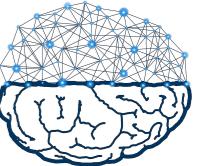


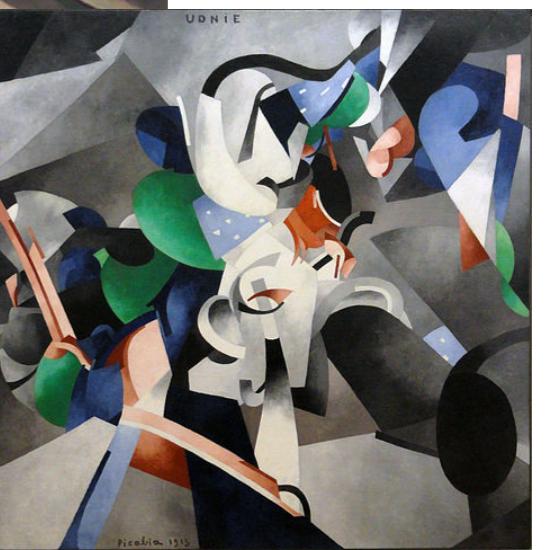


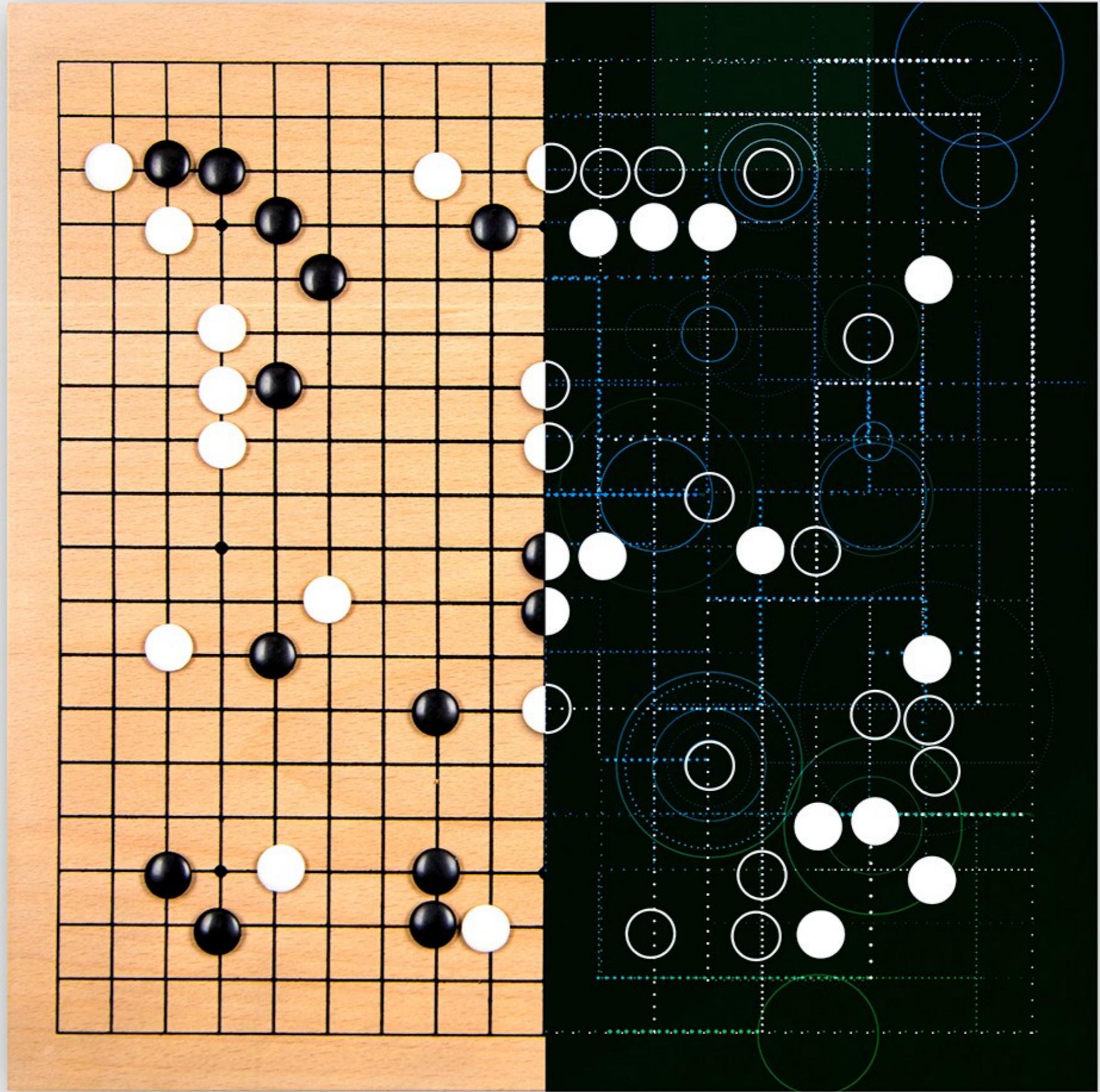
GDAŃSK UNIVERSITY
OF TECHNOLOGY



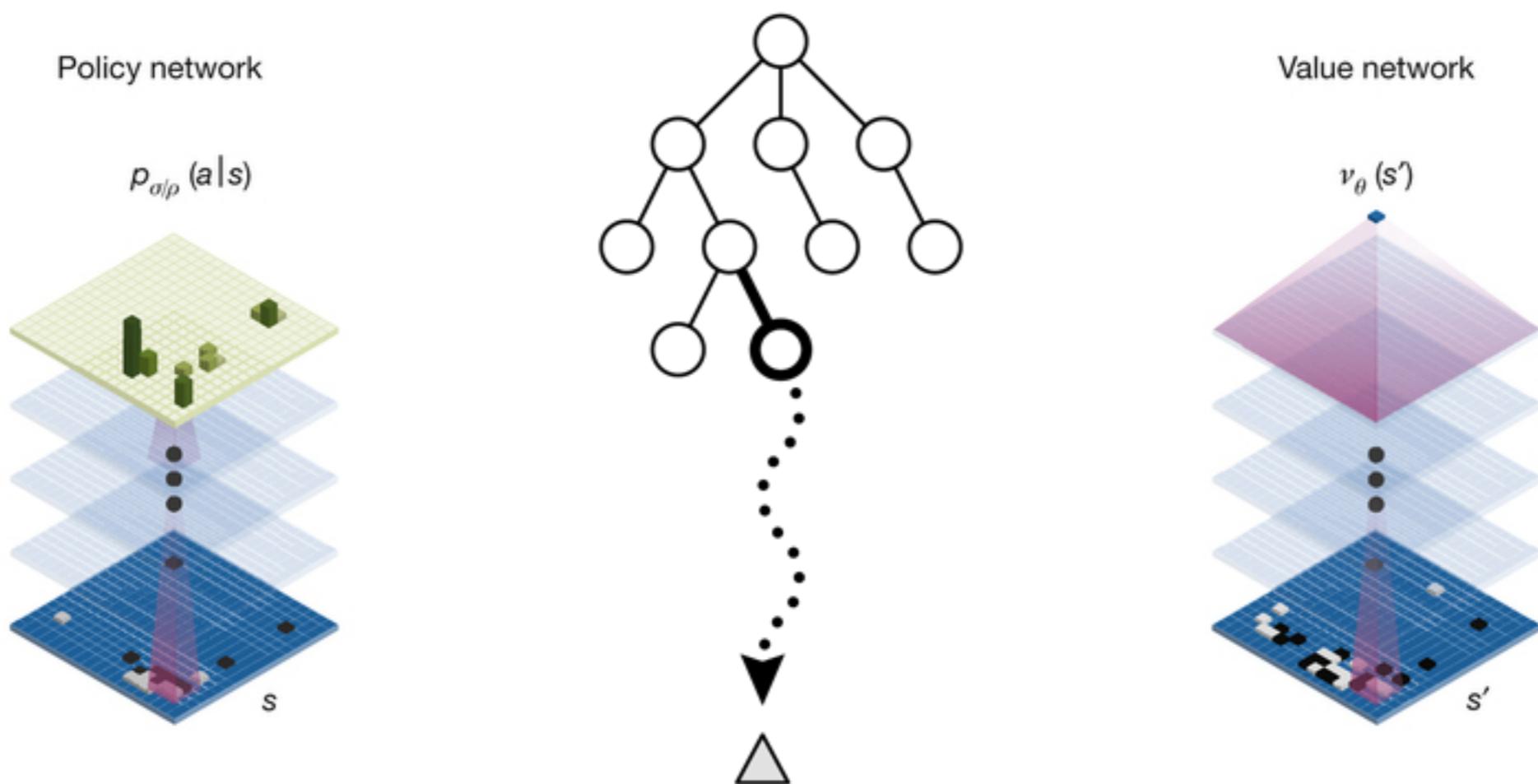
Deep Reinforcement Learning through AlphaZero lenses

Piotr Januszewski





Guided Tree Search

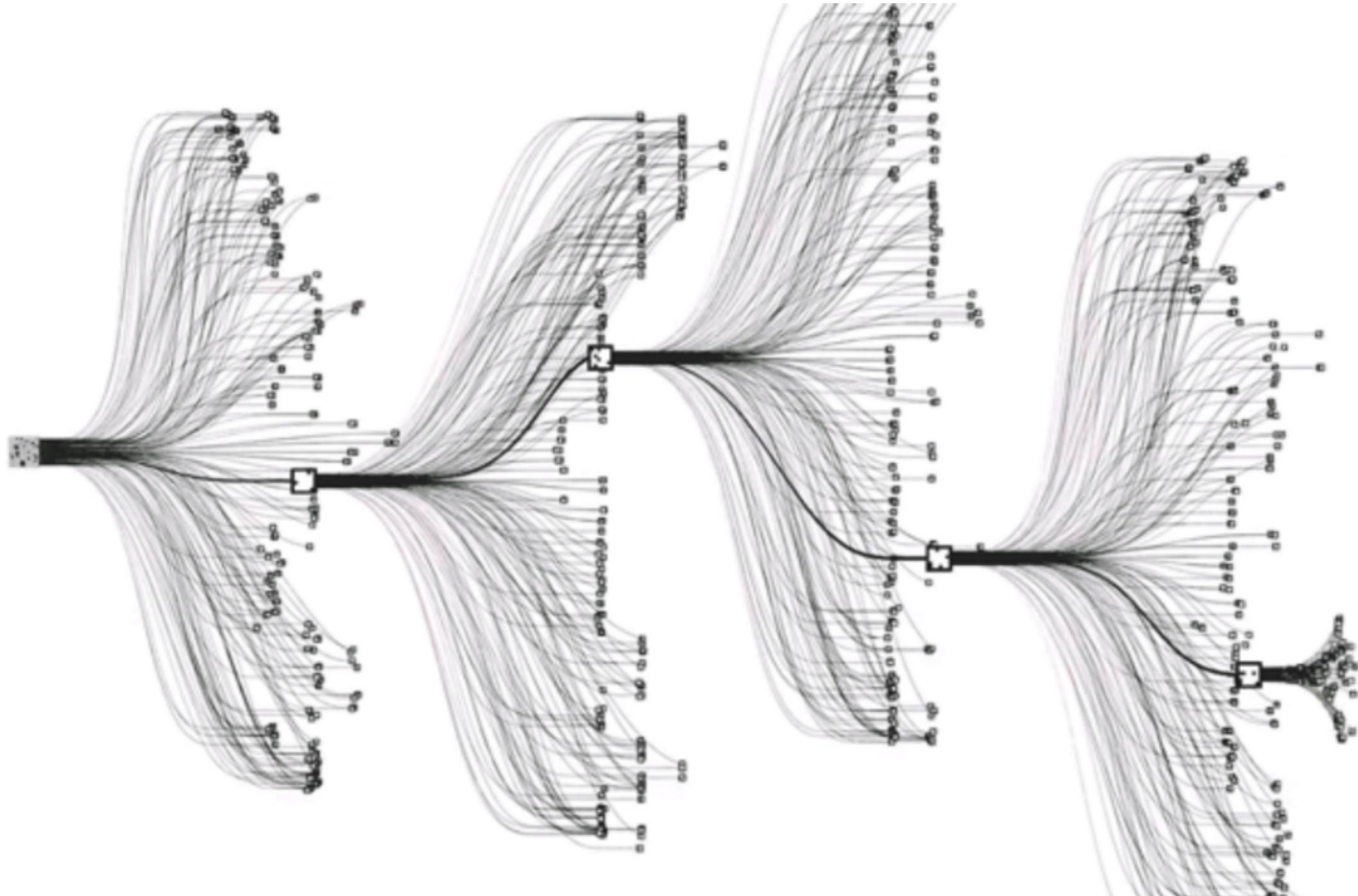


DeepMind's AlphaGo

AlphaGo Lee version of this algorithm became **the first AI to beat a human grandmaster of Go** in March of 2016.

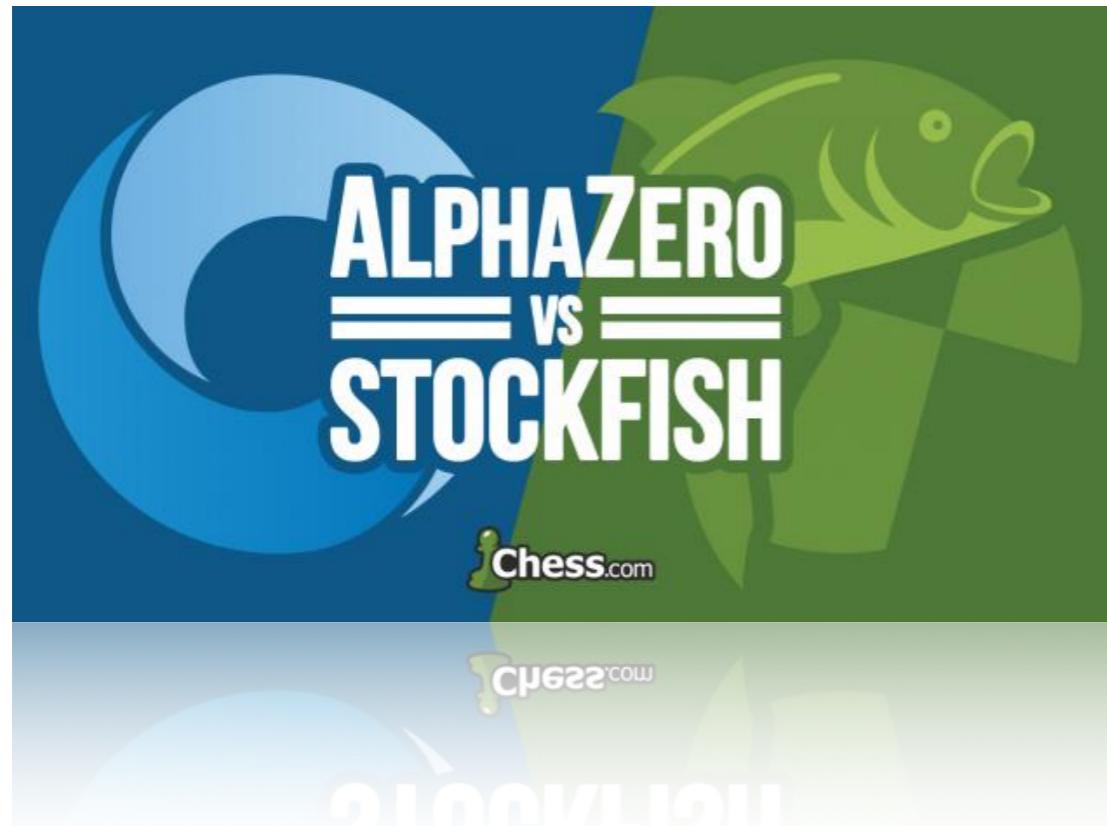
It is huge achievement as for a long time people thought that **Go is the holy grail of artificial intelligence** and we were still many years before AI will beat humans in this game!





The ancient Game of **Go** has more states then there are atoms in the known universe and it's branching factor is more then seven times bigger then the one of chess.

DeepMind's AlphaZero

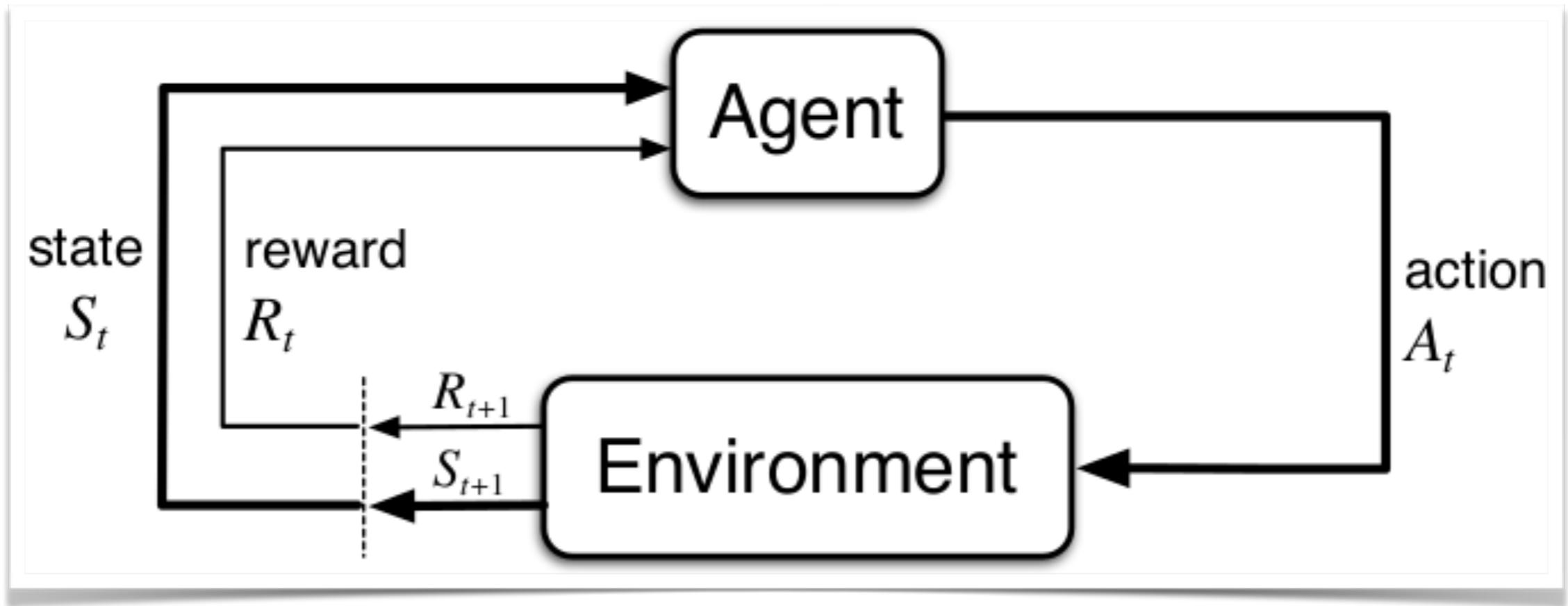


In October of 2017, DeepMind published new AlphaGo version. **AlphaGo Zero had defeated AlphaGo 100–0**. Incredibly, it had done so by learning solely through self-play. No longer was a database of human expert games required to build a super-human AI.

48 days later they publish current state-of-the-art version AlphaZero, which can play other games like Chess and Shogi. **It can beat the world-champion programs StockFish (for Chess) and Elmo (for Shogi) learning through self-play for only about 4 and 2 hours respectively!**

Reinforcement Learning

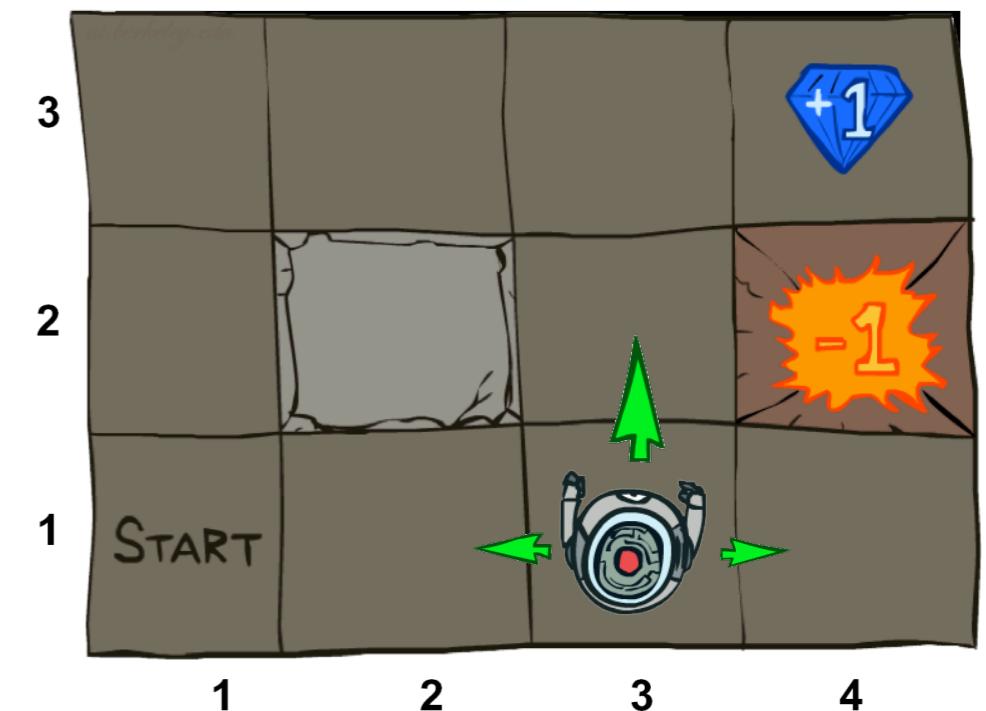
The background features a complex network of white lines and dots on a dark blue gradient. The lines form a dense, organic pattern that resembles a neural network or a complex graph. The dots are small and white, scattered throughout the network. The overall effect is a sense of depth and complexity, suggesting a sophisticated machine learning or data analysis process.



- Learning by trial-and-error, in real-time.
- Improves with experience
- Inspired by psychology – Agent + Environment
 - Agent selects actions to maximise total reward (return)

Example: Grid World

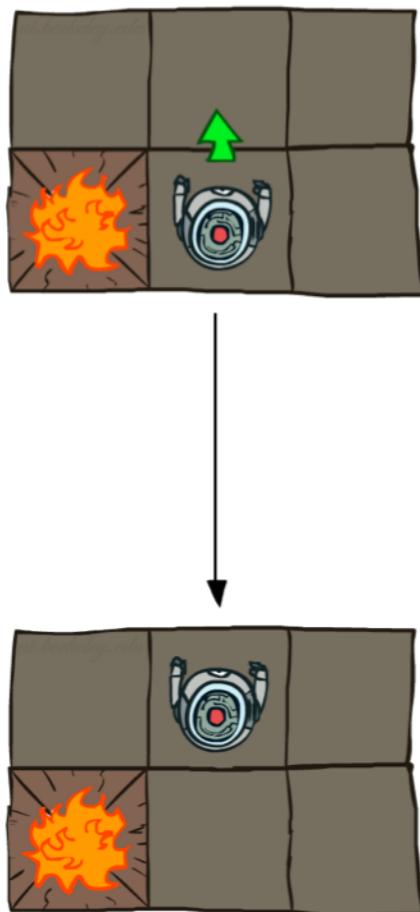
- A maze-like problem
 - The agent lives in a grid
 - Walls block the agent's path
 - States are robot positions
- The agent receives rewards each time step
 - Small “living” reward each step (can be negative)
 - Big rewards come at the end (good or bad)
- Goal: maximise sum of rewards



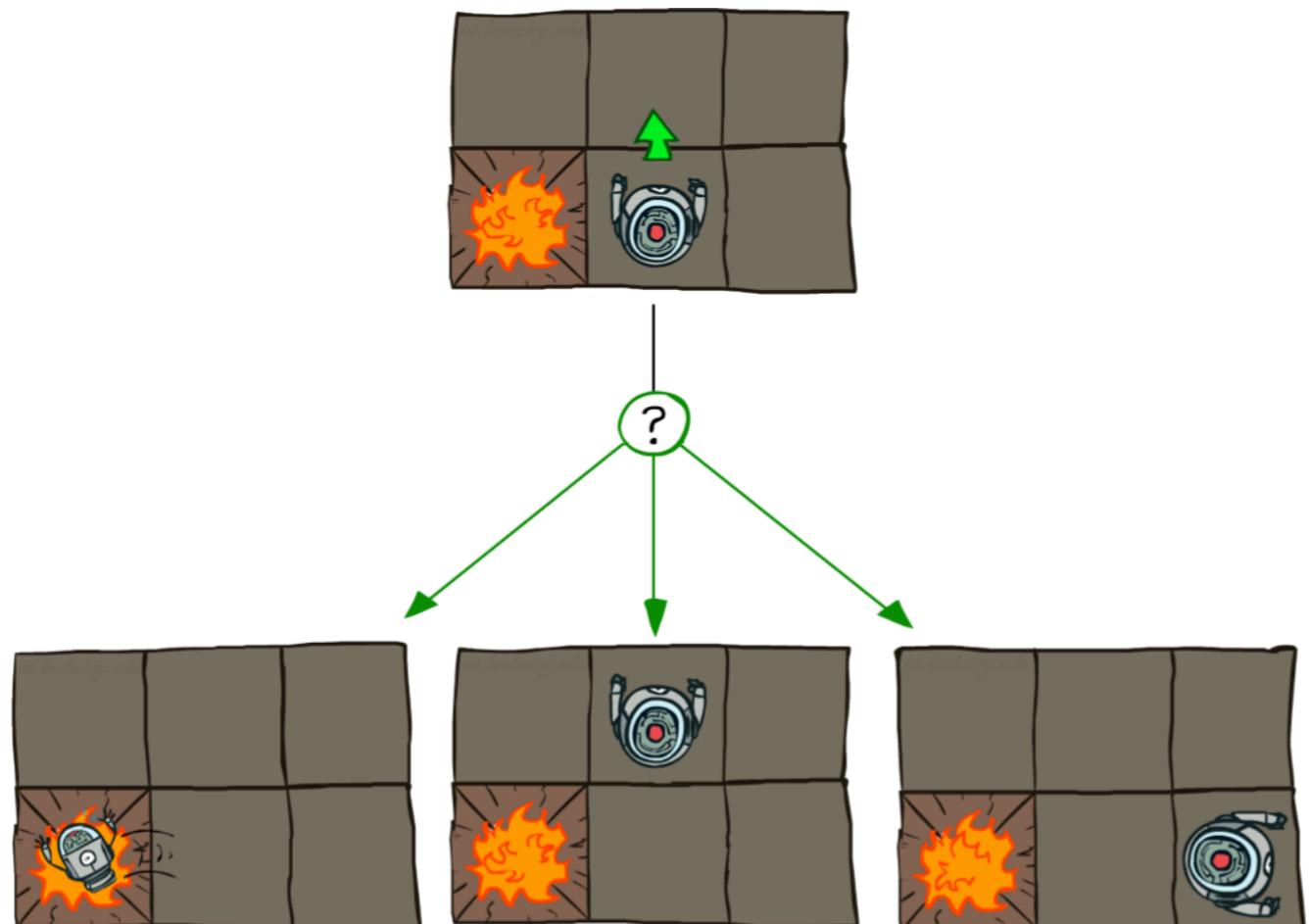
Source: [UC Berkeley CS188 Intro to AI](#)

Grid World Actions

Deterministic Grid World



Stochastic Grid World



Source: [UC Berkeley CS188 Intro to AI](#)

When to use RL?

- Data in the form of trajectories.
- Need to make a sequence of (related) decisions.
- Observe (partial, noisy) feedback to choice of actions.
- Tasks that require both learning and planning.

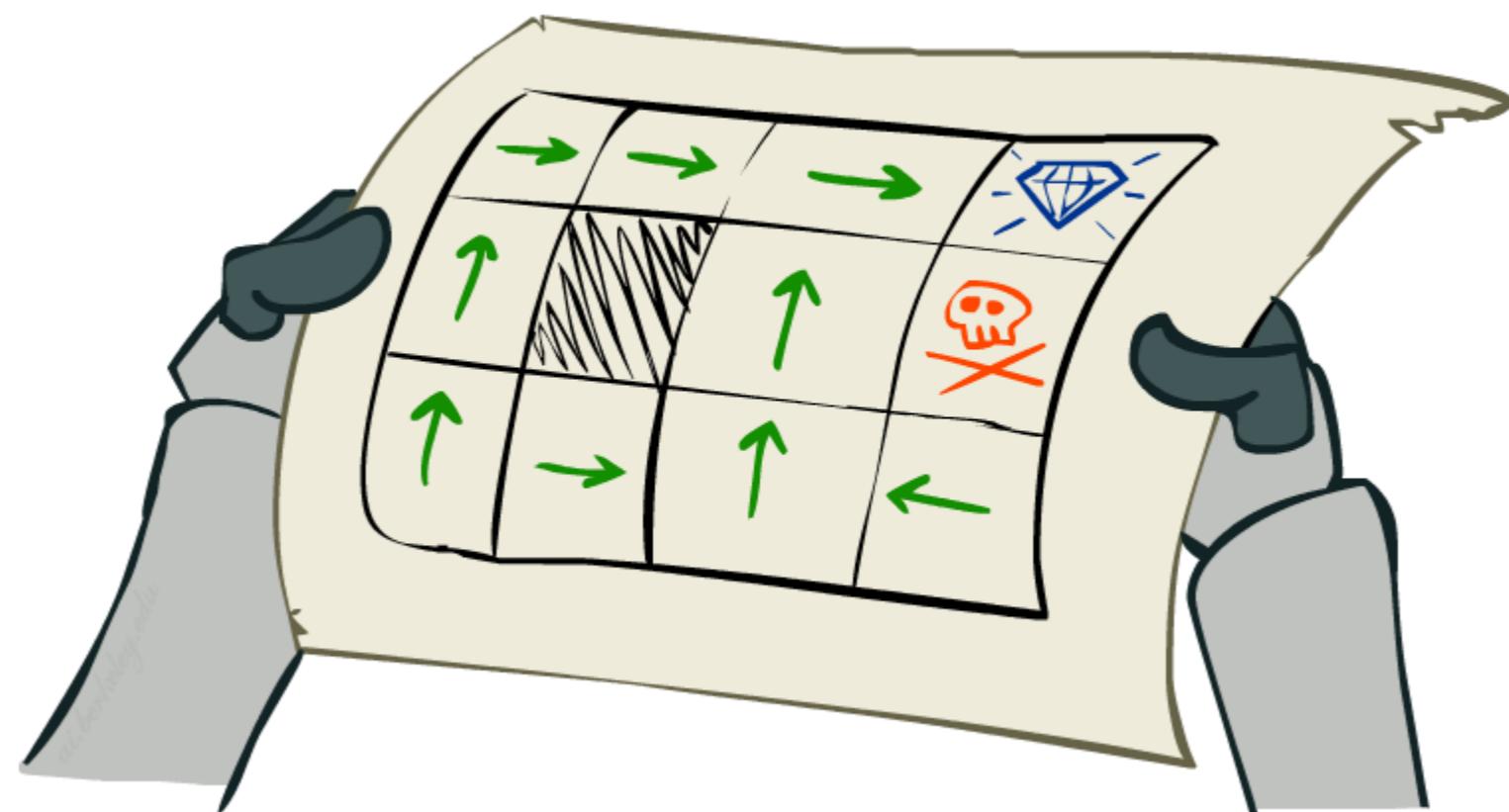
Where is it used?

- Robotics
- Video games
- Conversational systems
- Medical intervention
- Algorithm improvement
- Improvisational theatre
- Autonomous driving
- Prosthetic arm control
- Financial trading
- Query completion

RL applications at RLDM 2017

Policy Iteration

Policy tells agent how it should behave



Expected return

$$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T,$$

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1},$$

G_t – future expected (discounted) return from time t

R_{t+i} – random variable, reward at time $t + i$

γ – discount factor



1

Worth Now



γ

Worth Next Step



γ^2

Worth In Two Steps

State value

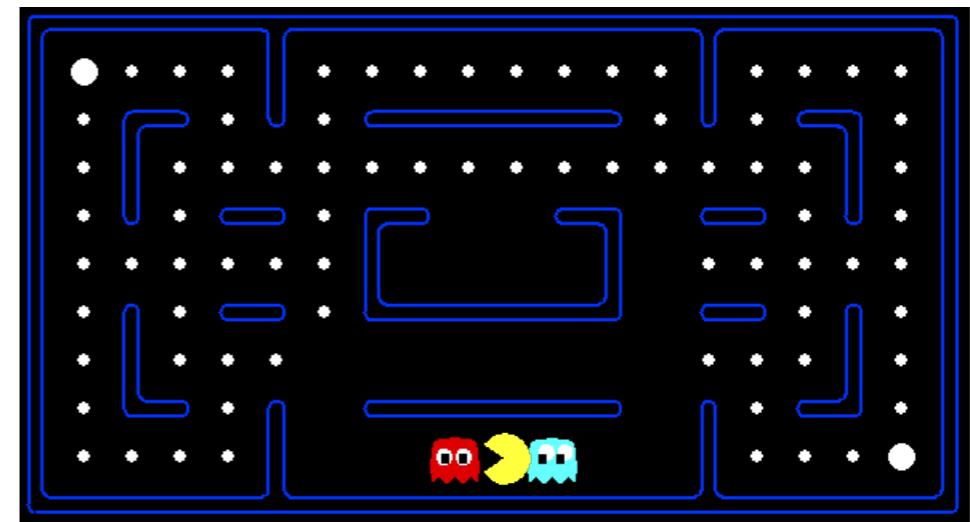
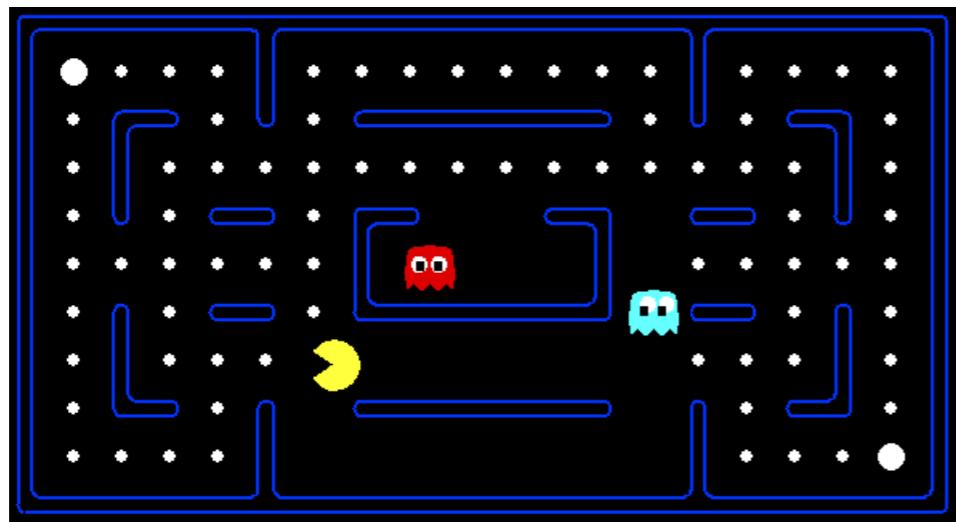
$$v_{\pi}(s) = \mathbb{E}_{\pi}[G_t | S_t = s] = R_{t+1} + R_{t+2} + R_{t+3} + \dots = r + v_{\pi}(s')$$

for $s' = P(s, a)$, $r = R(s, a, s')$

π – policy, decide on action a in state s

P – dynamics model, simulates environment

R – reward function, returns transition reward



Policy evaluation

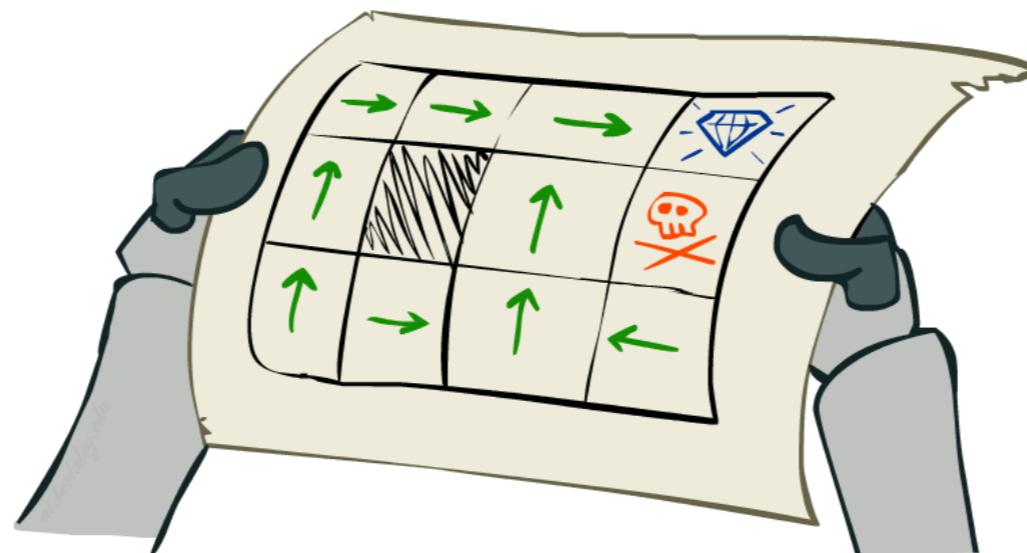
$$v_{k+1}(s) \doteq \mathbb{E}_{\pi}[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s]$$



Policy improvement

$$\begin{aligned}\pi'(s) &\doteq \arg \max_a q_\pi(s, a) \\ &= \arg \max_a \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a]\end{aligned}$$

q_π – state-action quality function

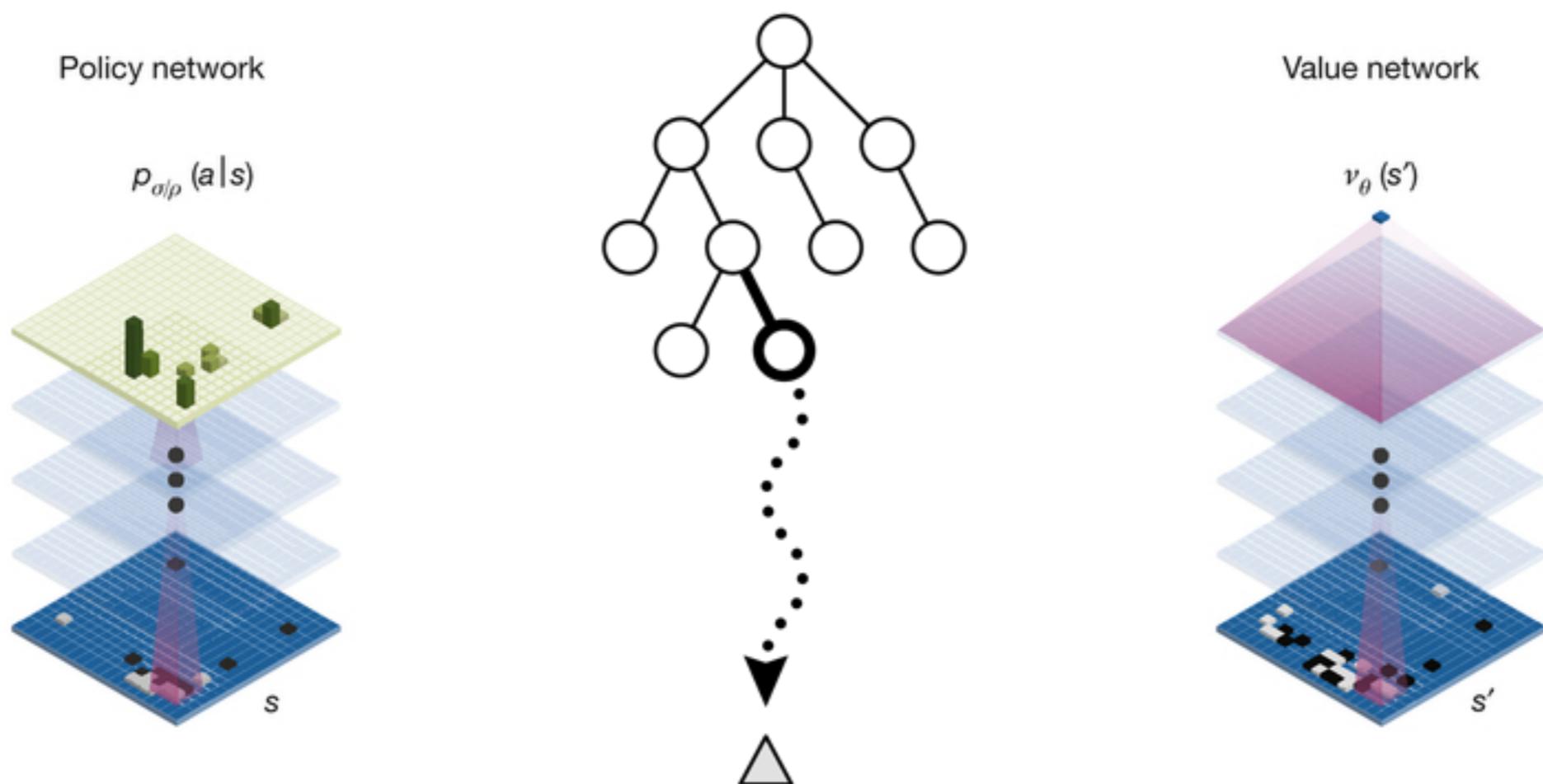


**AlphaZero is a General
Policy Iteration algorithm**



Planning and
learning

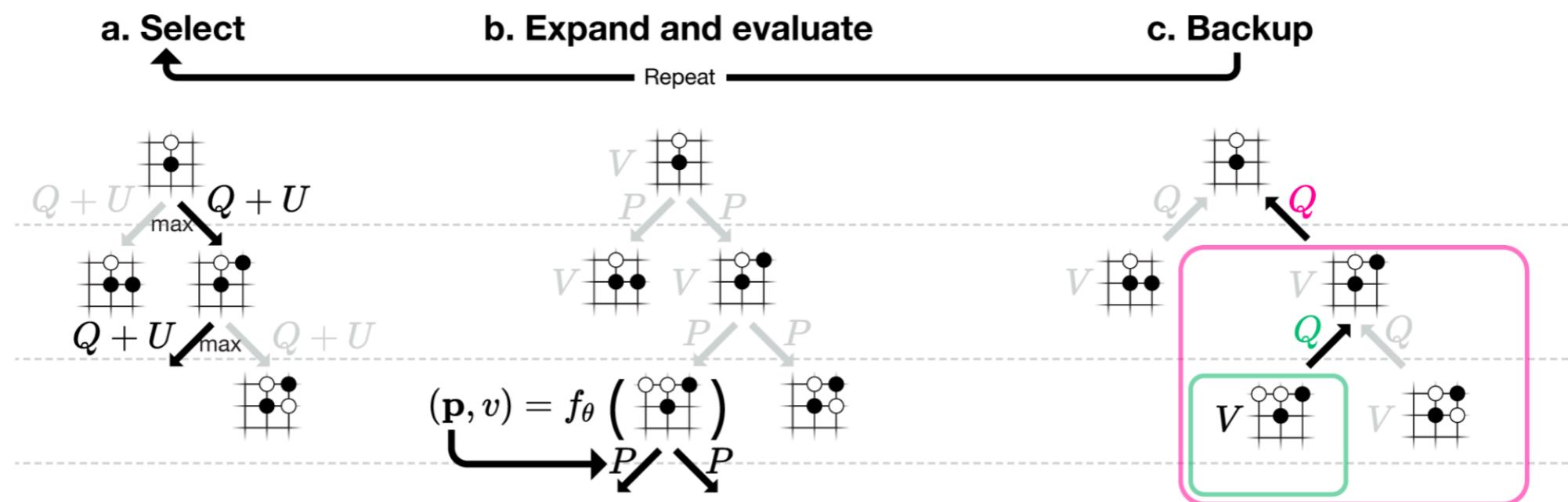
Guided Tree Search



Requirements on problem kind

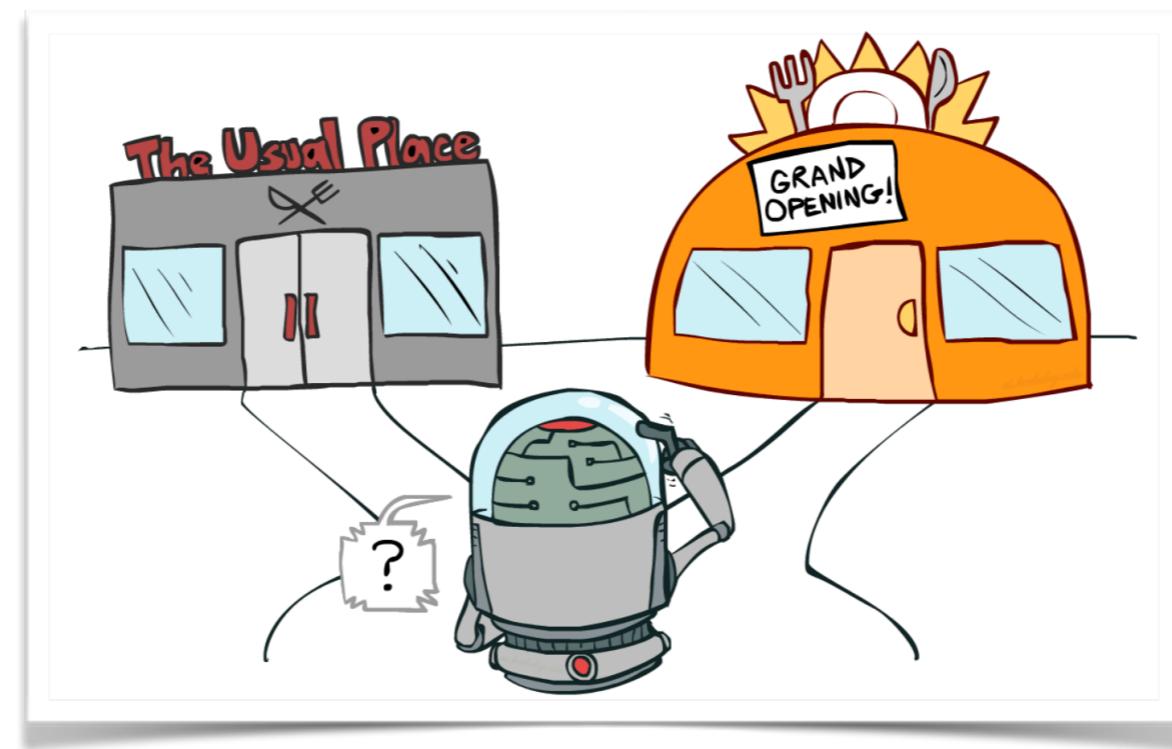
- **discrete** - states and actions sets form a discontinuous sequences, put simply they are discrete.
- **deterministic** - every move has one possible outcome.
- with **perfect information** - both players have perfect knowledge about current game state.
- access to **game rules** - there need to be some sort of environment simulator.

AlphaZero Tree Search



This algorithm meets **anytime property**. Estimates gets better and better with each simulation and we can cut calculations any time we have to.

Exploration-exploitation



Upper Confidence Tree

$$U_i = \frac{W_i}{N_i} + c \sqrt{\frac{\ln N_p}{N_i}}$$

We choose action that maximises U_i (UCT value of particular action)

Efficiency Through Expert Policies

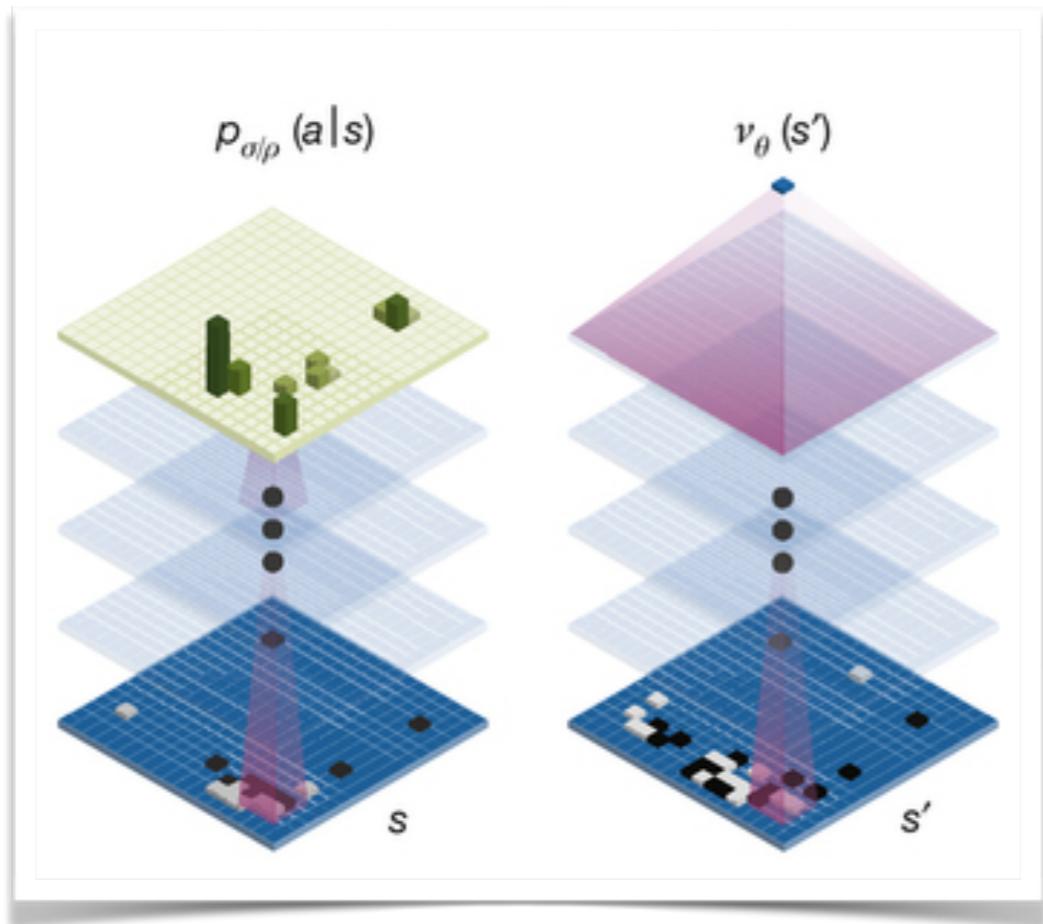
Games like chess  and Go  have very (in Go I mean VERY) large branching factor and number of legal positions.

It would be nice to have some expert policy, that would tell us which moves are good and which are bad. So we can focus on exploring only those promising moves.

Any ideas how to obtain expert policy?

$$U(s, a) = Q(s, a) + c_{puct} \cdot P(s, a) \cdot \frac{\sqrt{\Sigma_b N(s, b)}}{1 + N(s, a)}$$

AlphaZero Neural Net



How to model those two?

Deep Learning
for rescue!



Train through Policy Iteration

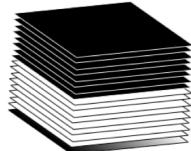
SELF PLAY

Create a 'training set'

The best current player plays 25,000 games against itself

See MCTS section to understand how AlphaGo Zero selects each move

At each move, the following information is stored



The game state
(see 'What is a Game State section')



The search probabilities
(from the MCTS)



The winner
(+1 if this player won, -1 if this player lost - added once the game has finished)

RETRAIN NETWORK

Optimise the network weights

A TRAINING LOOP

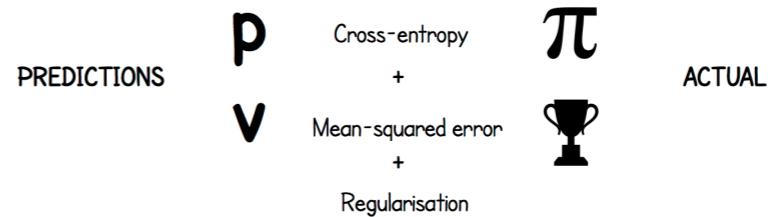
Sample a mini-batch of 2048 positions from the last 500,000 games

Retrain the current neural network on these positions

- The game states are the input (see 'Deep Neural Network Architecture')

Loss function

Compares predictions from the neural network with the search probabilities and actual winner



After every 1,000 training loops, evaluate the network

Full infographic in references, highly recommend to give a look

Sources and Good Reads

- [A Simple Alpha\(Go\) Zero Tutorial](#) by Surag Nair
- [AlphaGo Zero cheatsheet](#) by David Foster
- [AlphaGo Zero - How and Why it Works](#) by Tim Wheeler
- [Deep Reinforcement Learning: Pong from Pixels](#) by Andrej Karpathy
- [UC Berkeley AI course](#)
- [Mastering the game of Go with deep neural networks and tree search](#) by David Silver et al.
- [Mastering the game of Go without Human Knowledge](#) by David Silver et al.
- [The Multi-Armed Bandit Problem and Its Solutions](#) by Lilian Weng
- [Deep Learning and Reinforcement Learning Summer School, Montreal 2017](#)
- Reinforcement Learning: An Introduction 2nd edition
- **AlphaGo movie on Netflix** 

Case Study



This presentation and AlphaZero code

<https://github.com/piojanu>

Thank you 😊