



# How would an AI play Track the Crocotta?

Jake Tucker

# Table of contents

**01**

Programming AI



**02**

AI and Chess



**03**

Applying to TTC

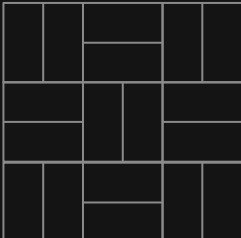




# 01

# Programming for AI

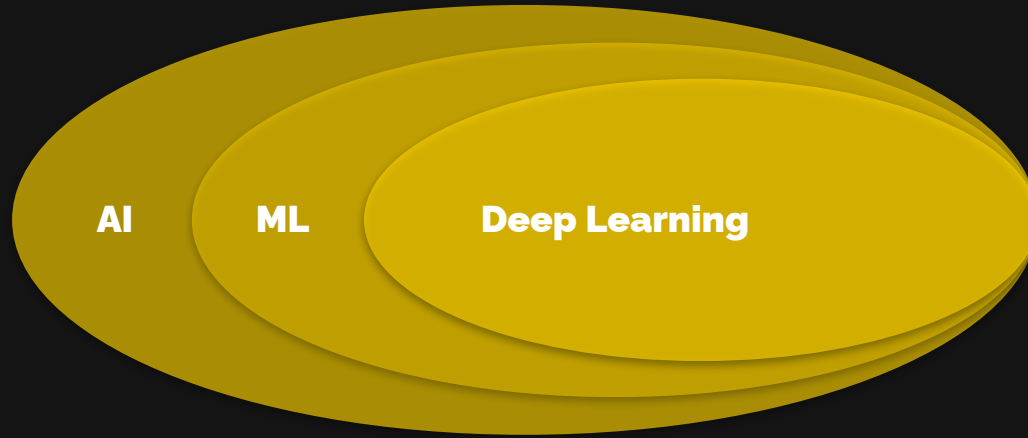
The methods in which a  
robot is programmed to  
make decisions





# What is AI?

The field and study of artificial intelligence in computing has been around since the 1950's, with the creation of the Alan Turing test, and the first computerised game of "checkers".



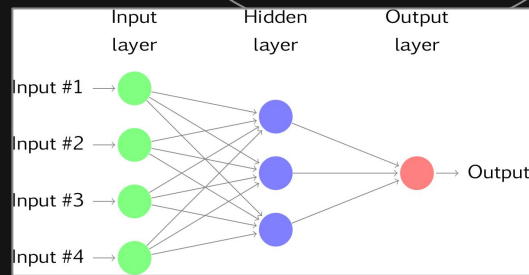
AI is broader than ML and DL. Many AI applications don't need ML in order to be effective.



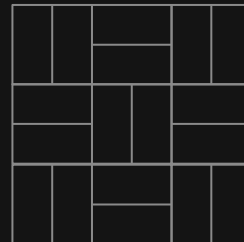
# Defining **Deep** & **Machine** Learning

Do you know what helps you make your point crystal clear?  
Lists like this one:

- Supervised and unsupervised learning in ML
- Set of desired outputs and inputs to develop rules/strategies
- Use of neural networks in deep learning that take an enormous amount of data & time to be trained



Neural network visual



# Pros and Cons of Deep Learning

## Pros

- Creates solutions to advanced problems that would be too hard to code conventionally
- Incredibly scalable
- No need to label/categorize data

## Cons

- Massive data requirement
- Requires lots of time and processing power to train
- Often leaves programmers often without an idea of why something is going wrong



02

# AI and Chess

Implementations of  
computerised chess







**Deep Blue vs. GM Garry Kasparov**

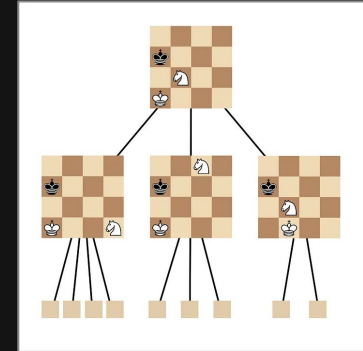
# The **big** Chess ideas

## Piece Weights

Each piece has their own weight. A knight may have a weight of 100 while a pawn only has a weight of 10.

## Minimax Search

Goes through all of the possible moves based on where the computer's pieces are. Time complexity typically improved by "alpha-beta" pruning.





# 03

# Applying to TTC

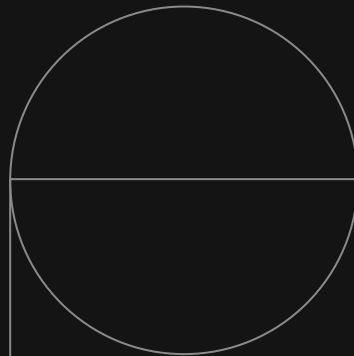
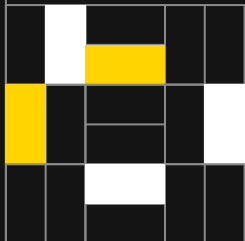
Creating an implementation  
of an AI that plays “Track the  
Crocotta”

# Using a **conventional** coding approach

Since there is not nearly as many possible situations in this game as there is in chess, there is no need to train a model to play the game.

## Game Situations

- An adjacent space with a pit
- An adjacent space with the crocotta
- An adjacent space with nothing

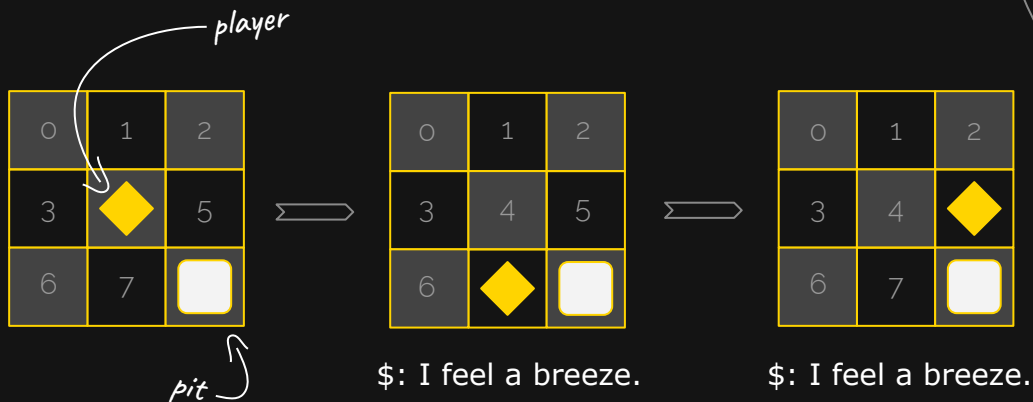


# Making use of chess ideas

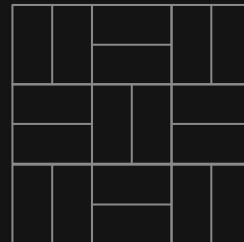
Keep two arrays that contains a set of regions on the 5x5 grid that *possibly* contains the crocotta and *possibly* contains a pit respectively. The weight (or probability in this case) that a position contains a pit changes as the AI searches adjacent spaces.

When prompted with either a pit or crocotta being adjacent, move to the previous space, and check all of the surrounding spaces around that space and add to the probability in each space for the respective warning. Every position that is visited by the player is marked with a 0% probability of it being a pit or containing the crocotta. If a space is presented with no warnings, all adjacent spaces are safe.

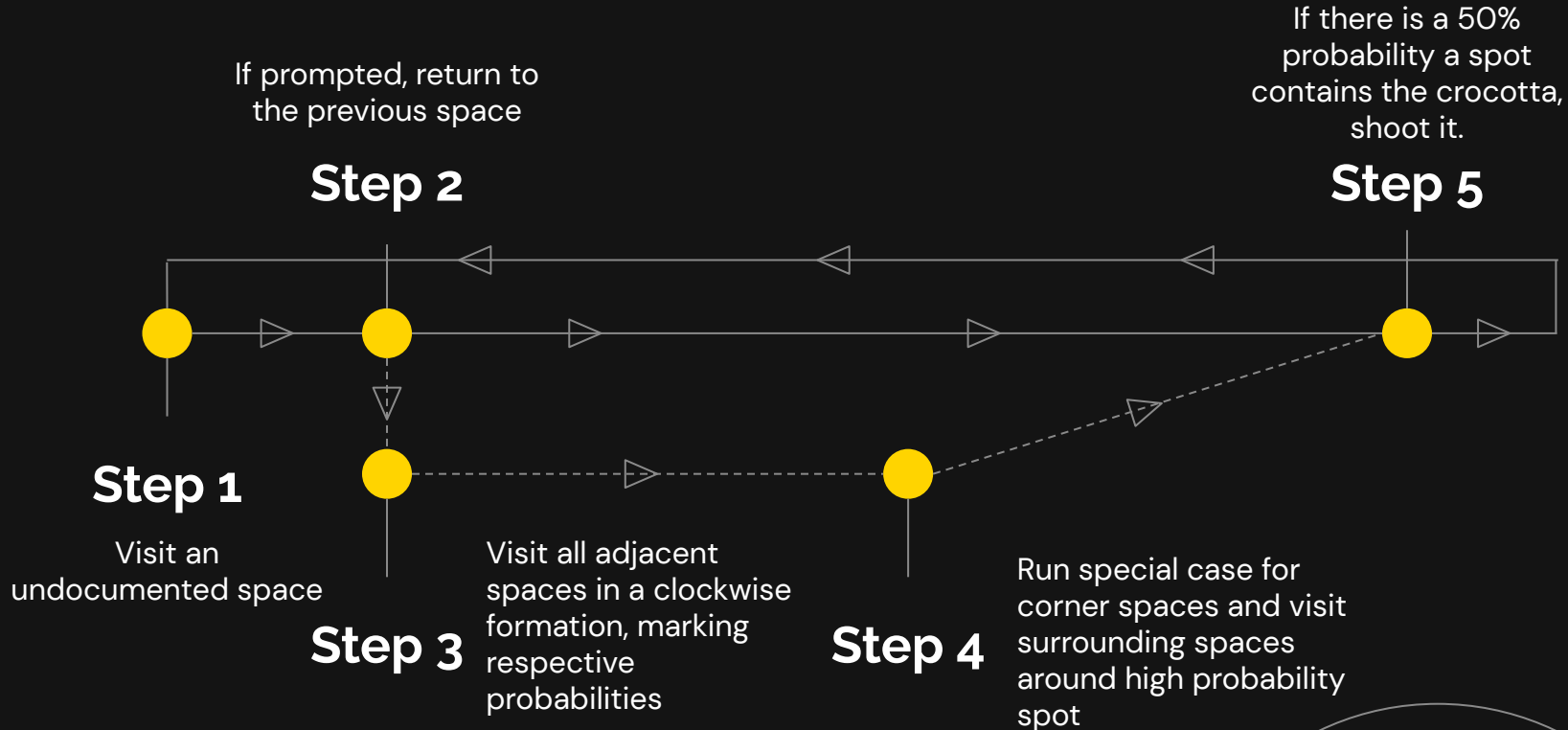
# A quick visual:



Pos 8 pit probability: 25%      Pos 8 pit probability: 50%



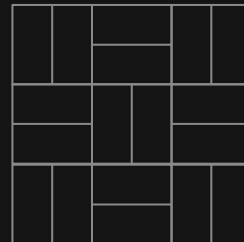
# Algorithm Loop



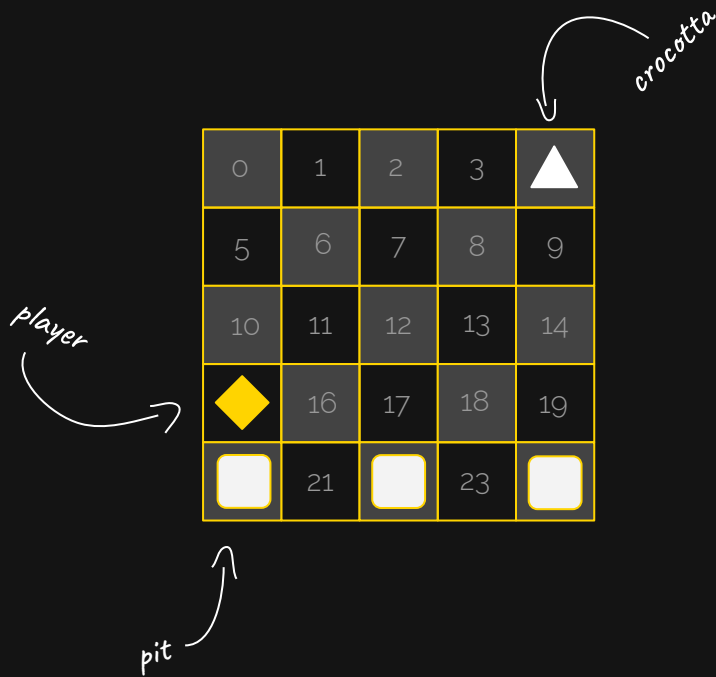
# Special cases: corners and edges

The AI can't visit all four surround spots for a point of interest. Therefore, 100% probability cannot be solved, with a similar situation arising on the edges.






- Need to check surrounding edges
- Using the list of visited locations, we can add the remaining probabilities to edge locations







We are warned for each pit we are next to. In this case, if we move to spot 21, we will be warned twice for a pit.

0	1	2	3	4
5		7	8	9
10	11			14
15		17	18	19
20	21	22	23	

Say we visit space 7. We return to space 6, and visit all adjacent spaces around. Knowing that 1,5,7,11 are open, we can move to space 11, where we are warned about the pit and the crocotta. We were not warned when visiting space 5, so we can conclude that space 10 is safe

# Algorithm Pseudocode

START

While (true):

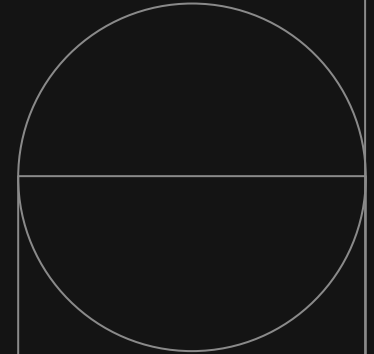
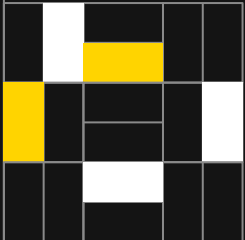
Move in clockwise fashion (move right when possible, then down... prioritize moving to unvisited spots)

If prompted with a warning:

- If warning is for crocotta, and spot probability is  $\geq 75\%$ :  
shoot
- Return to previous spot
- Check surrounding unvisited spots in CW order
- Repeat if warned again
- If all spots present a warning, move to the spot with the lowest probability (implementation in next slide)

break

END



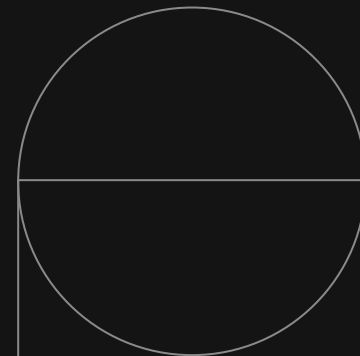
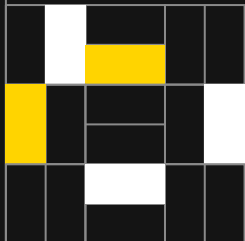
# Implementation Details

Make use of sets in c++, or lists in python:

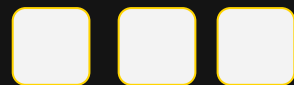
- Set containing known safe spaces
- Set of potentially unsafe spaces
- Set of absolutely unsafe spaces
- Total set of spaces

If the robot must try and traverse a space that does not exist in the known safe spaces (all spaces that it could travel to have already been visited or are absolutely unsafe), then choose a space with the lowest probability of having something.

- Use a depth first search to return a list of spaces to visit in order to get to that potentially unsafe space
- If there is an equal probability for two or more of the lowest probability spaces, choose one at random



0	1	2	3	4
5	6	7	8	9
10	11	12	13	14
15	16	17	18	19
20	21	22	23	24



PITS



CROCOTTA



PLAYER



# Thanks!

[tuckerjake11@gmail.com](mailto:tuckerjake11@gmail.com)

(970) 290-6998  
[linkedin.com/jaketuckerr](https://www.linkedin.com/jaketuckerr)

**CREDITS:** This presentation template was created by **Slidesgo**, and includes icons by **Flaticon**, and infographics & images by **Freepik**

