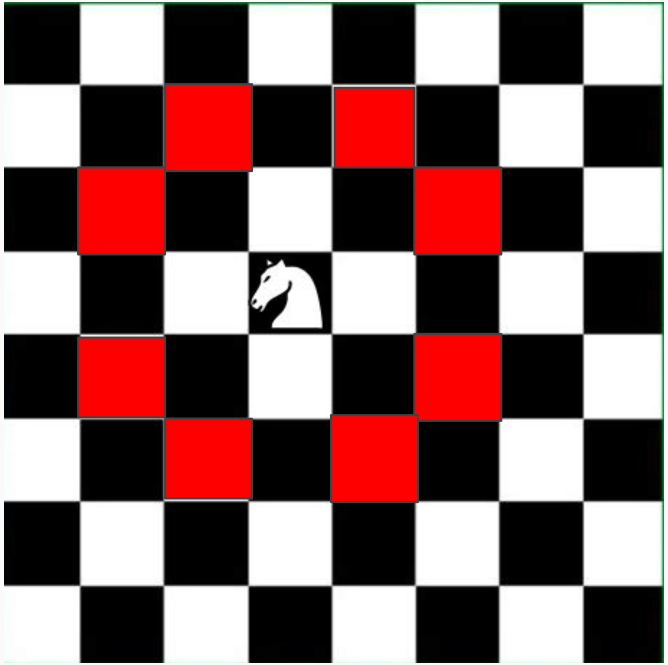
P versus NP

- If a problem’s solution can be quickly checked, is the problem easy to solve?
- Most people think not, but we are not sure since there is no proof
- $1,000,000 waiting for the person that finds an answer!

Project : Moving a knight on a chessboard

Moving a knight

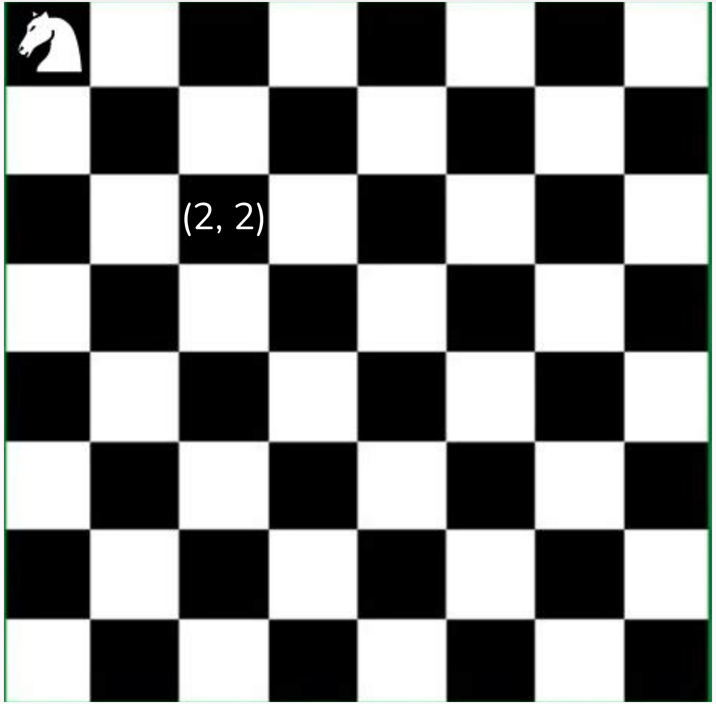
In chess, a knight can move in an "L" shape around the board.



Moving a knight

Given a starting and goal position, what is the best way to get there?

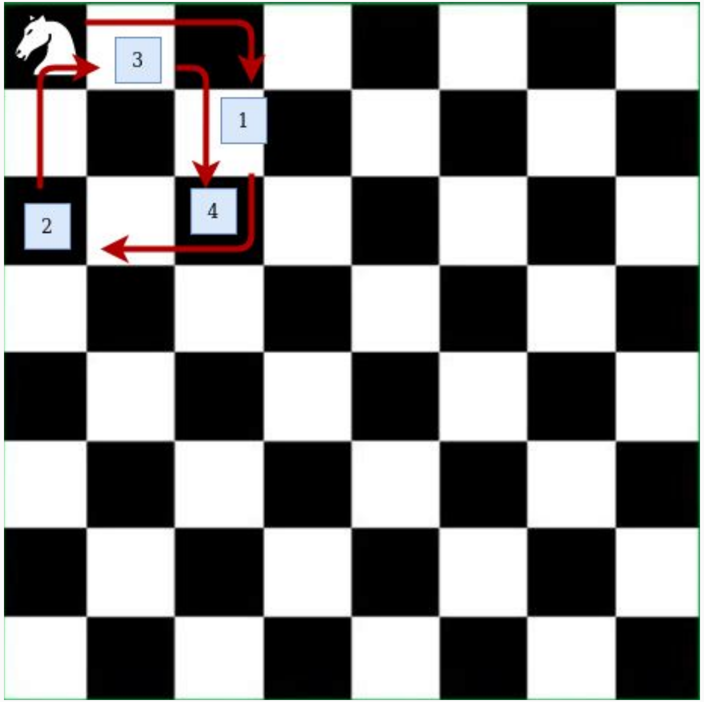
How do we get from (0, 0) to (2, 2)?



Moving a Knight

An optimal solution is:

(0, 0)->(2, 1)->(0, 2)->(1, 0)->(2, 2)



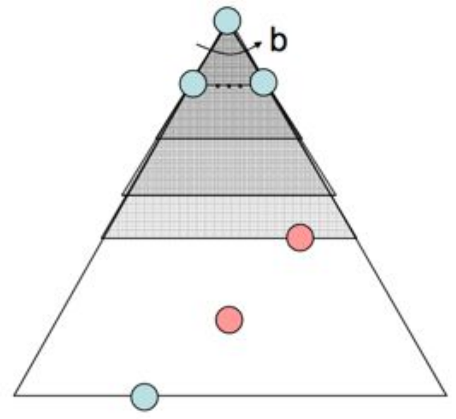
How do we find a solution?

We can search for an optimal solution!

- BFS
- DFS
- A*

Breadth-First Search

BFS - Search all the moves at the current level before moving forward



Main flow of our program

1. Print welcome message
2. Get the starting and ending coordinates
3. Execute the algorithm
4. Print the board with the order of the nodes we explore

Get the starter file

http://bit.ly/FCA_AI2_Starter

Let’s code BFS first

Steps for BFS

1. Create a list to store the nodes we have to explore.
2. Put the starting node into the list.
3. While the list is not empty, pop the oldest node in it.
4. Check if it is the goal node.
5. Add all of the unvisited child nodes to the list.

Let’s code our BFS function!

First, we have to create our queue.

This is where we store the nodes in the current level of the search tree.

```python
def BFS(board, s_x, s_y, e_x, end_y):
    expanded = 0
    # TODO: Make our frontier and store the initial tile
    queue = []
    queue.append(board[s_x][s_y])
```

Let’s code our BFS function!

Next, we loop while the queue is not empty

We also pop the first element in the queue, and mark it as visited

```
    while queue:
        # TODO: Get the oldest node in our queue (closest to the root)
        current = queue.pop(0)

        # TODO: Mark the node as visited
        current.visited = True
        expanded += 1
```

Let's Code our BFS function!

Append the child node to the queue if it has not been visited

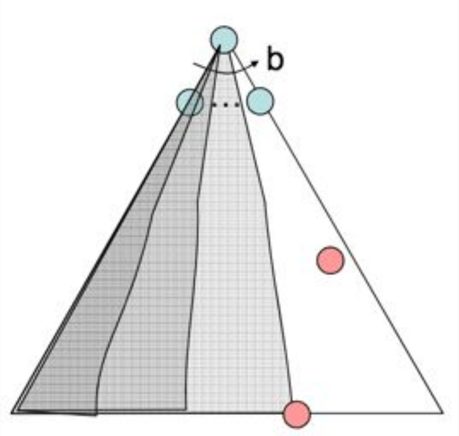
```
# TODO: Set the current node as the child's parent node
child.parent = current

# TODO: append the child to our queue
queue.append(child)
```

That’s it for BFS! Now DFS

Depth-First Search

DFS - Explore a possible solution until we can't go further.



Steps for DFS

1. Create the frontier to store unexplored nodes.
2. Add the starting node to the frontier.
3. While the frontier is not empty, keep popping most recent node.
4. Loop through the node’s children and add all the unvisited nodes to the frontier.

Let's code our DFS function!

Create the frontier and append the initial node.

```
def DFS(board, s_x, s_y, e_x, e_y):
    # Keeping track of the nodes we expanded
    expanded = 0
    # TODO: Create the frontier, where we store the nodes we want to explore
    frontier = []
    # TODO: append the initial node to our stack
    frontier.append(board[s_x][s_y])
```

Let's code our DFS function!

Loop while the frontier is not empty

Pop the most recent node from the frontier and mark it as visited.

```python
frontier.append(board[s_x][s_y])
while frontier:
    # TODO: Pop the node we want to explore
    current = frontier.pop()

    # TODO: Mark our node as visited
    current.visited = True
```

Let’s code our DFS function!

Get the coordinates of the children

```
        # TODO: get the coordinates of the children's moves
        child_x = current.x + move[0]
        child_y = current.y + move[1]
```

Let’s code our DFS function!

If the child node has not been visited:

1. Set its parent to the current node
2. Append it to the frontier

```python
if not child.visited:
    # TODO: Set the child node's parent to the current node
    child.parent = current
    # TODO: Append the child node to our frontier
    frontier.append(child)
```

Run it!

```
Enter name of algorithm: bfs
Enter the starting coordinates: 0 0
Enter the ending coordinates: 3 4
An optimal path is:
(0, 0)->(2, 1)->(4, 2)->(3, 4)->
Expanded 43 nodes using bfs.
1  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
---|----|----|----|----|----|----|----|----|
0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
---|----|----|----|----|----|----|----|----|
0  | 2  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
---|----|----|----|----|----|----|----|----|
0  | 0  | 0  | 0  | 4  | 0  | 0  | 0  | 0  |
---|----|----|----|----|----|----|----|----|
0  | 0  | 3  | 0  | 0  | 0  | 0  | 0  | 0  |
---|----|----|----|----|----|----|----|----|
0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
---|----|----|----|----|----|----|----|----|
0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
---|----|----|----|----|----|----|----|----|
0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
---|----|----|----|----|----|----|----|----|
0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
---|----|----|----|----|----|----|----|----|
```

Challenges

Can you apply any other search algorithm?

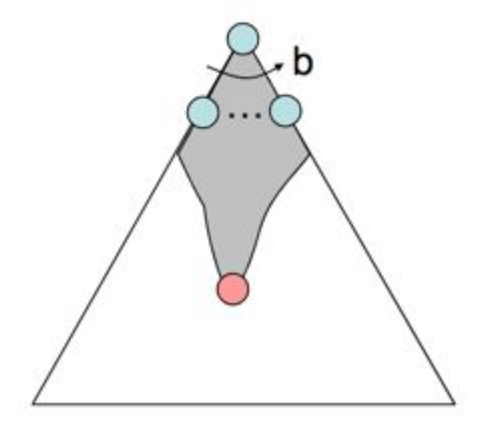
What about an algorithm that uses a heuristic?

A **heuristic** is an “educated guess” as to how much remains to the end.

:D

A*

A* - Every time, use only the node that has the best heuristic value.



Steps for Heuristic Search

1. Define some function to decide which move is better (heuristic)
2. Check the heuristic of the possible moves.
3. Add the node to the frontier.

Solution

Create a heuristic function, we use “Manhattan Distance”

```python
def heuristic(x1, y1, x2, y2):
    # Sum of rows and cols from one point to next
    return (abs(x1 - x2) + abs(y1 - y2))

# Choose the next unvisited node with the lowest heuristic
def heuristic_search(board, s_x, s_y, e_x, e_y):
```

Solution

Loop through all the child nodes.

```
            print_path(current)
            return expanded

        # TODO: Check the possible moves
        for move in movements:
            cx = current.x + move[0]
            cy = current.y + move[1]

            if not is_valid(cx, cy):
                continue

            child = board[cx][cy]
```

Solution

Only add the nodes with decreasing heuristic.

```python
            # TODO: Only explore nodes with a shorter distance
            child.h = heuristic(child.x, child.y, e_x, e_y)
            if child.h < current.h:
                print(f'Adding {child.x},{child.y}')
                child.parent = current
                frontier.put((child.h, child))
                expanded += 1
    return expanded
```

Run

However, it only works for some output....

```
Enter name of algorithm: heur
Enter the starting coordinates: 0 0
Enter the ending coordinates: 7 7
An optimal path is:
(0, 0)->(2, 1)->(4, 2)->(6, 3)->(8, 4)->(6, 5)->(7, 7)
Expanded 20 nodes using heur.
1  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
---|----|----|----|----|----|----|----|----|
0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
---|----|----|----|----|----|----|----|----|
0  | 2  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
---|----|----|----|----|----|----|----|----|
0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
---|----|----|----|----|----|----|----|----|
0  | 0  | 3  | 0  | 0  | 0  | 0  | 0  | 0  |
---|----|----|----|----|----|----|----|----|
0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
---|----|----|----|----|----|----|----|----|
0  | 0  | 0  | 4  | 0  | 6  | 0  | 0  | 0  |
---|----|----|----|----|----|----|----|----|
0  | 0  | 0  | 0  | 0  | 0  | 0  | 7  | 0  |
---|----|----|----|----|----|----|----|----|
0  | 0  | 0  | 0  | 5  | 0  | 0  | 0  | 0  |
---|----|----|----|----|----|----|----|----|
```

Run

....for others, it gets one tile away and can't find a tile closer with better heuristic!

So....not all algorithms work all the time.

```
Enter name of algorithm: heur
Enter the starting coordinates: 0 0
Enter the ending coordinates: 2 2

Expanded 2 nodes using heur.
0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
---|----|----|----|----|----|----|----|----|
0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
---|----|----|----|----|----|----|----|----|
0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
---|----|----|----|----|----|----|----|----|
0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
---|----|----|----|----|----|----|----|----|
0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
---|----|----|----|----|----|----|----|----|
0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
---|----|----|----|----|----|----|----|----|
0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
---|----|----|----|----|----|----|----|----|
0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
---|----|----|----|----|----|----|----|----|
0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
---|----|----|----|----|----|----|----|----|
```

Quiz: http://bit.ly/FCA_Quiz_AI

That’s it for today!

Artificial Intelligence II

Lesson 1 - Introduction to AI

First Code
Academy