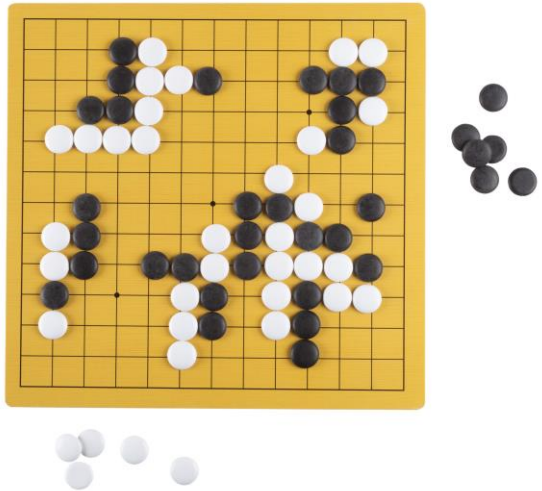
Game AI Programming



Part one based on Prof. Horst Eidenbergers Lecture on Strategy Game Programming @TU Vienna 2019

Part two based on David Silvers slides and Lectures, at al.

Prepared by Lukas Steindl, September 21

Introduction

Lukas Steindl,

10+ years at Microsoft mainly working with database systems

BSc. in Business Informatics from University of Hagen

Currently working on my master thesis https://github.com/embedded-machine-learning/ragweed-drone-detection as part of the Data Science Master @ TU Vienna

https://github.com/LukasSteindl

Agenda

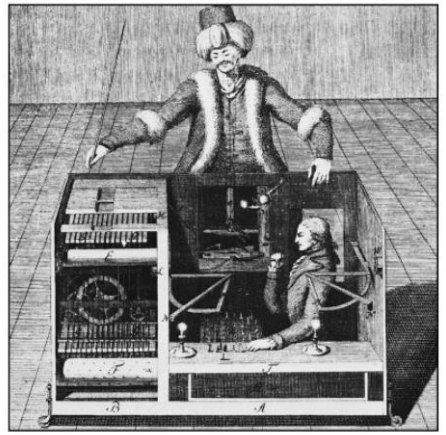
- Traditional Game AI Algorithms
  - Minimax algorithm
  - Alpha Beta Pruning
  - Monte Carlo Tree Search
- A brief introduction to Reinforcement Learning
- Microsoft Offerings
  - Deep Reinforcement Learning (RL) with Azure ML
- If time permits: AlphaGo, AlphaGo Zero, AlphaZero, MuZero

Lab Project

- Students should be able to implement a Game AI for the Mancala Game Engine. (by Robert Fischer https://github.com/metzzo)

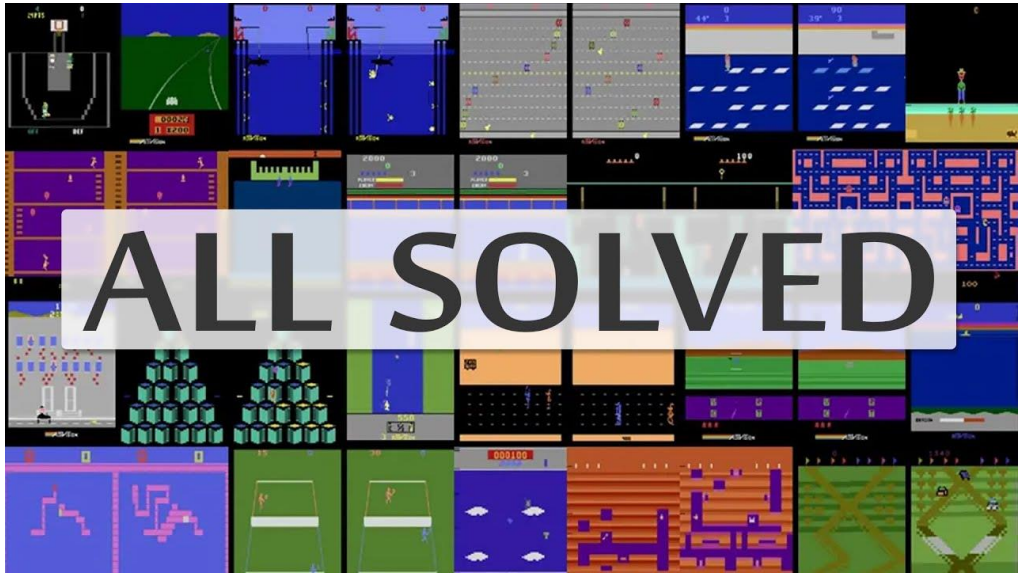
- Agents will play against each other!

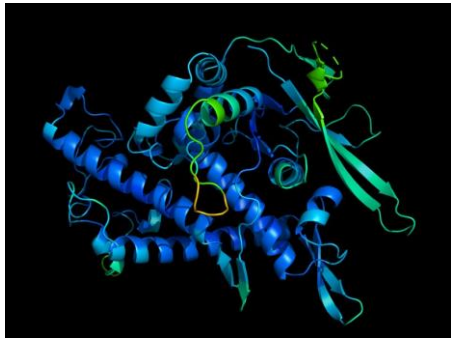
History of Game Machines



- 18th century: The Turk: Fake chess machine
- 1970ies: First chess programs
- 1997: Deep Blue beats chess world champion
- 2015: DeepMind solved all Atari Games (Agent 57)
- 2016: AlphaGo beats Lee Sedol in the Game of GO
- 2017: OpenAI wins Dota2 in 1on1
- 2018: AlphaStar wins Starcraft
- 2019: OpenAI wins Dota2 in 5on5
- 2020 Deep Mind practically „solved“ Proteinfolding

despite beeing NP-Complete





What Makes a Game?

Dimension	Values
Reward system	Zero sum (strict competition), cooperative
Level of information	Fully observable, partially (fog of war)
Game determinism	Yes/no (elements of chance)
Move order	Sequential, simultaneous
Time structure	Discreet, real-time
Feedback type	Combinatorial (immediate rewards), real-world (delayed)

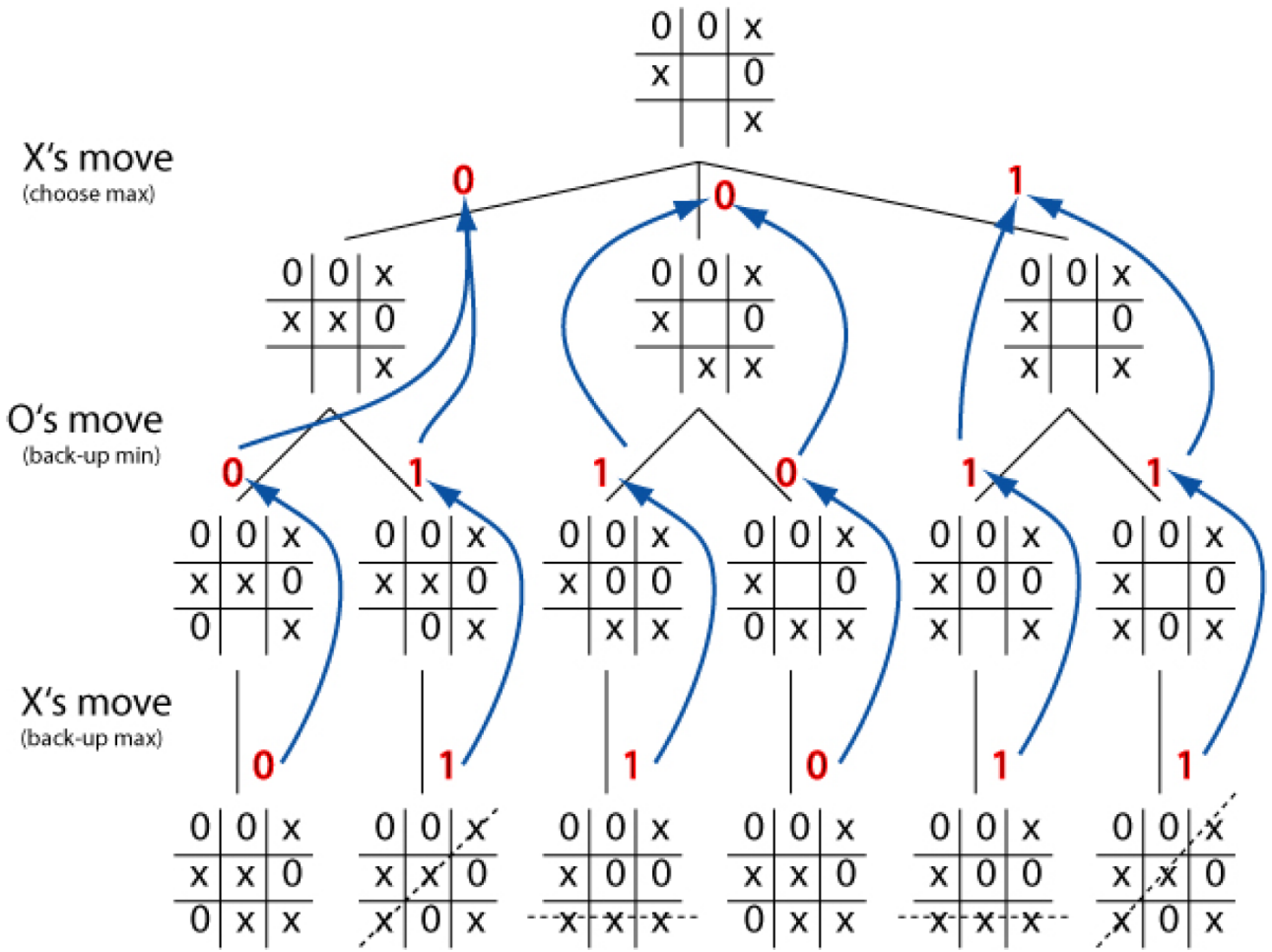
- Examples:
  - *Chess*: Zero sum, fully observable, deterministic, sequential, discreet, combinatorial
  - *Civilization:* Cooperative, partially observable, not deterministic, sequential, discreet, real-world.
  - *Command & Conquer*: Cooperative, partially observable, not deterministic, simultaneous, real-time, real-world

See www.ims.tuwien.ac.at for more information.

Algorithmic Concepts & Terms

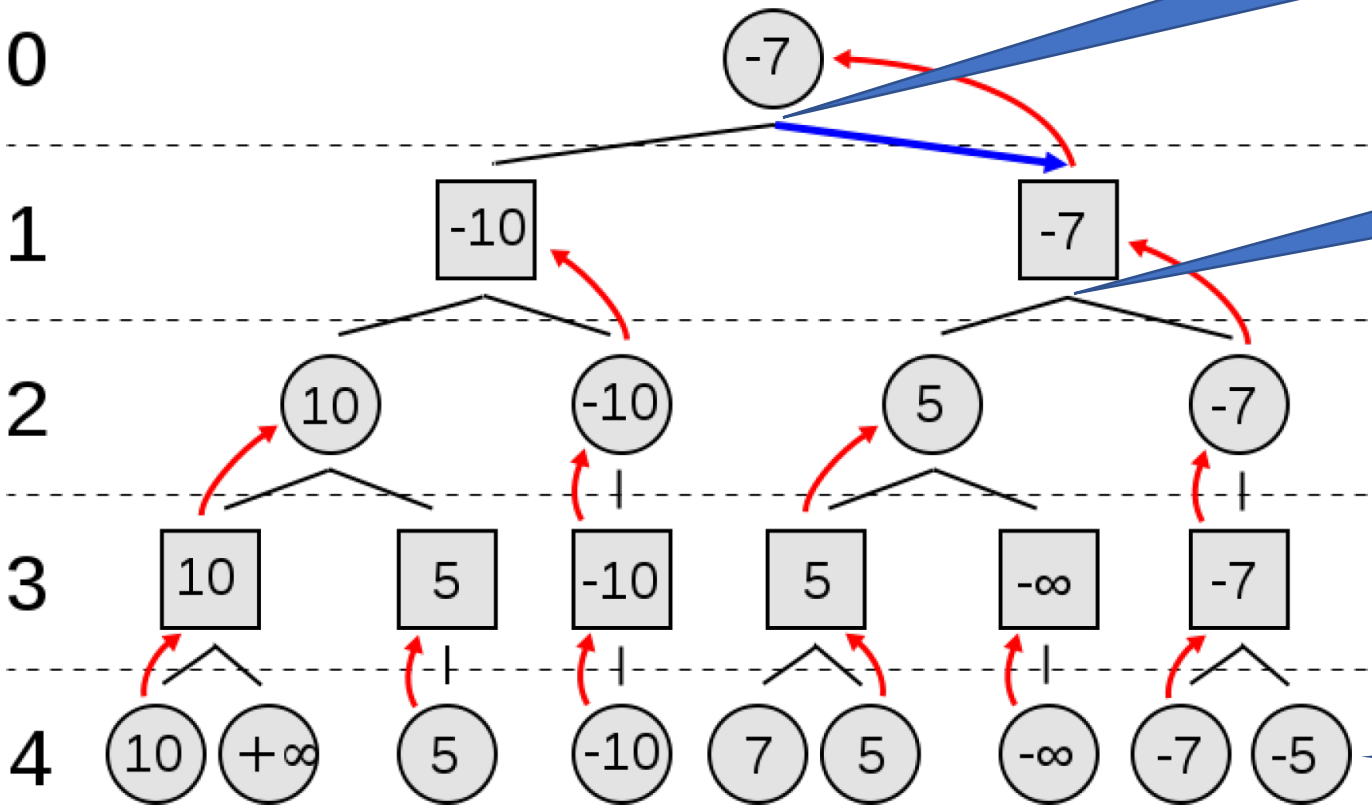
- What makes a strategy?
  - Rules, freedom of choice, rewards for actions
- Search strategies:
  - Dumb fast (e.g Minimax) -> Exhaustive tree search (trying out all moves)
  - Smart slow (e.g MCTS)
- Search stability:
  - Fail hard: deterministic, stable, expensive
  - Fail soft: probabilistic, heuristic, unstable, cheaper

tic toe- Dumb Fast: Minimax Algorithm



See www.ims.tuwien.ac.at for more information.

Numeric Example



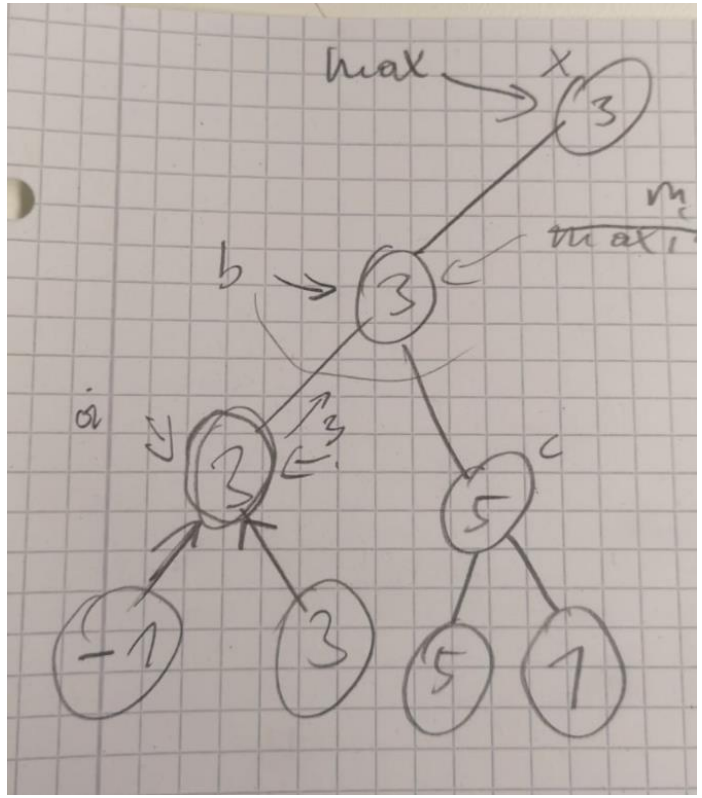
Even player always wants to maximize his reward

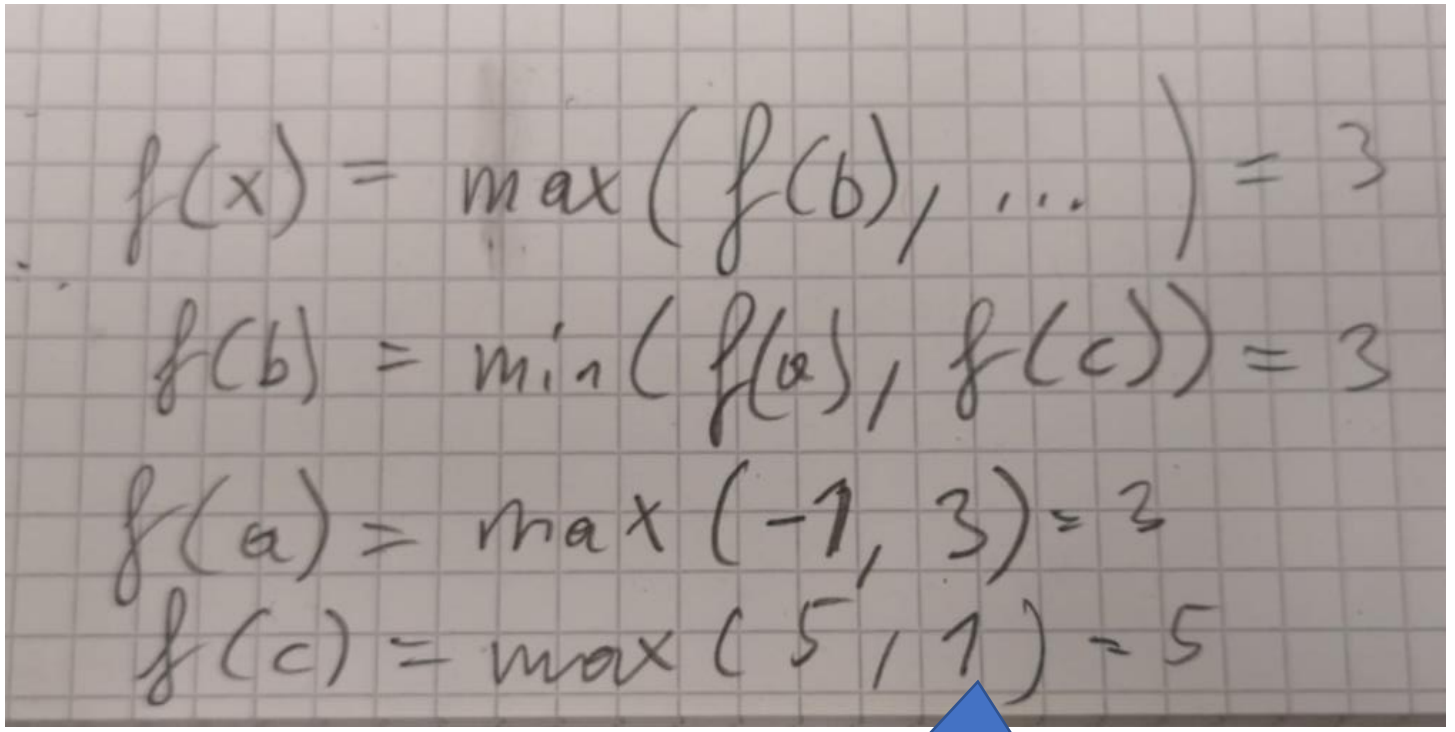
Odd player always wants to minimize his reward

Actual outcome for the even Player

See www.ims.tuwien.ac.at for more information.

Minimax all you need is a recursive function f(gamestate,player) to evaluate the value of every move that is currently possible.





Anchor of the recursion when the leave node is reached (depthcounter = 0)

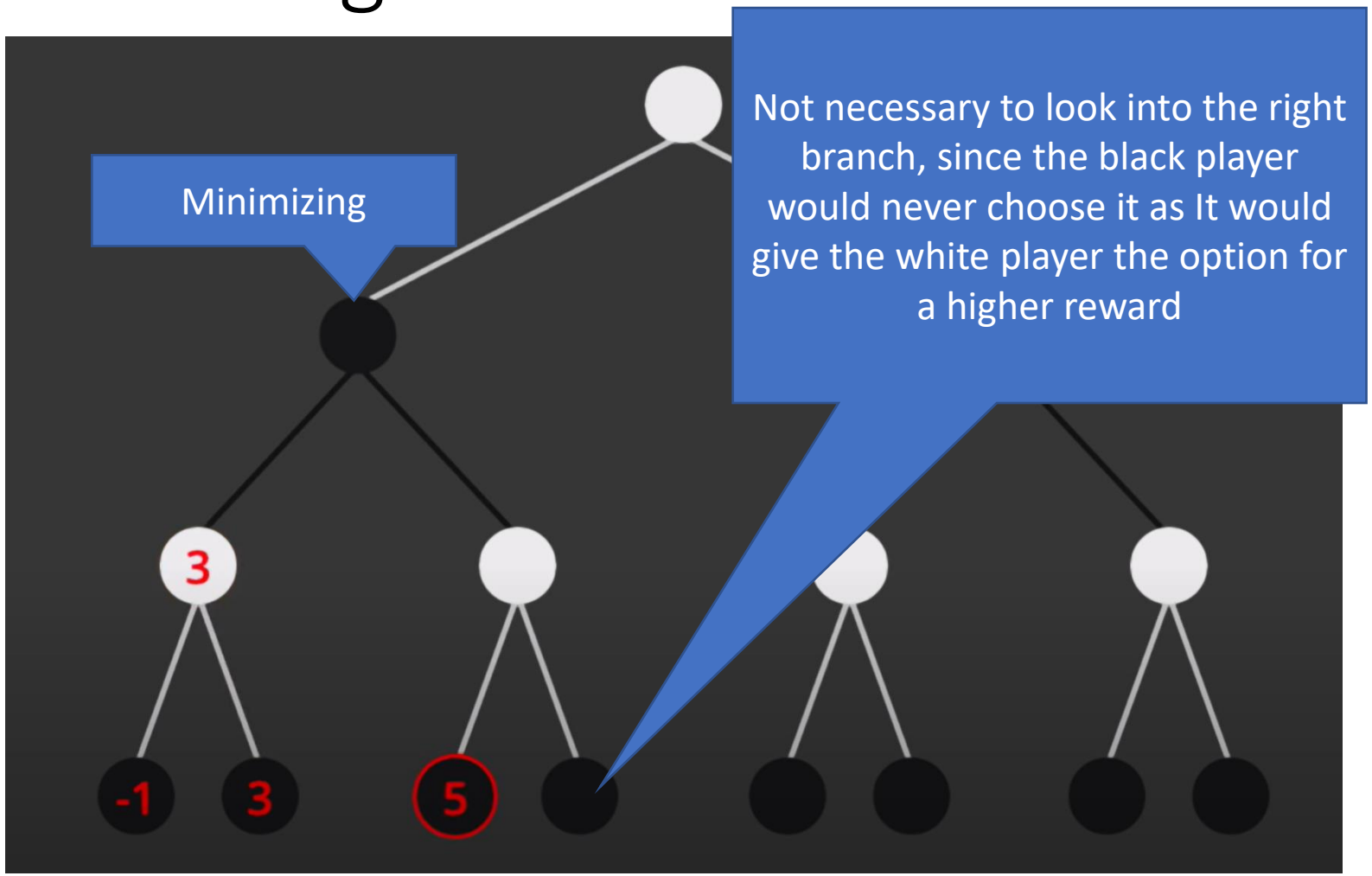
https://www.youtube.com/watch?v=l-hh51ncgDI&t=250s&ab_channel=SebastianLague

Alpha Beta Pruning

- Idea: do not explore paths when the player in the current position would never go that way.
- This can happen when doing a move would provide the opponent with a higher scoring move as already observed.

- https://www.youtube.com/watch?v=l-hh51ncgDI&t=250s&ab_channel=SebastianLague
-

Alpha Beta Pruning



- Long time the state of the art for chess (using domain expert evaluation functions)

Problems in Case of

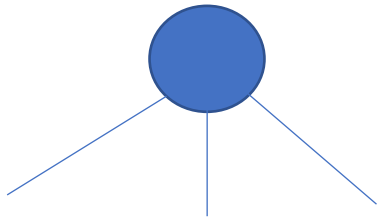
- High Branching Factor (e.g Game of GO)
- Missing Evaluation function

Monte Carlo Tree Search – slow and smart

- MCTS creates a statistics tree (describing the value of a position)
- Idea is to focus the search time to more relevant options.
- It comes up with these values by simulating steps (first random later more exploiting)
- Selection, Expansion, Simulation, Update -> https://www.youtube.com/watch?v=Fbs4lnGLS8M&ab_channel=FullstackAcademy

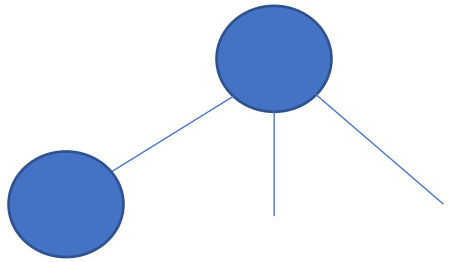
- Open Ended Environments (no domain dependent evaluation function required)

Selection, Expansion, Simulation, Update



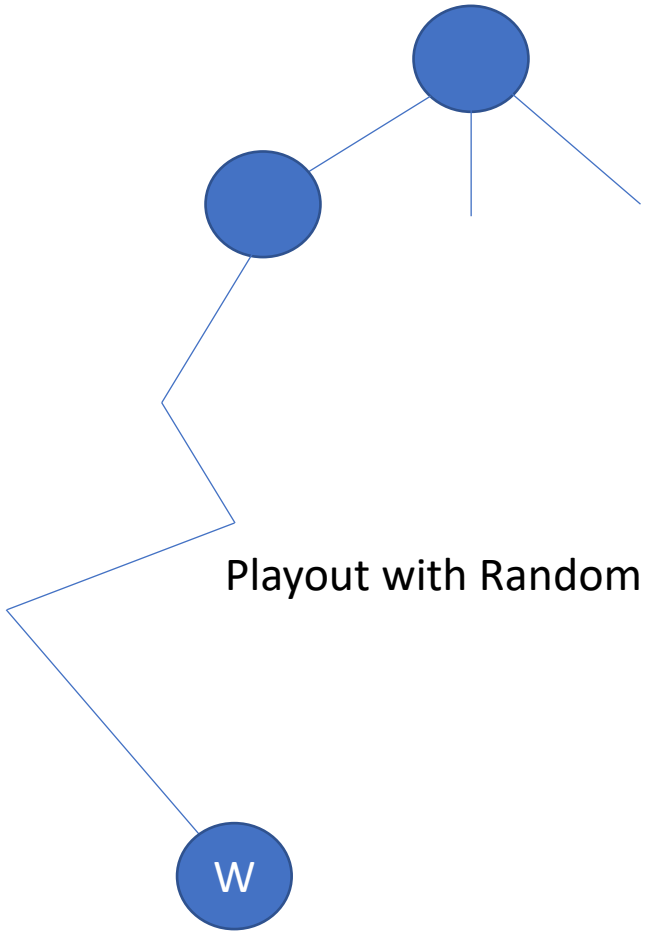
MCTS begins with a stats tree

Selection, Expansion, Simulation, Update



Expansion

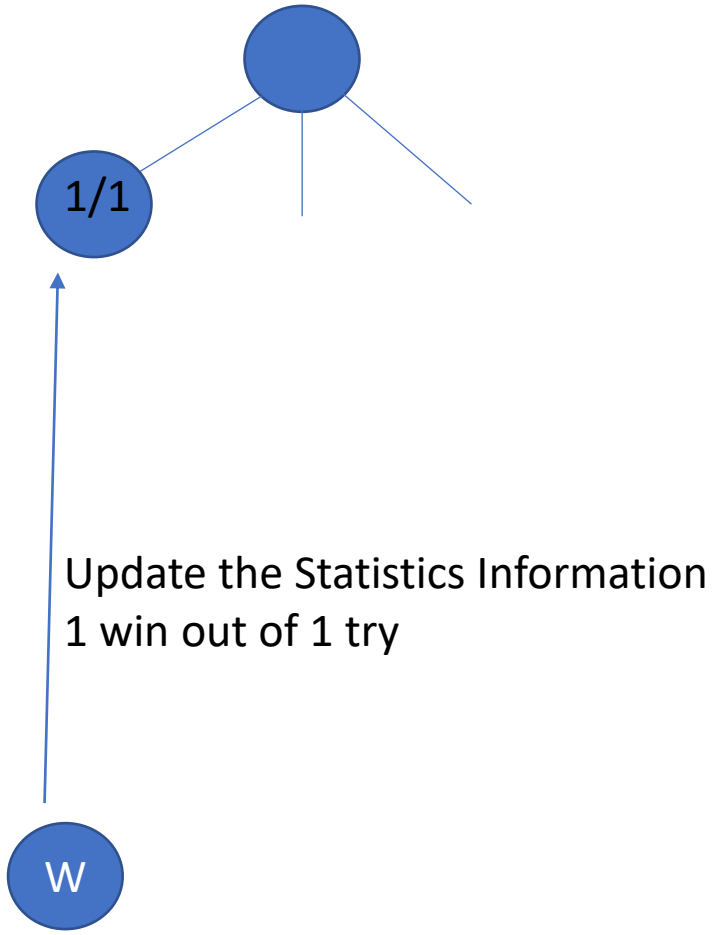
Selection, Expansion, Simulation, Update



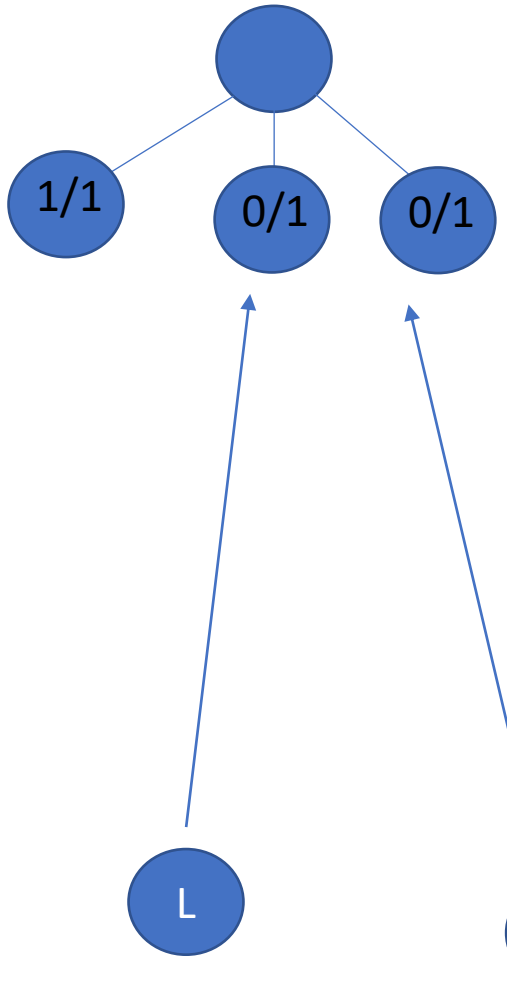
Simulation

Playout with Random Moves for itself and the opponent

Selection, Expansion, Simulation, Update



Selection, Expansion, Simulation, Update

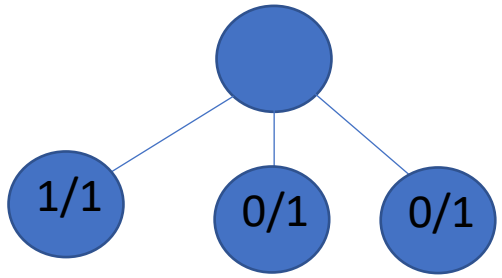


Same for the remaining actions

At this point the left node would be the best move!
But too small sample size to be sure!

Update the Statistics Information
0 win out of 1 try

Selection, Expansion, Simulation, Update

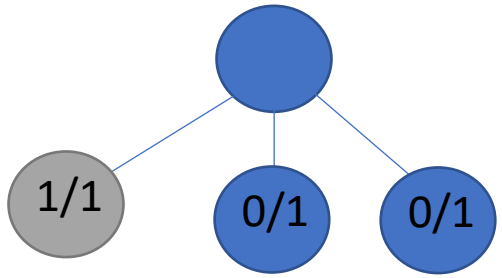


Selection

Now that all child nodes have been visited

The selection is based on how good are the stats?
How much has a child been ignored?

Selection, Expansion, Simulation, Update



Selection

Now that all child nodes have been visited

The selection is based on how good are the stats?
How much has a child been ignored?

$$\bar{x}_i \pm \sqrt{\frac{2 \ln n}{n_i}}$$

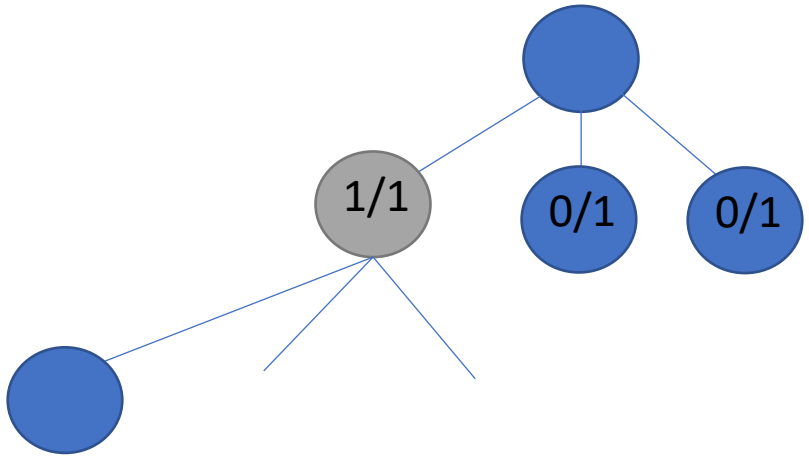
$\bar{x}_i$: Mean "value" of node

n_i: Number of simulations done for child node (i)

n: Number of total simulations done for all nodes

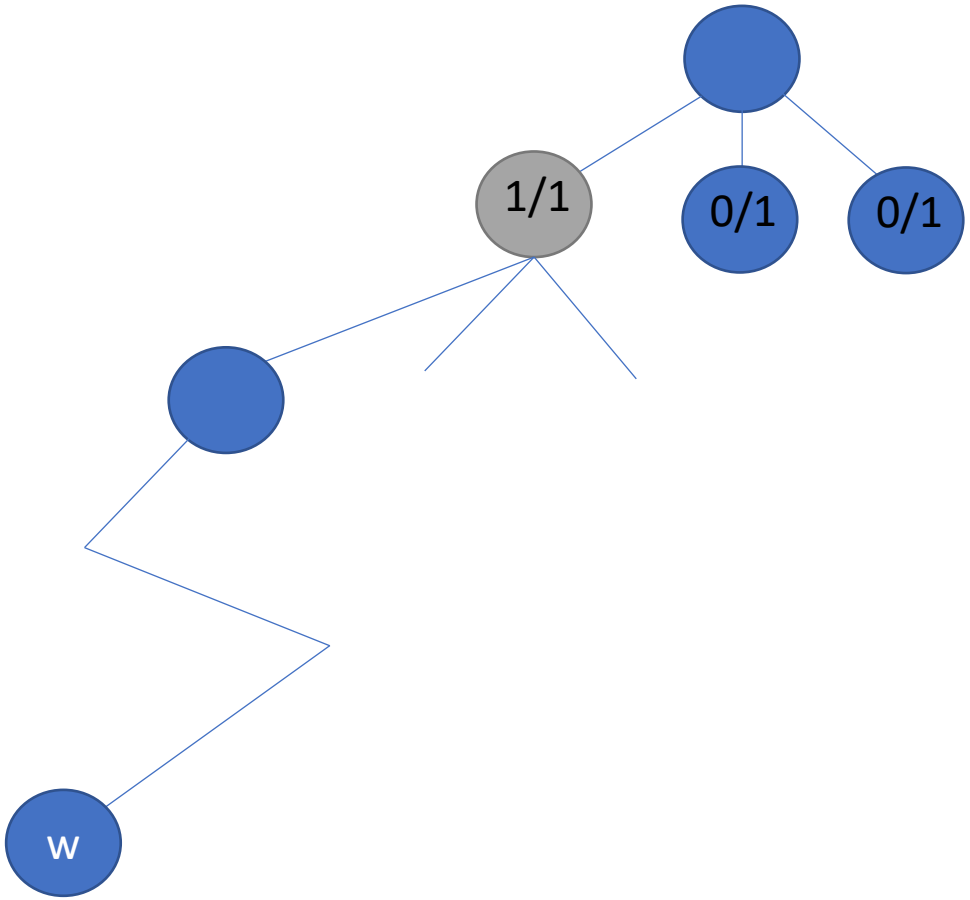
With this you maintain a good balance between Exploration and exploitation.

Selection, Expansion, Simulation, Update



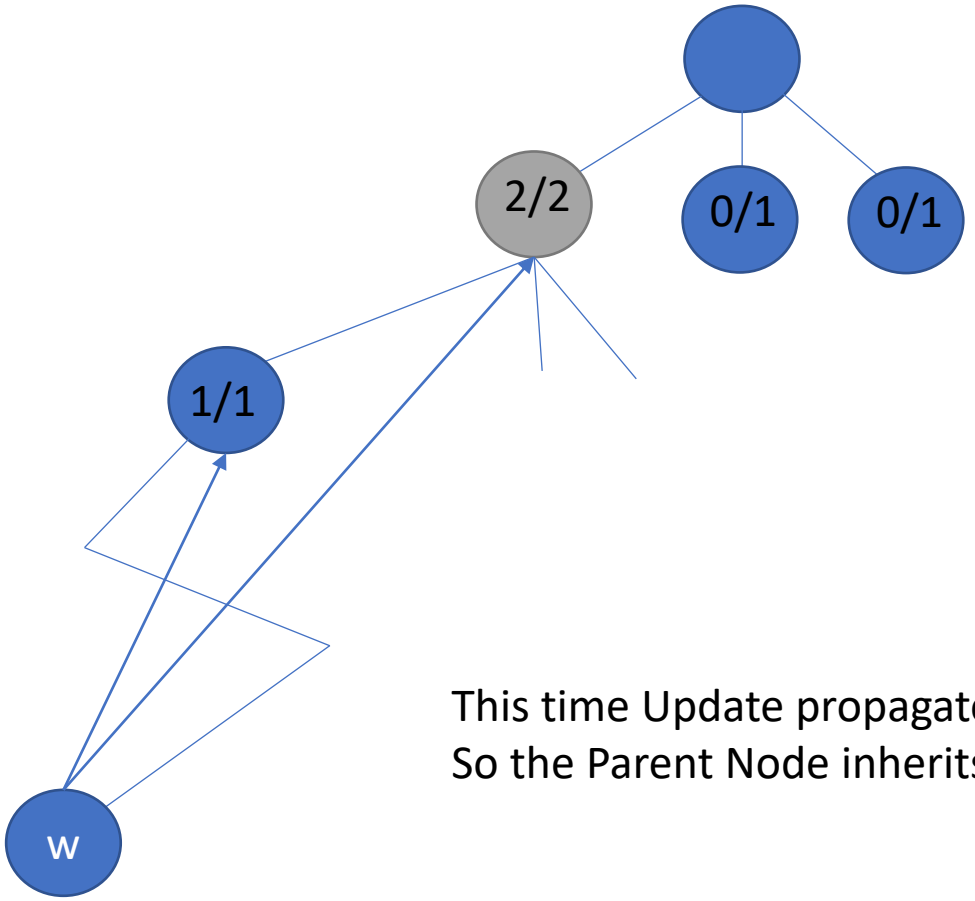
Expansion

Selection, Expansion, Simulation, Update



Simulation

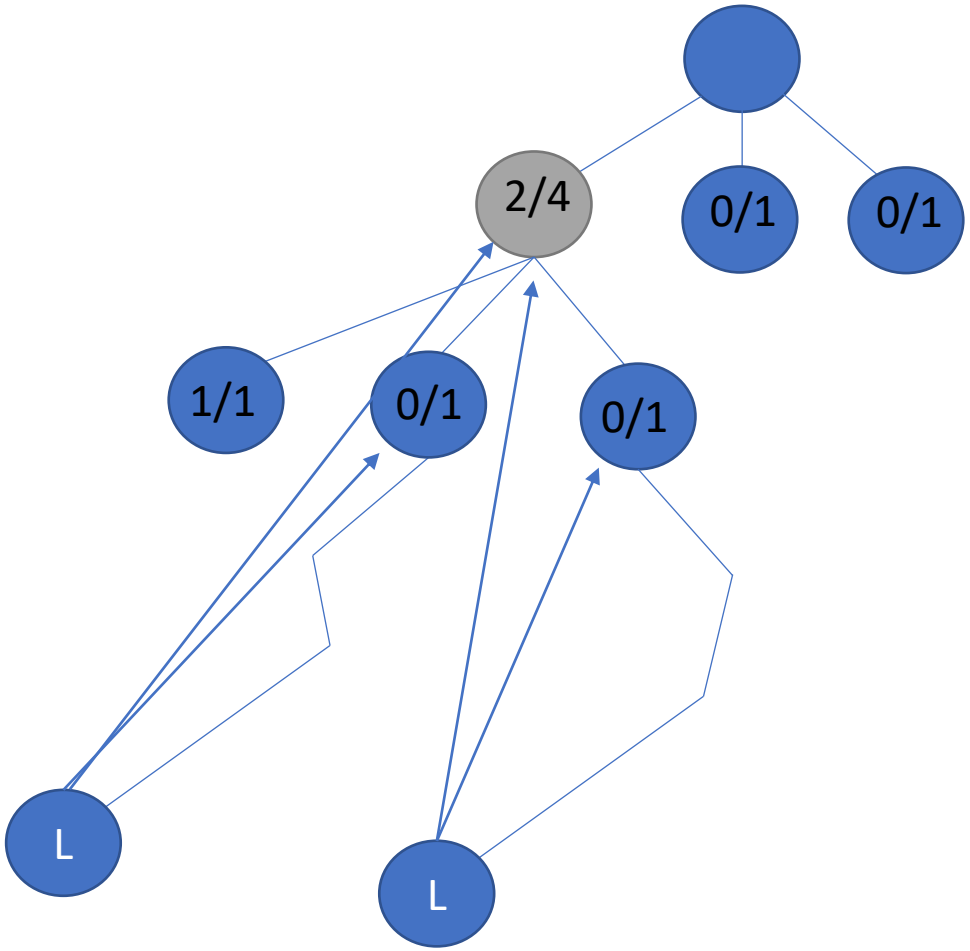
Selection, Expansion, Simulation, Update



Update

This time Update propagates back to the first child node.
So the Parent Node inherits the stats from its child nodes!

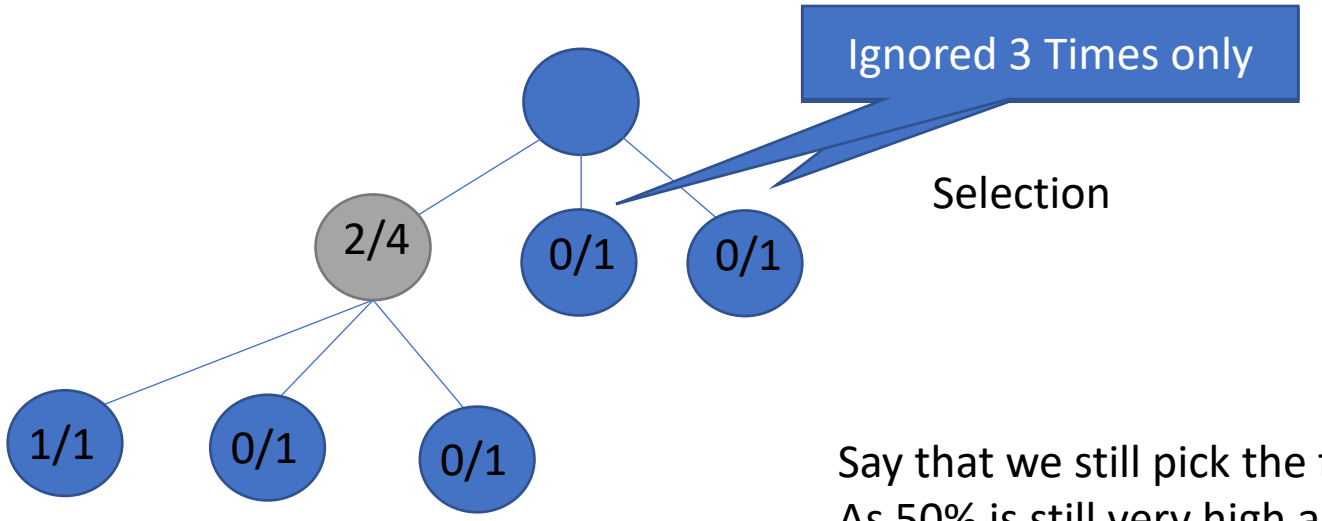
Selection, Expansion, Simulation, Update



Simulation & Update

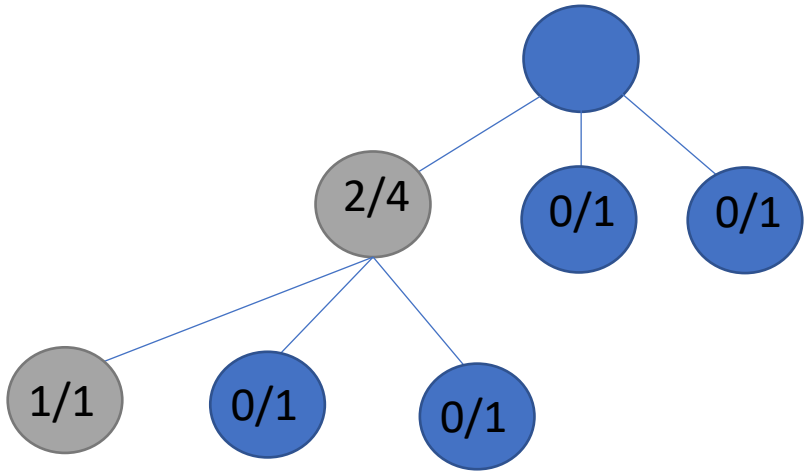
Now that the child nodes of the last selection
Have all been explored at least once,
It will run the selection from the top again!

Selection, Expansion, Simulation, Update



Say that we still pick the first child of the root.
As 50% is still very high and we only ignored the other Direct childs of the root three times!

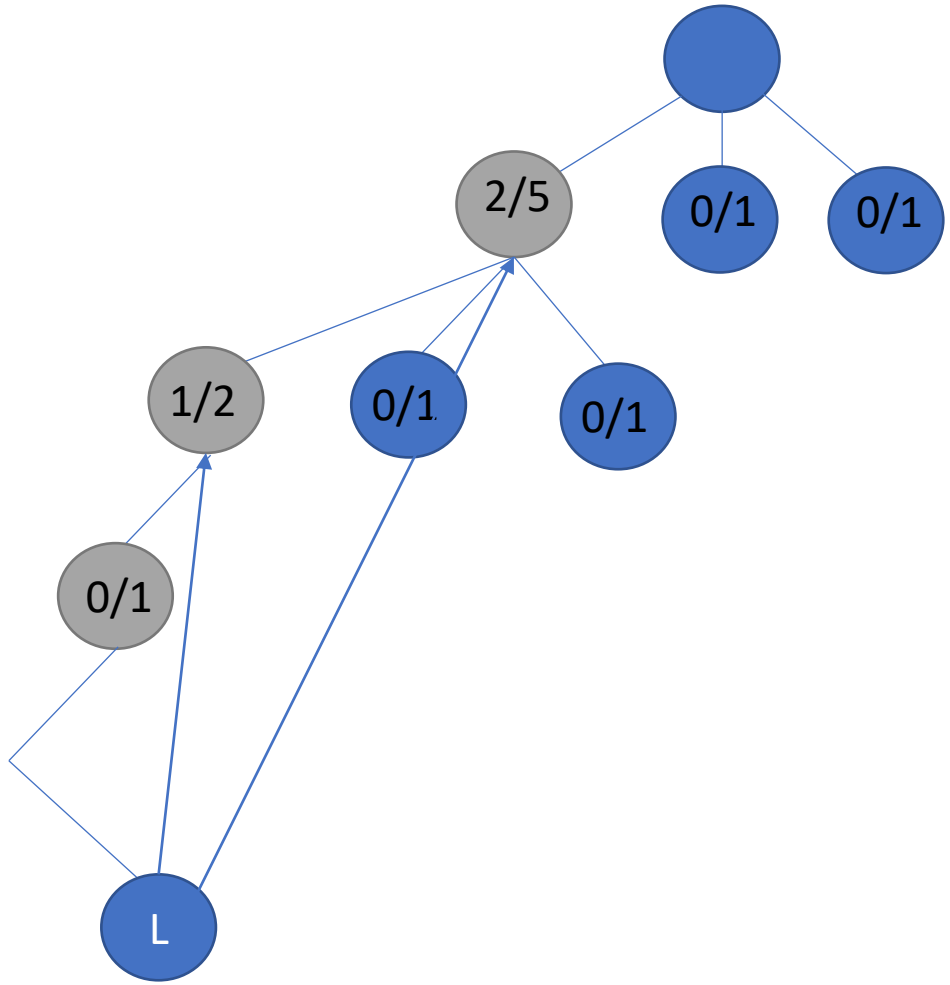
Selection, Expansion, Simulation, Update



Selection

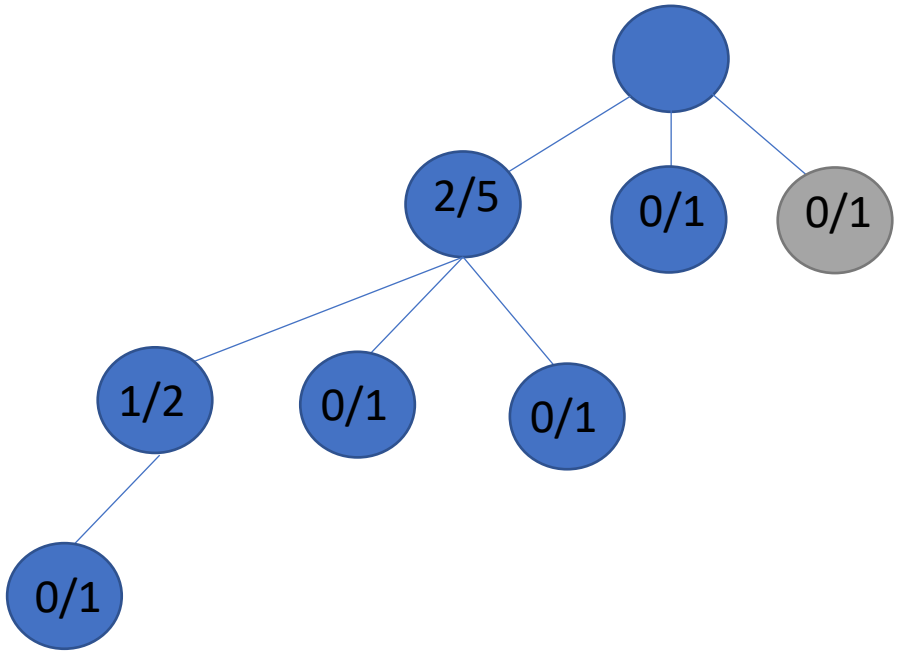
As all the child nodes have been expanded already,
It will run the selection again, this time selecting
The left most node.

Selection, Expansion, Simulation, Update



Expansion, Simulation, Update,

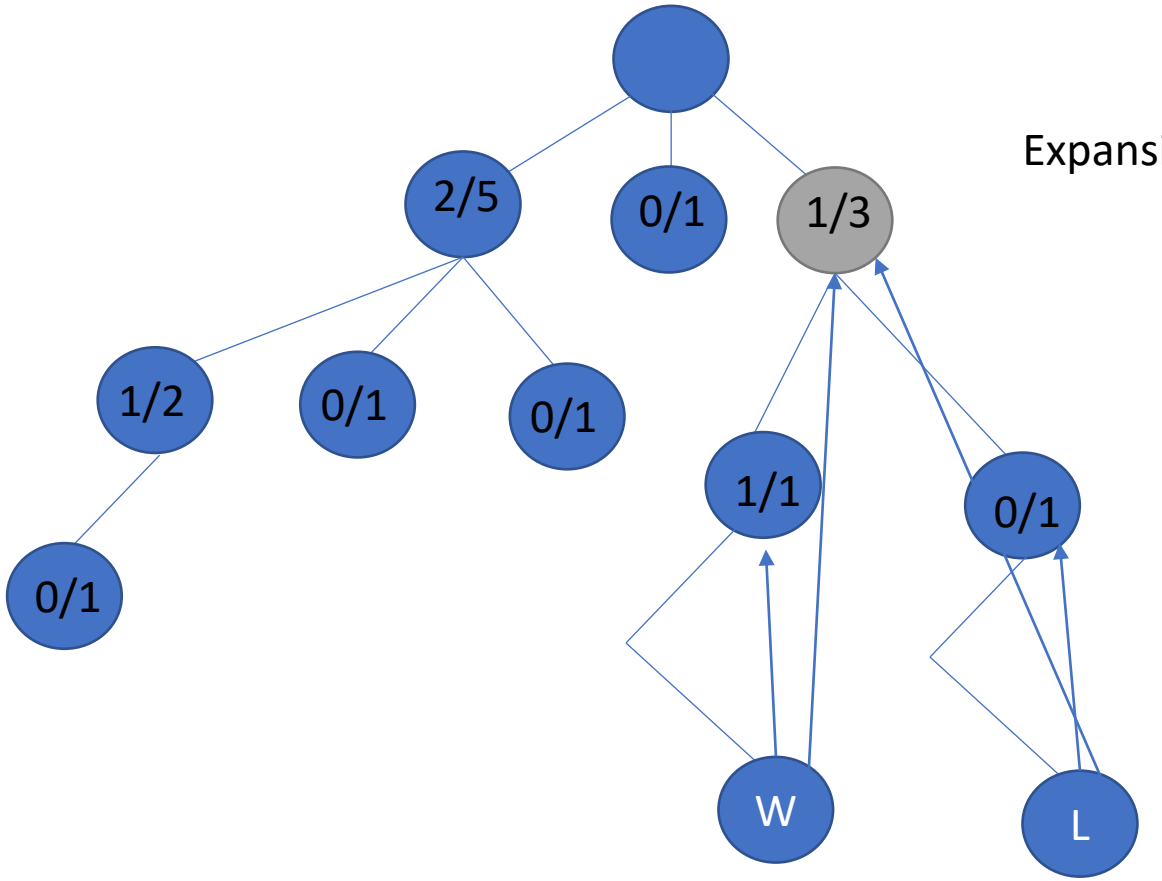
Selection, Expansion, Simulation, Update



Selection

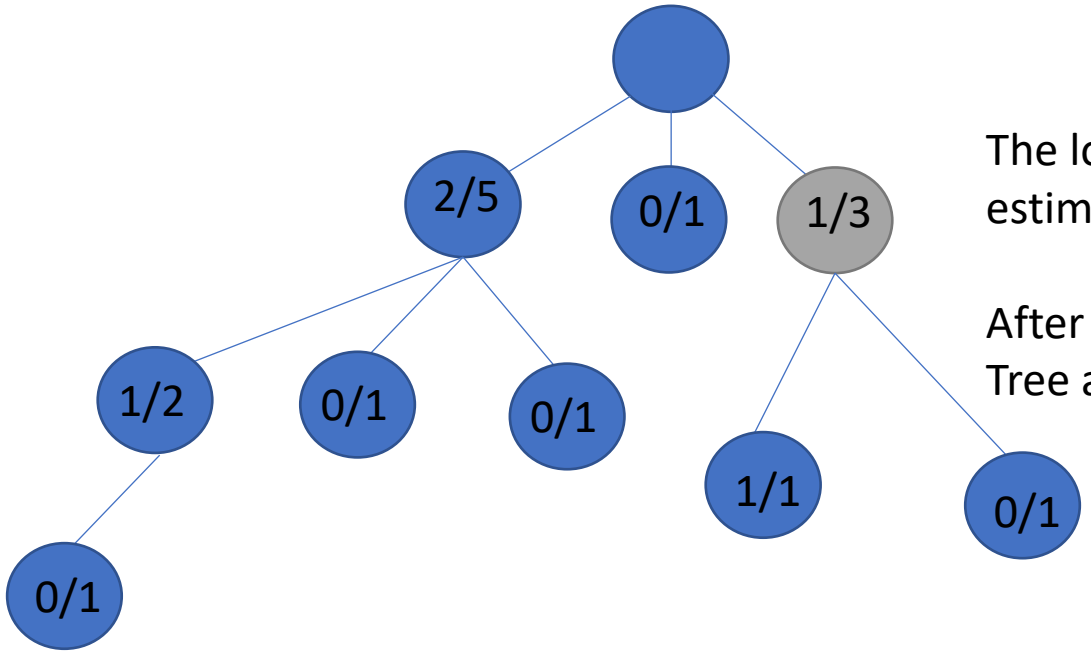
Now the algorithm decides
to explore the ignored nodes

Selection, Expansion, Simulation, Update



Expansion, Simulation, Update

Selection, Expansion, Simulation, Update



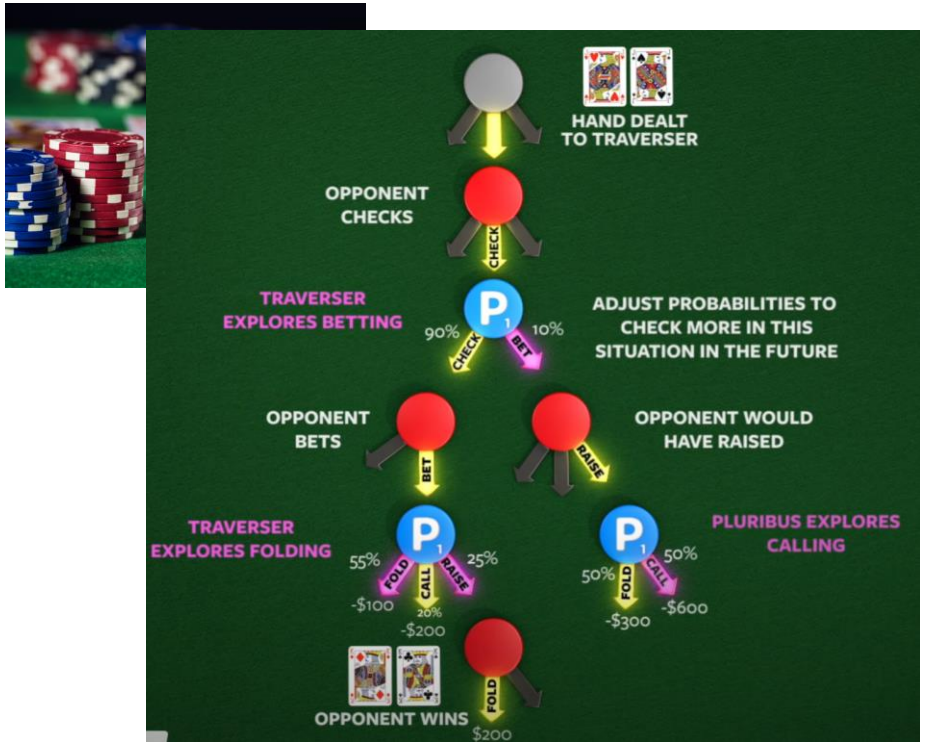
The longer you repeat, the more accurate your estimates become!

After infinite iterations you would obtain a similar Tree as minimax.

If you ask the system now to pick a move.
It would take the left most childnode of the rootnode.
Not because it has the highest chance but because
It was visited most often.

MCTS is domain independent!

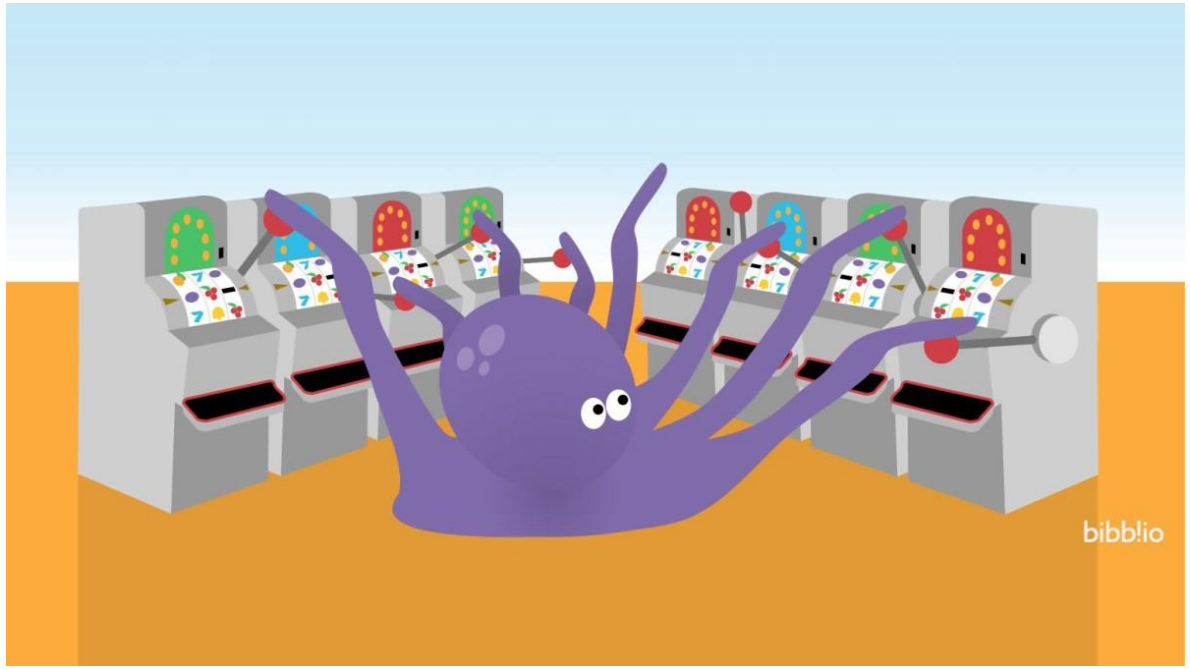
- In addition of being effective in situation with a very high branching factor MCTS is domain independent.



https://www.youtube.com/watch?v=u90TbxK7VEA&ab_channel=TwoMinutePapers

Bandit Based Methods

- Exploration Only
- Exploitation Only
- Epsilon Greedy



- https://www.youtube.com/watch?v=e3L4VocZnnQ&ab_channel=ritvikmath

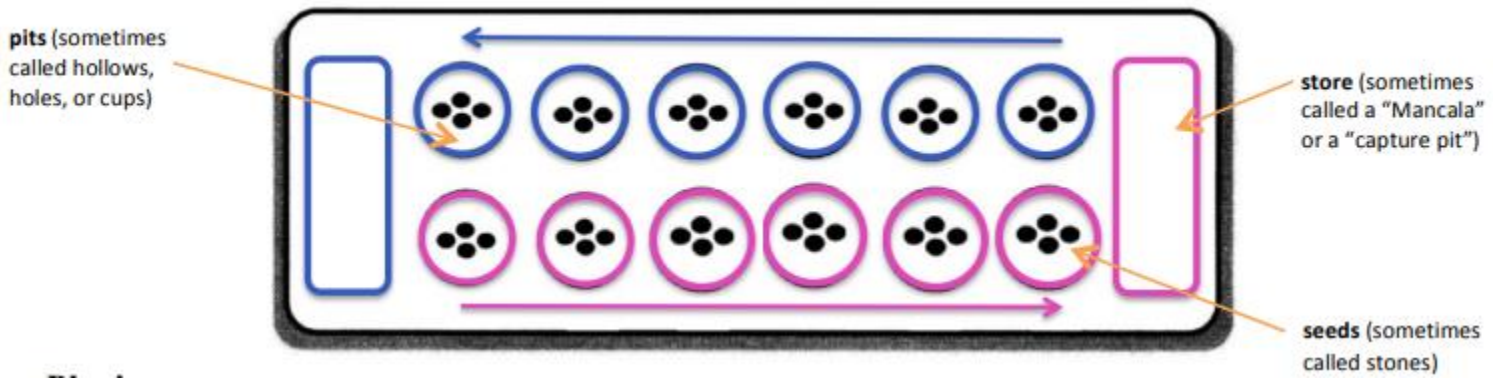
Upper confidence bound, version 1 (UCB 1) Strategy

- When you are uncertain about your estimate of the value of a move.

- At each Time t, Pick Move such that
- Mü_r + sqrt(2ln(t)/N_t(r)) is maximized
- Where mü_r is the expoitation and the rest is the exploration.

- https://www.youtube.com/watch?v=FgmMK6RPU1c&ab_channel=ritvikmath

Lab



Basic Rules:

- Play always moves around the board in a counter-clockwise circle (to the right)
- * The store on your right belongs to you. That is where you keep the seeds you win.
- * The six pits near you are your pits.
- * Only use one hand to pick up and put down seeds.
- * Once you touch the seeds in a pit, you must move those seeds.
- * Only put seeds in your own store, not your opponent's store.

Starting the Game: On a turn, a player picks up all the seeds in one pit and "sows" them to the right, placing one seed in each of the pits along the way. If you come to your store, then add a seed to your store and continue. You may end up putting seeds in your opponent's pits along the way. Play alternates back and forth, with opponents picking up the seeds in one of their pits and distributing them one at a time into the pits on the right, beginning in the pit immediately to the right.

Special Rules:

*When the last seed in your hand lands in your store, take another turn.

*When the last seed in your hand lands in one of your own pits, if that pit had been empty you get to keep all of the seeds in your opponents pit on the opposite side. Put those captured seeds, as well as the last seed that you just played on your side, into the store.

Ending the Game: The game is over when one player's pits are completely empty. The other player takes the seeds remaining in her pits and puts those seeds in her store. Count up the seeds. Whoever has the most seeds wins.

Game of Mancala

- https://github.com/metzzo/mancala

See also:

- Java Developer Newsletter: From your Oracle account, select **Subscriptions**, expand **Technology**, and subscribe to **Java**.
- Java Developer Day hands-on workshops (free) and other events
- Java Magazine

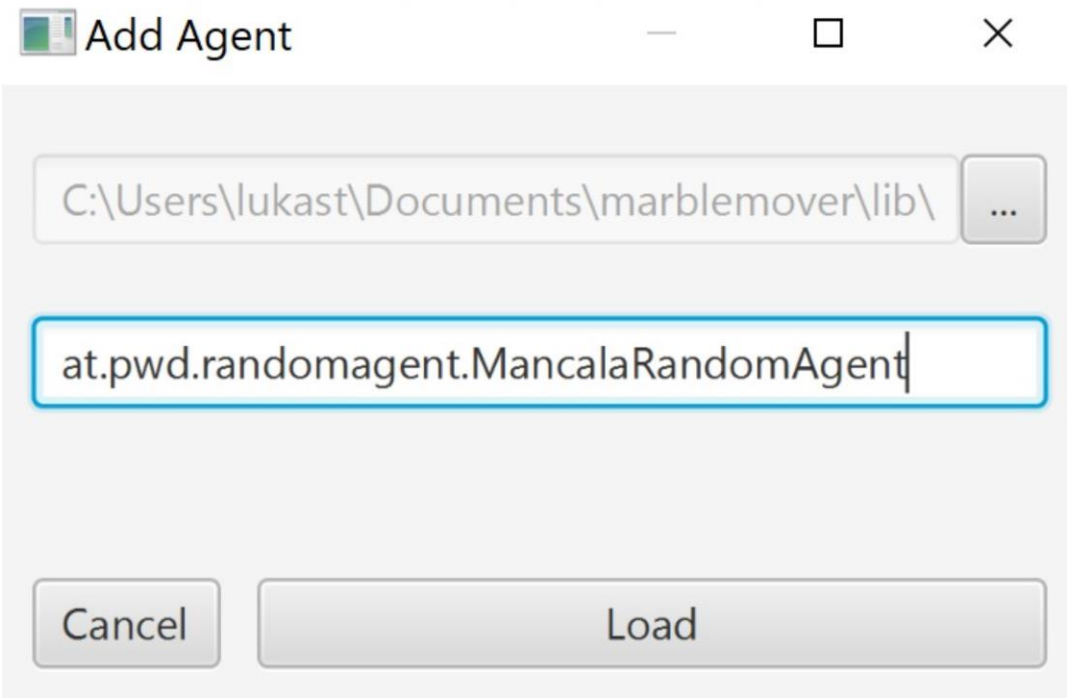
JDK 8u231 checksum

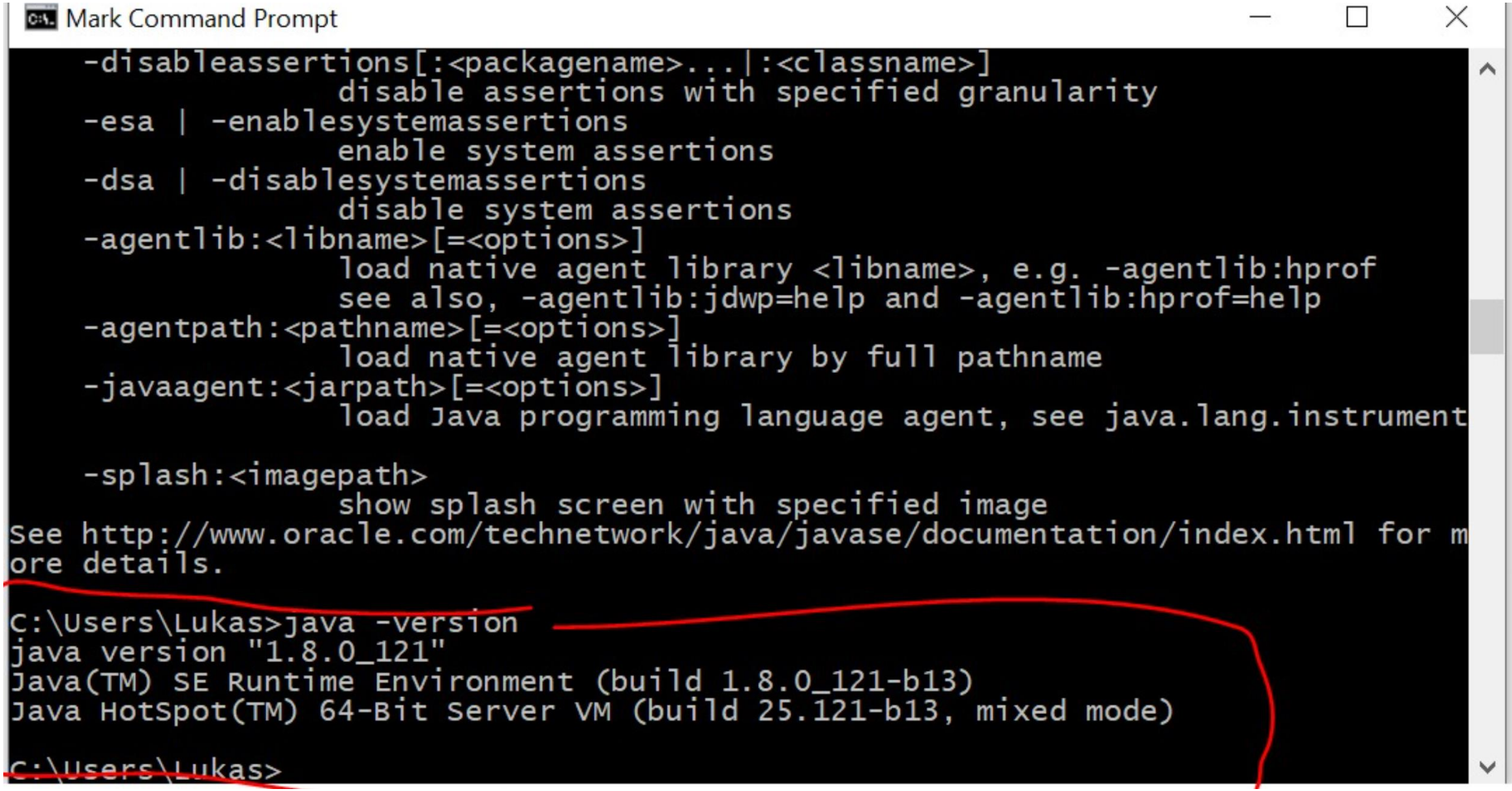
Java SE Development Kit 8u231

You must accept the Oracle Technology Network License Agreement for Oracle Java SE to download this software.

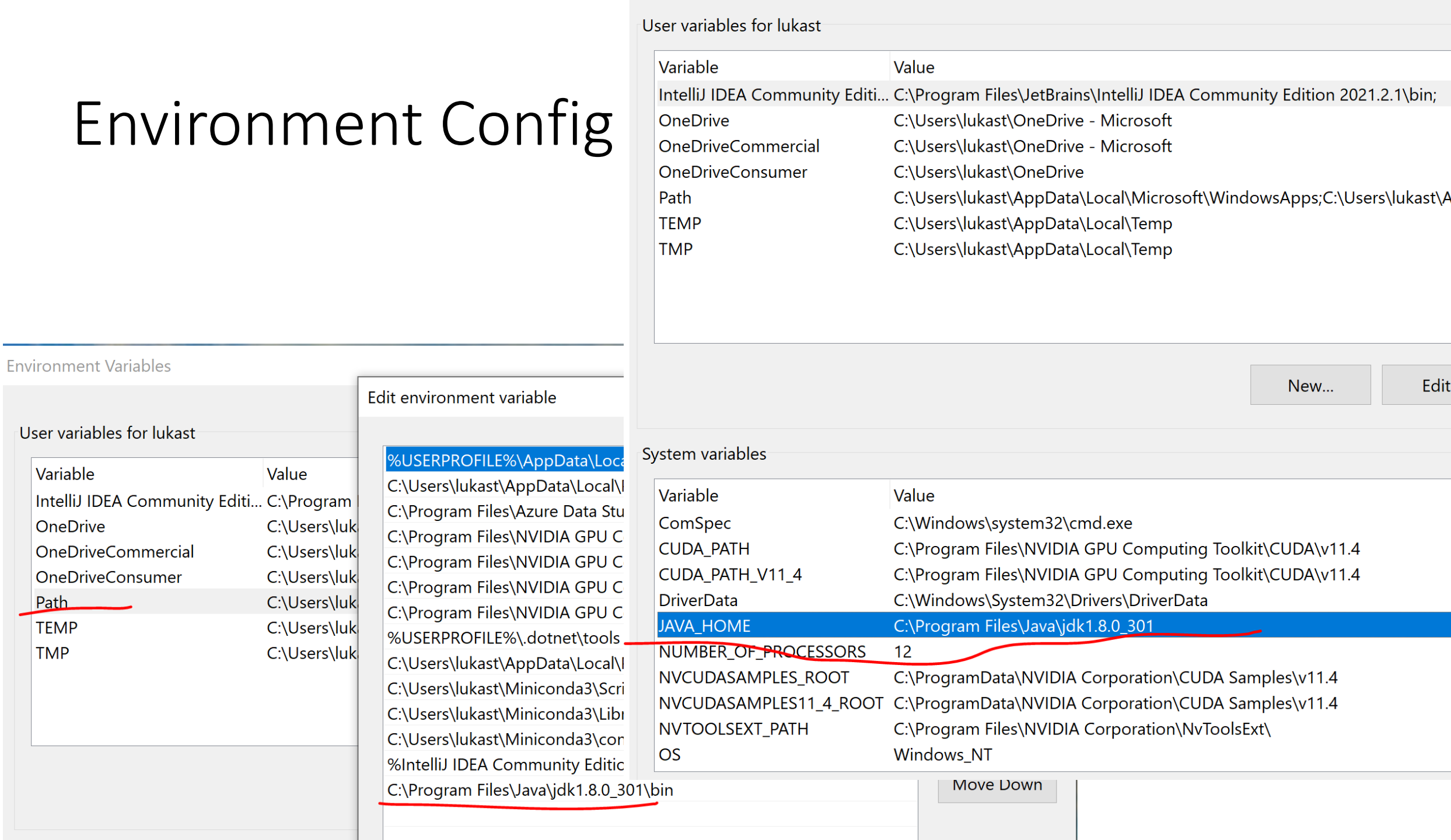
Accept License Agreement **Decline License Agreement**

Product / File Description	File Size	Download
Linux ARM 32 Hard Float ABI	72.9 MB	jdk-8u231-linux-arm32-vfp-hflt.tar.gz
Linux ARM 64 Hard Float ABI	69.8 MB	jdk-8u231-linux-arm64-vfp-hflt.tar.gz
Linux x86	170.93 MB	jdk-8u231-linux-i586.rpm
Linux x86	185.75 MB	jdk-8u231-linux-i586.tar.gz
Linux x64	170.32 MB	jdk-8u231-linux-x64.rpm
Linux x64	185.16 MB	jdk-8u231-linux-x64.tar.gz
Mac OS X x64	253.4 MB	jdk-8u231-macosx-x64.dmg
Solaris SPARC 64-bit (SVR4 package)	132.98 MB	jdk-8u231-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	94.16 MB	jdk-8u231-solaris-sparcv9.tar.gz
Solaris x64 (SVR4 package)	133.73 MB	jdk-8u231-solaris-x64.tar.Z
Solaris x64	91.96 MB	jdk-8u231-solaris-x64.tar.gz
Windows x86	200.22 MB	jdk-8u231-windows-i586.exe
Windows x64	210.18 MB	jdk-8u231-windows-x64.exe

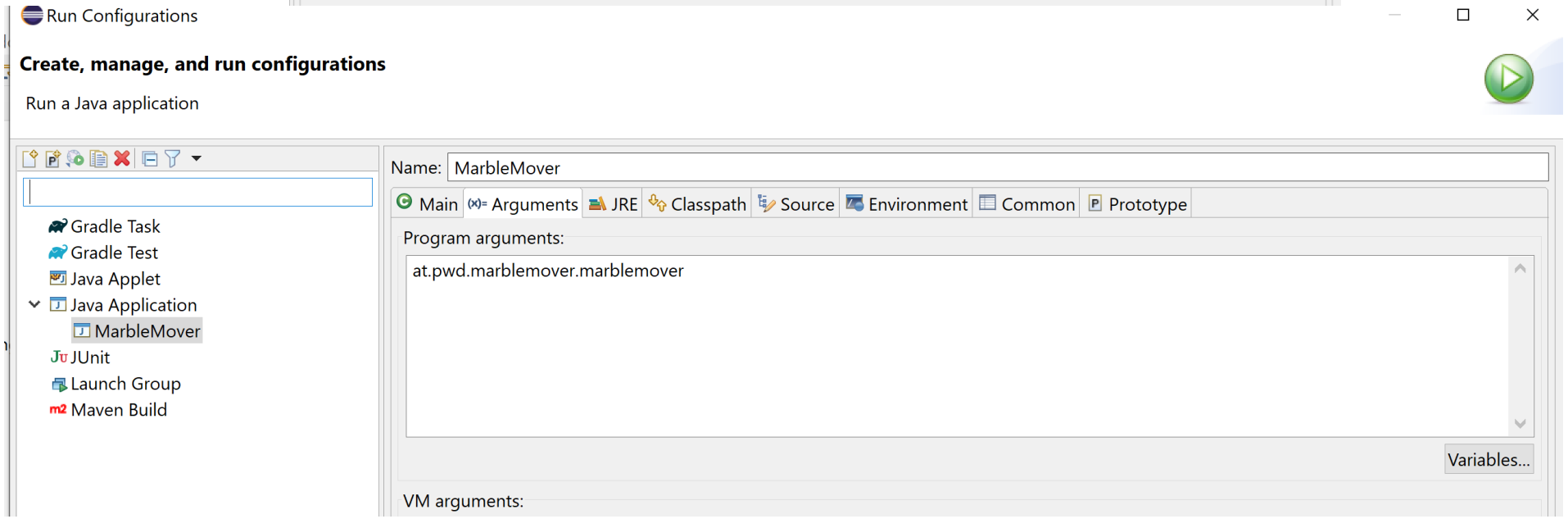



Mark Command Prompt
-disableassertions[:<packagename>...|:<classname>]
disable assertions with specified granularity
-esa | -enablesystemassertions
enable system assertions
-dsa | -disablesystemassertions
disable system assertions
-agentlib:<libname>[=<options>]
load native agent library <libname>, e.g. -agentlib:hprof
see also, -agentlib:jdwp=help and -agentlib:hprof=help
-agentpath:<pathname>[=<options>]
load native agent library by full pathname
-javaagent:<jarpath>[=<options>]
load Java programming language agent, see java.lang.instrument
-splash:<imagepath>
show splash screen with specified image
See http://www.oracle.com/technetwork/java/javase/documentation/index.html for m
ore details.
C:\Users\Lukas>java -version
java version "1.8.0_121"
Java(TM) SE Runtime Environment (build 1.8.0_121-b13)
Java HotSpot(TM) 64-Bit Server VM (build 25.121-b13, mixed mode)
C:\Users\Lukas>

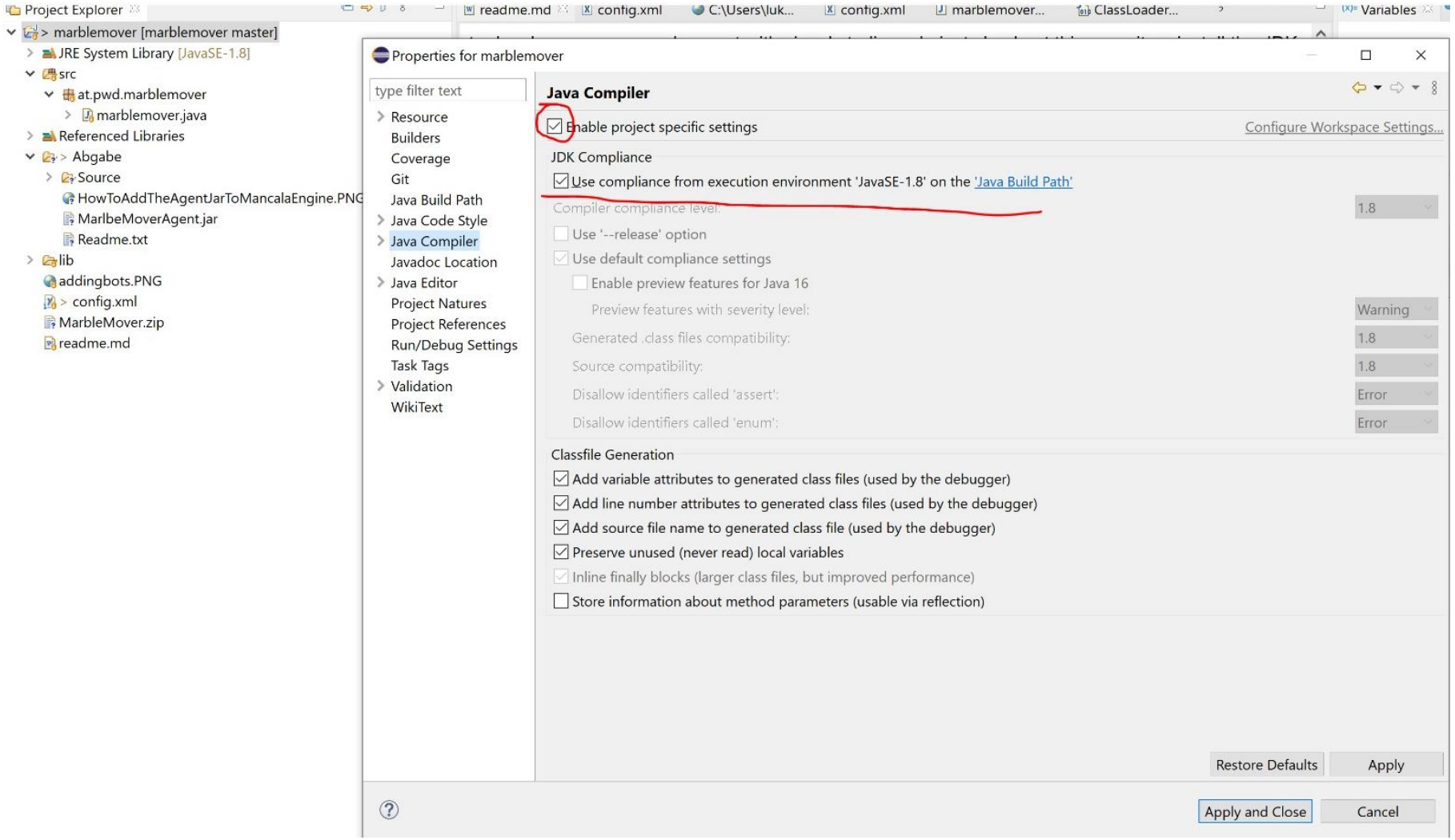
Environment Config



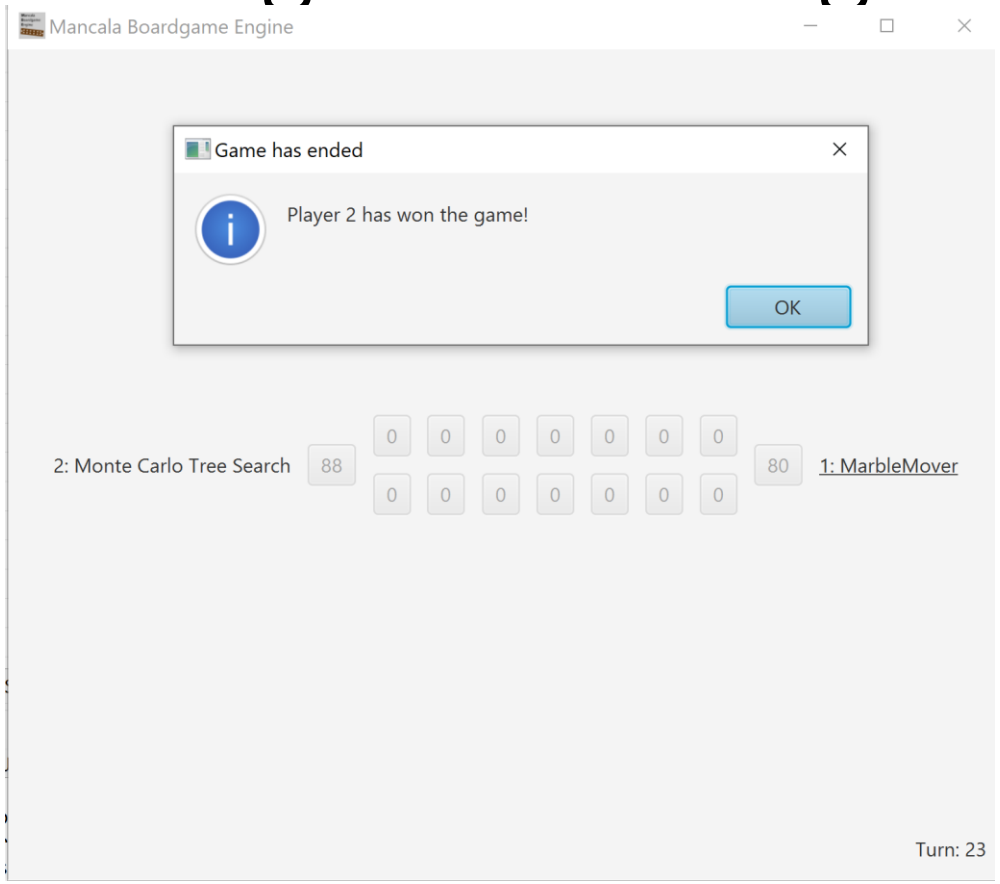
Eclipse Run Configuration

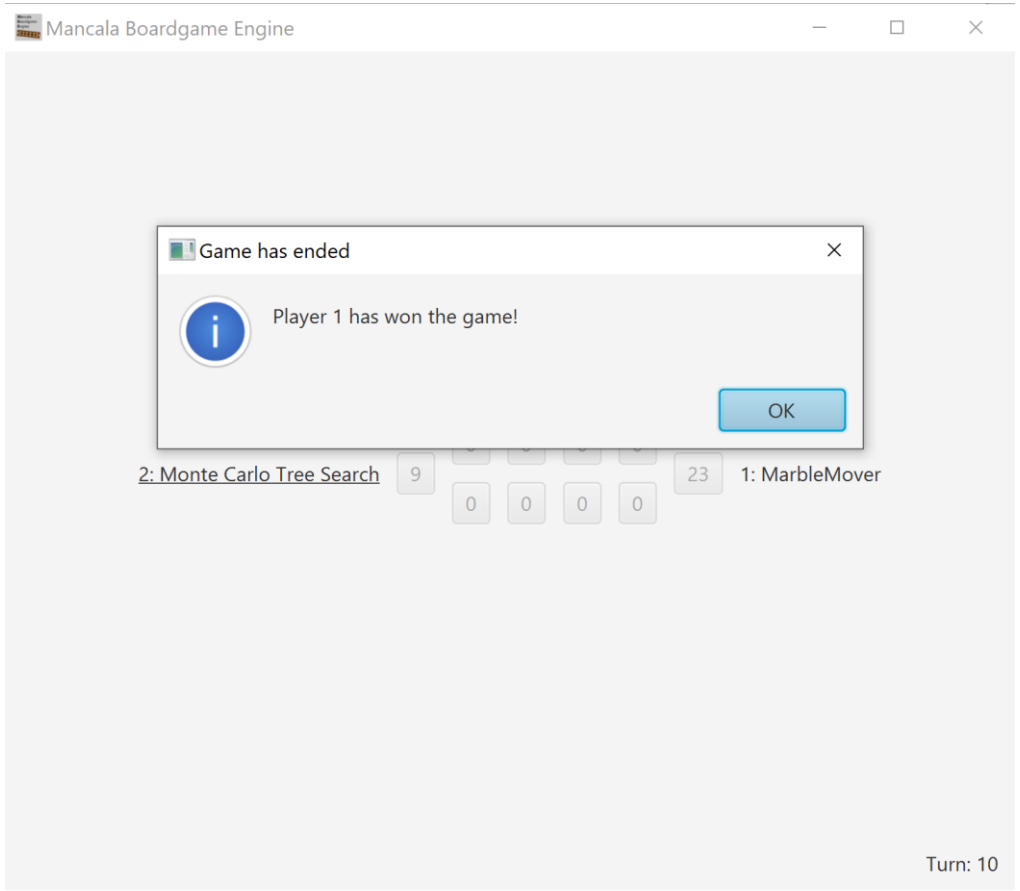


Change Compiler settings

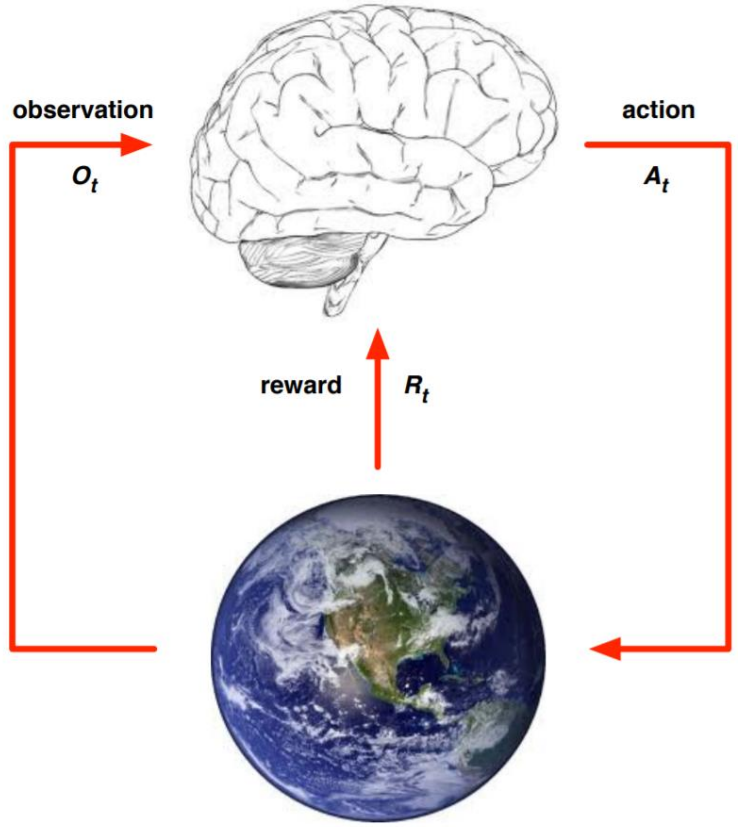


Example Play – MCTS outperforms Minimax on high branching factor





A brief introduction to Reinforcement Learning



https://www.davidsilver.uk/wp-content/uploads/2020/03/intro_RL.pdf

Comparison to supervised learning:
No Ground Truth Labels to learn from
Instead you have an Environment as “Teacher”

Key idea, start by doing random actions.
At one point (early or late) you will receive a reward or penalty

Reinforce the actions that lead to higher reward.

Game Environments: Gym

- Open AI Framework
  - Atari Games
  - Classic Control problems -> good for testing
  - Board Games
  - …
- Issues: environments may work on windows or not.
- Even Cause Bluescreens
- Demo!

Example of a RL Algorithm Cross Entropy Method

1. Play N episodes using the current model and environment
2. Calculate reward for every episode
3. Keep only the top 70% of the episodes by reward
4. Train neural net on these “elite” episodes taking state as input and action as output.
5. Repeat till convergence

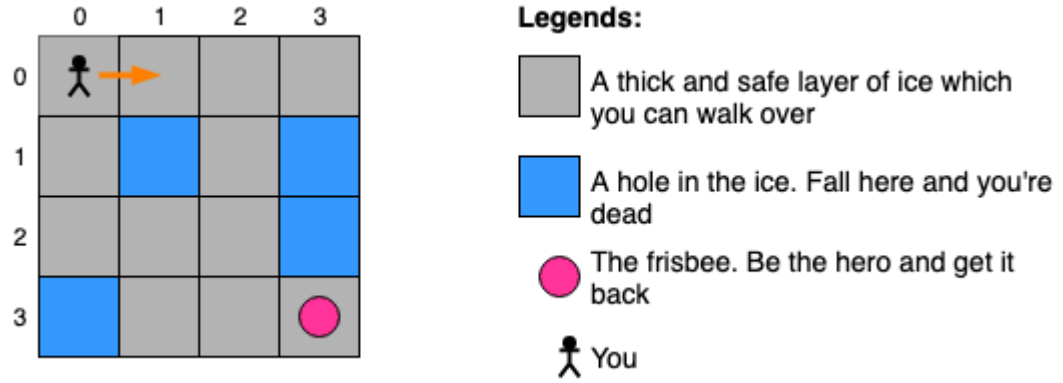
Categorizing RL agents (1)

- Value Based
  - No Policy (Implicit)
  - Value Function
- Policy Based
  - Policy
  - No Value Function
- Actor Critic
  - Policy
  - Value Function

- Policy Based -> Predicting a probability distribution over the next actions
- Value Based -> Predicting the value of beeing in the next state

Value Based Method

- For each next state estimate the discounted reward.
- Easy when number of states is low and your world is deterministic.

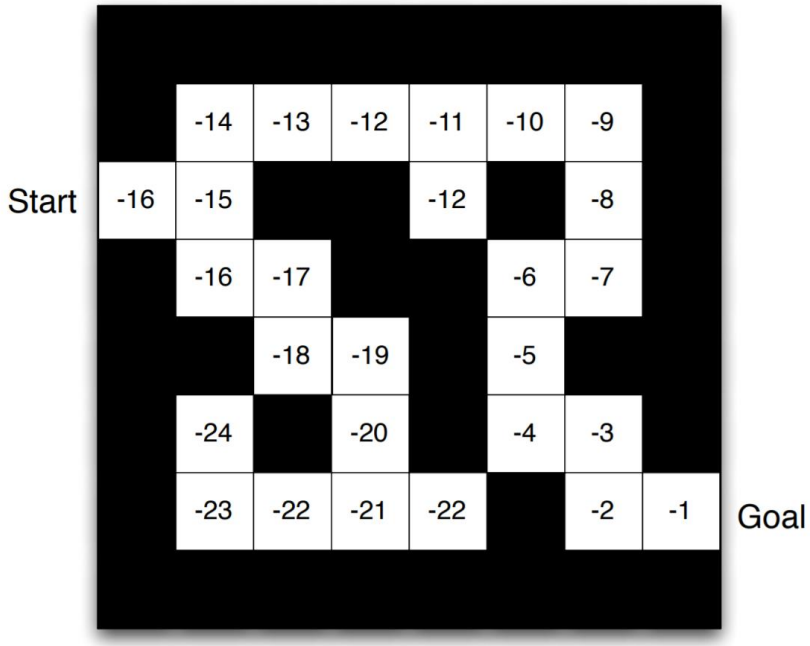


- Imagine you have all Q(s,a) -Values in a table

then you can take the action with highest Q-Value

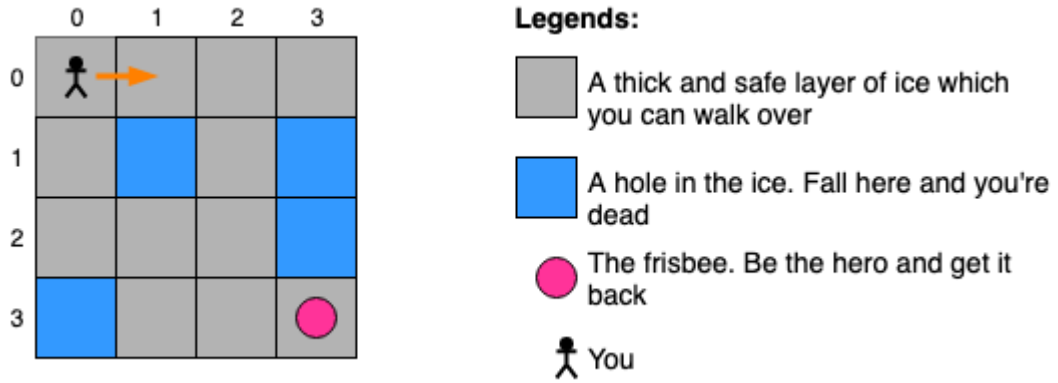
Example of a deterministic environment with a value for every position:

Maze Example: Value Function



- Numbers represent value $v_\pi(s)$ of each state s

Example of a stochastic environment:



https://reinforcement-learning4.fun/2019/06/09/introduction-reinforcement-learning-frozen-lake-example/

Bellman Equation

The Value of state x
is the maximum of the sum of
immediate reward and the discounted next states Value. Only the one action where the value is maximized is selected.

Recursive Definition!
Often you get the reward only at the end of the game.

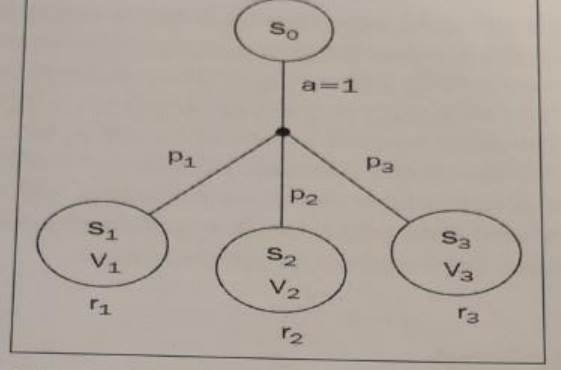


Figure 4: An example of the transition from the state in a stochastic case

Here we have one action, which can lead to three different states with different probabilities: with probability p_1 it can end up in state s_1, p_2 in state s_2, and p_3 in state s_3 ($p_1 + p_2 + p_3 = 1$, of course). Every target state has its own reward (r_1,r_2, or r_3). To calculate the expected value after issuing action 1, we need to sum all values, multiplied by their probabilities:

$$V_0(a = 1) = p_1(r_1 + \gamma V_1) + p_2(r_2 + \gamma V_2) + p_3(r_3 + \gamma V_3)$$

or more formally,

$$V_0(a) = \mathbb{E}_{s \sim S}[r_{s,a} + \gamma V_s] = \sum_{s \in S} p_{a,0 \to s}(r_{s,a} + \gamma V_s)$$

By combining the Bellman equation, for a deterministic case, with a value for stochastic actions, we get the Bellman optimality equation for a general case:

$$V_0 = \max_{a \in A} \mathbb{E}_{s \sim S}[r_{s,a} + \gamma V_s] = \max_{a \in A} \sum_{s \in S} p_{a,0 \to s}(r_{s,a} + \gamma V_s)$$

(Note that $p_{a,i \to j}$ means the probability of action a, issued in state i, to end up in state j.)

The interpretation is still the same: the optimal value of the state is equal to the action, which gives us the maximum possible expected immediate reward, plus discounted long-term reward for the next state. You may also notice that this definition is recursive: the value of the state is defined via the values of immediate reachable states.

Source: Maxim Lapan, Deep Reinforcement Learning Hands-On

Example of a value based Method DQN

- Simple Demo running in local rl Conda Environment

Bellman Equation visible in the code:

target = (reward + self.gamma * np.amax(self.model.predict(next_state)[0]))

Policy-Based Reinforcement Learning

- In the last lecture we approximated the value or action-value function using parameters θ,

$$V_\theta(s) \approx V^\pi(s)$$
$$Q_\theta(s,a) \approx Q^\pi(s,a)$$

- A policy was generated directly from the value function
  - e.g. using ϵ-greedy
- In this lecture we will directly parametrise the policy

$$\pi_\theta(s,a) = \mathbb{P}\left[a \mid s, \theta\right]$$

- We will focus again on model-free reinforcement learning

Advantages of Policy-Based RL

Advantages:

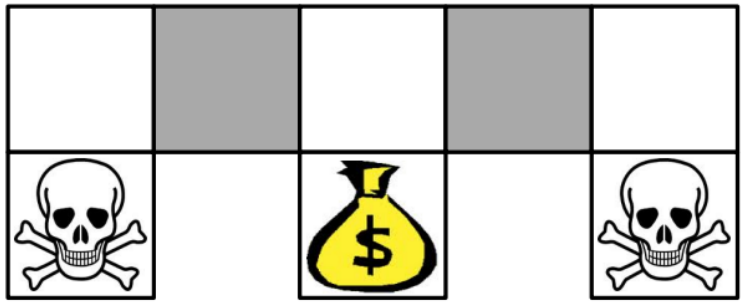
- Better convergence properties
- Effective in high-dimensional or continuous action spaces
- Can learn stochastic policies

Disadvantages:

- Typically converge to a local rather than global optimum
- Evaluating a policy is typically inefficient and high variance

- Two-player game of rock-paper-scissors
  - Scissors beats paper
  - Rock beats scissors
  - Paper beats rock
- Consider policies for *iterated* rock-paper-scissors
  - A deterministic policy is easily exploited
  - A uniform random policy is optimal (i.e. Nash equilibrium)

Example: Aliased Gridworld (1)



- The agent cannot differentiate the grey states
- Consider features of the following form (for all N, E, S, W)

$$\phi(s, a) = \mathbf{1}(\text{wall to N}, a = \text{move E})$$

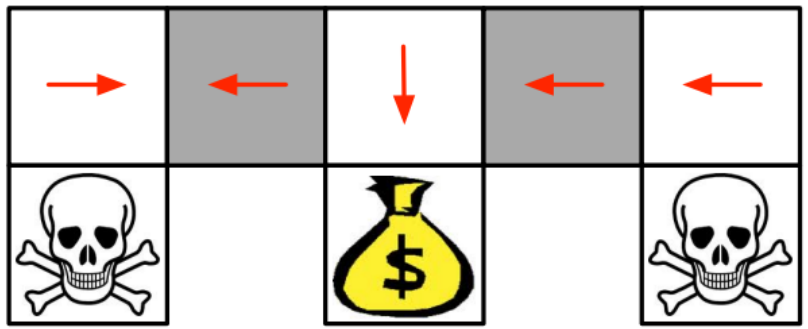
- Compare value-based RL, using an approximate value function

$$Q_\theta(s, a) = f(\phi(s, a), \theta)$$

- To policy-based RL, using a parametrised policy

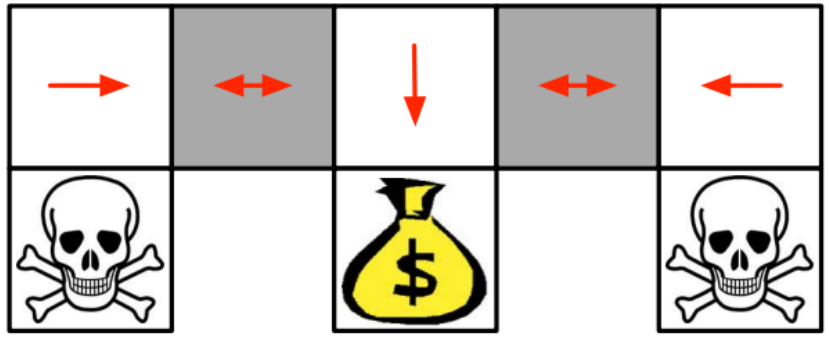
$$\pi_\theta(s, a) = g(\phi(s, a), \theta)$$

Example: Aliased Gridworld (2)



- Under aliasing, an optimal deterministic policy will either
  - move W in both grey states (shown by red arrows)
  - move E in both grey states
- Either way, it can get stuck and *never* reach the money
- Value-based RL learns a near-deterministic policy
  - e.g. greedy or ϵ-greedy
- So it will traverse the corridor for a long time

Example: Aliased Gridworld (3)



- An optimal stochastic policy will randomly move E or W in grey states

$$\pi_\theta(\text{wall to N and S, move E}) = 0.5$$
$$\pi_\theta(\text{wall to N and S, move W}) = 0.5$$

- It will reach the goal state in a few steps with high probability
- Policy-based RL can learn the optimal stochastic policy

Example of a Policy Based Method REINFORCE

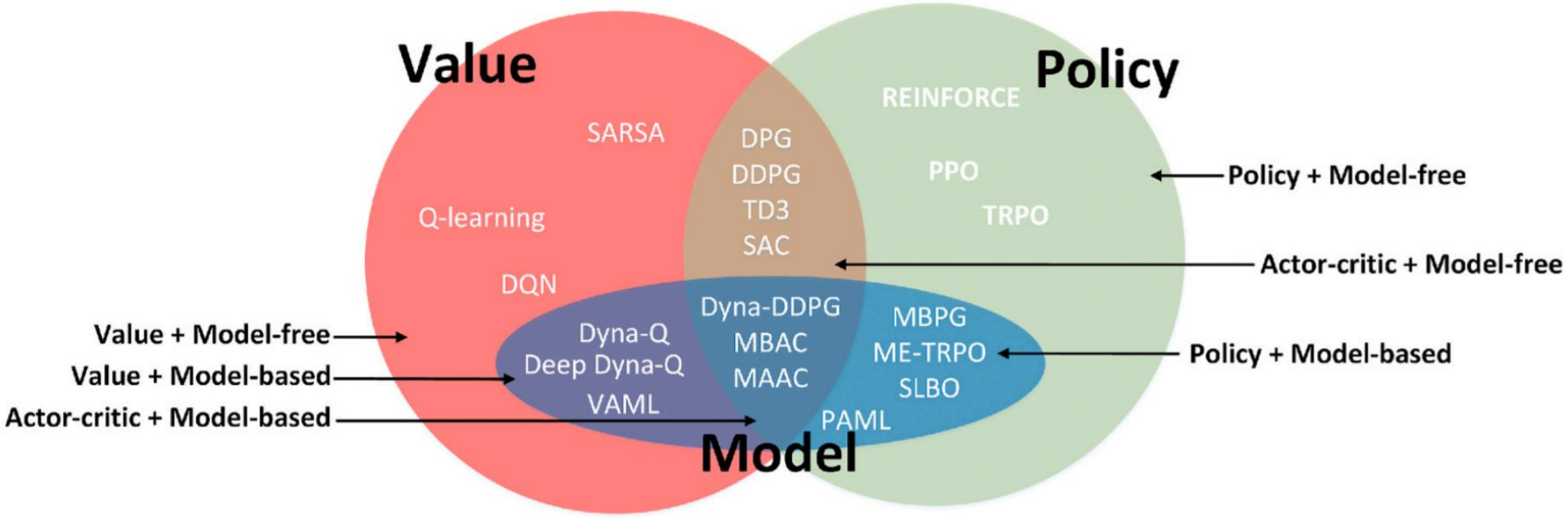
function REINFORCE
Initialise θ arbitrarily
for each episode $\{s_1, a_1, r_2, ..., s_{T-1}, a_{T-1}, r_T\} \sim \pi_\theta$ **do**
for $t = 1$ to $T - 1$ **do**
$\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t$
end for
end for
return θ
end function

- https://medium.com/@thechrisyoon/deriving-policy-gradients-and-implementing-reinforce-f887949bd63
- Demo:

https://cs.stanford.edu/people/karpathy/reinforcejs/gridworld_dp.html

Reinforcement Learning

- Policy based is better in case of broken Markov property.
- Value based is often more sample efficient (not necessarily more computationally efficient)



https://www.davidsilver.uk/wp-content/uploads/2020/03/pg.pdf

https://www.youtube.com/watch?v=KHZVXao4qXs&ab_channel=DeepMind

Development Environment Options

Directly on your Windows Desktop Client

Windows /Linux Dual Boot
Linux Subsystem -> just no

Docker

Azure Data Science VM

Azure ML + Ray RLLib

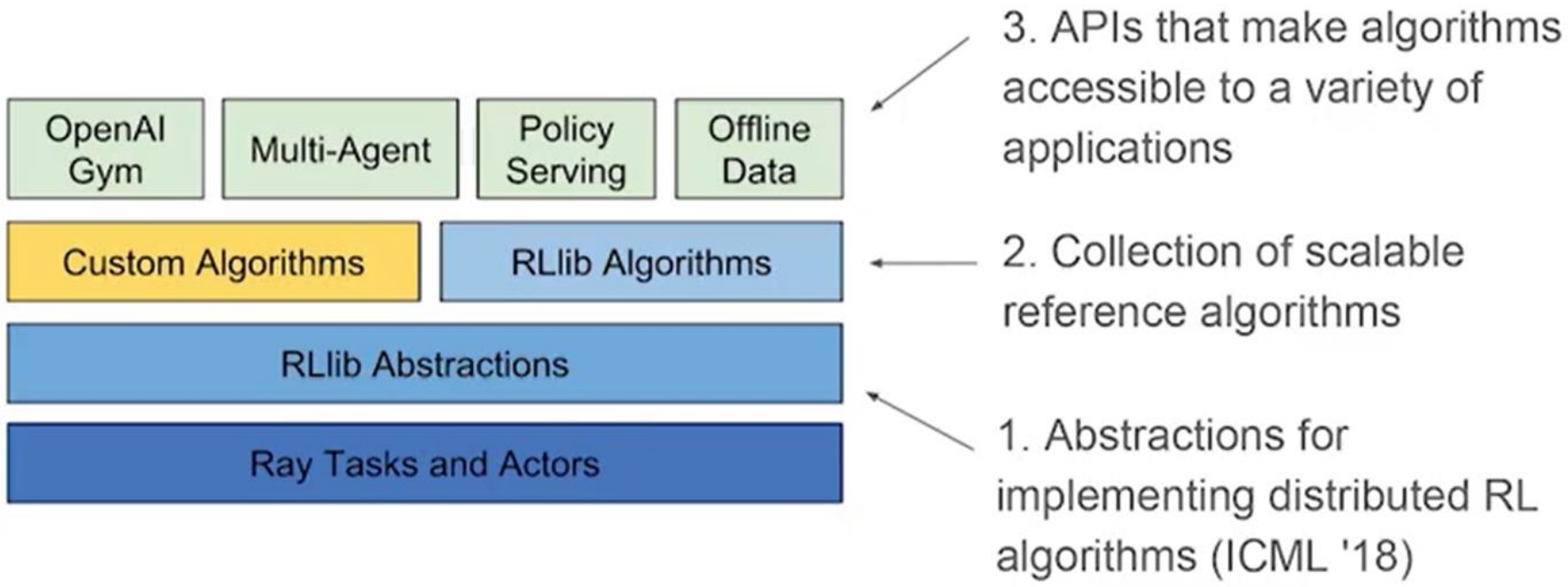
Azure ML Reinforcement Learning

- Managed RL service for large scale distributed simulation and training, using Ray/RLlib framework
- Customers create compute clusters and submit simulation/training jobs using standard Azure ML pattern with SDK & CLI
- RL algorithms are in RLlib – Deep training is Tensorflow by default, Pytorch possible

RAY

RLlib: A Unified Library for Reinforcement Learning

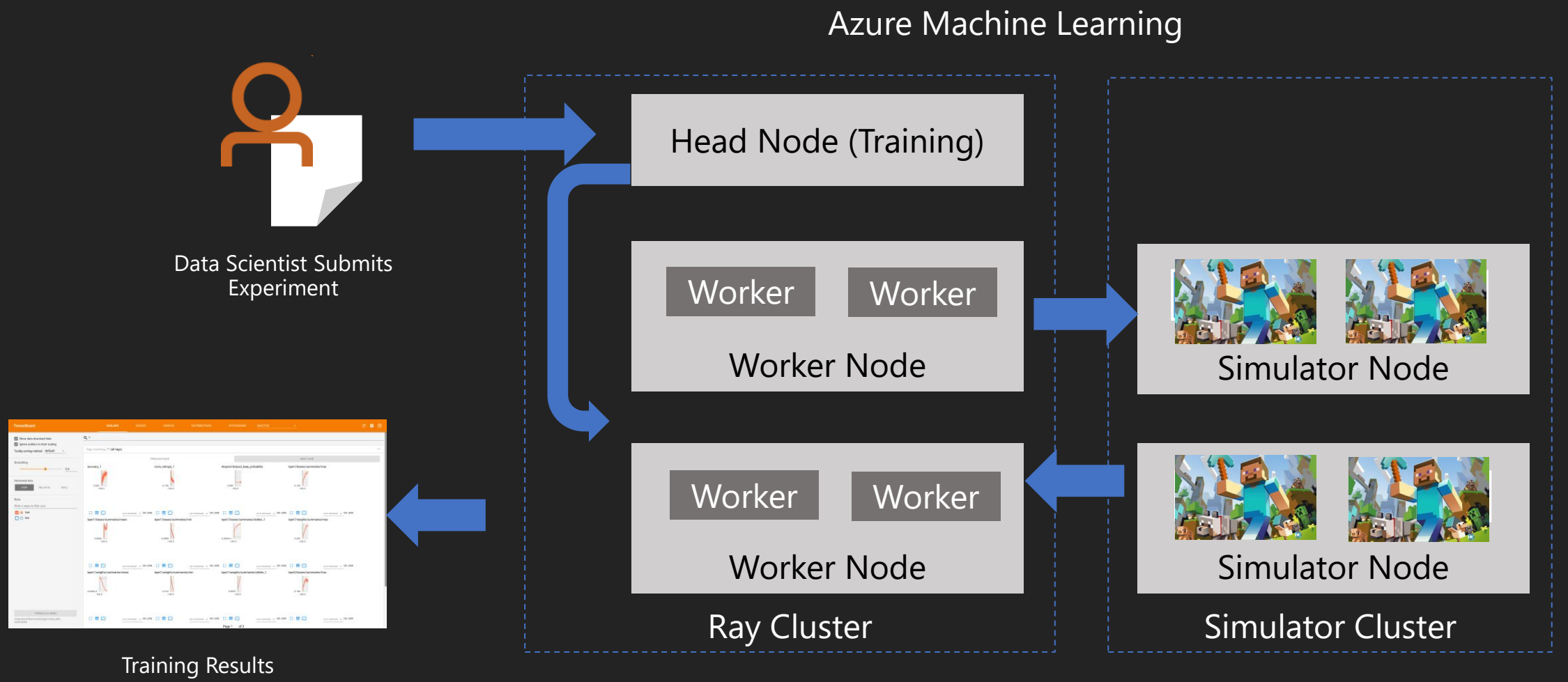
Three main value adds:



Custom Environments [RLlib Environments — Ray v2.0.0.dev0](#) , Example with external Game Engine: [ray/unity3d_server.py at master · ray-project/ray (github.com)](#)

Available Algorithms: [RLlib Algorithms — Ray v1.6.0](#)
https://www.youtube.com/watch?v=8tG8PJC6oaU&ab_channel=AICamp

How It Works


Azure Machine Learning
Data Scientist Submits Experiment
Head Node (Training)
Worker
Worker
Worker Node
Worker
Worker
Worker Node
Ray Cluster
Simulator Node
Simulator Node
Simulator Cluster
Training Results

Requirements

- 100’s of parallel simulations
- Training: multiple days
- Multiple Ray jobs
- Resilient to simulator / worker failures
- ML Ops pipeline integration

Simulators

- Open AI Gym
- Custom simulators with Open AI Gym Environment interface – worker local or remote in simulator
- Windows support
- Investigating additional simulator support

Simple Demo

Shows the simple cartpole_sc.ipynb

Shows compute cluster and how to look at the logs & output

Advanced Demo

Shows the atari-on-distributed compute.ipynb

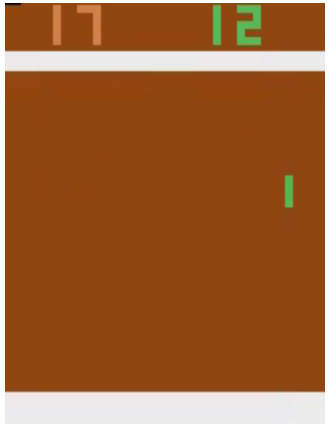
Shows compute cluster and how to look at the logs & output

Issues with Cartpole

- For the cartpole rl on sc demo I had some missing python packages in the docker image (resulting in an error when trying to capture the video) -> fixed this in a fork of the repo and created pull request
- Missing Azure CLI for the Atari Distributed pong Demo

Install the Azure CLI for Windows | Microsoft Docs

Issues with Pong



- File "/opt/miniconda/lib/python3.7/site-packages/tensorflow/python/client/session.py", line 1149, in _run
- str(subfeed_t.get_shape())))
- ValueError: Cannot feed value of shape (4, 4, 4, 16) for Tensor 'default_policy/Placeholder_default_policy/conv1/kernel:0', which has shape '(8, 8, 4, 16)'
- 2021-08-25 15:51:15.470861: CPU Usage: 24.1%, Memory Usage: 6.0%, Memory Total/Available: 59074052096/55516184576
- [2m[36m(pid=147)[0m /opt/miniconda/lib/python3.7/site-packages/tensorboard/compat/tensorflow_stub/dtypes.py:541: FutureWar

If time permits

- Alpha Go
- Alpha Go Zero
- Alpha Star
- Mü Zero

AlphaGo

- Policy Network to predict potential moves (Reducing the breadth of the search tree)
- Value Network to predict the outcome of the game (reducing the depth of the search tree)
- https://www.youtube.com/watch?v=x5Q79XCxMVc&ab_channel=LondonMachineLearningMeetup

AlphaGo Zero

- Essentialy Montecarlo Tree search with selfplay.

- Only One Network trained using self play
- No Random MCTS Rollouts (Problems occur if random rollouts often lead to wins by accident you start to believe certain positions are very effective)

- Single Network guides the MCTS Algorithm, suggesting moves -> then play that move run another search(using the model and so on... Till game is completed.
- Then they trained a policy network P‘ to predict what the MCTS would have produced.
- Also trained a new value network V‘ that predicts the winner based on the board

- Search based policy iteration (policy improvement , policy valuation) Alternate between the stages gives you stronger and stronger policies.
- Critical Success Factor: Search based policy evaluation! (testing a new policy by seeing what the value is when playing it to the end)
- Deepmind AlphaZero - Mastering Games Without Human Knowledge - YouTube

Alpha Zero

- Works similar to the human way of playing … looking at promising moves (simulating the next steps) ignoring obviously bad moves.
- Value Network -> Takes Game State as Input -> outputs value of that state. Very simple function approximation from sequences.

- Policy Network -> trained such that the output is the same as MCTS
- MCTS uses Policy Network to determine which positions are promissing

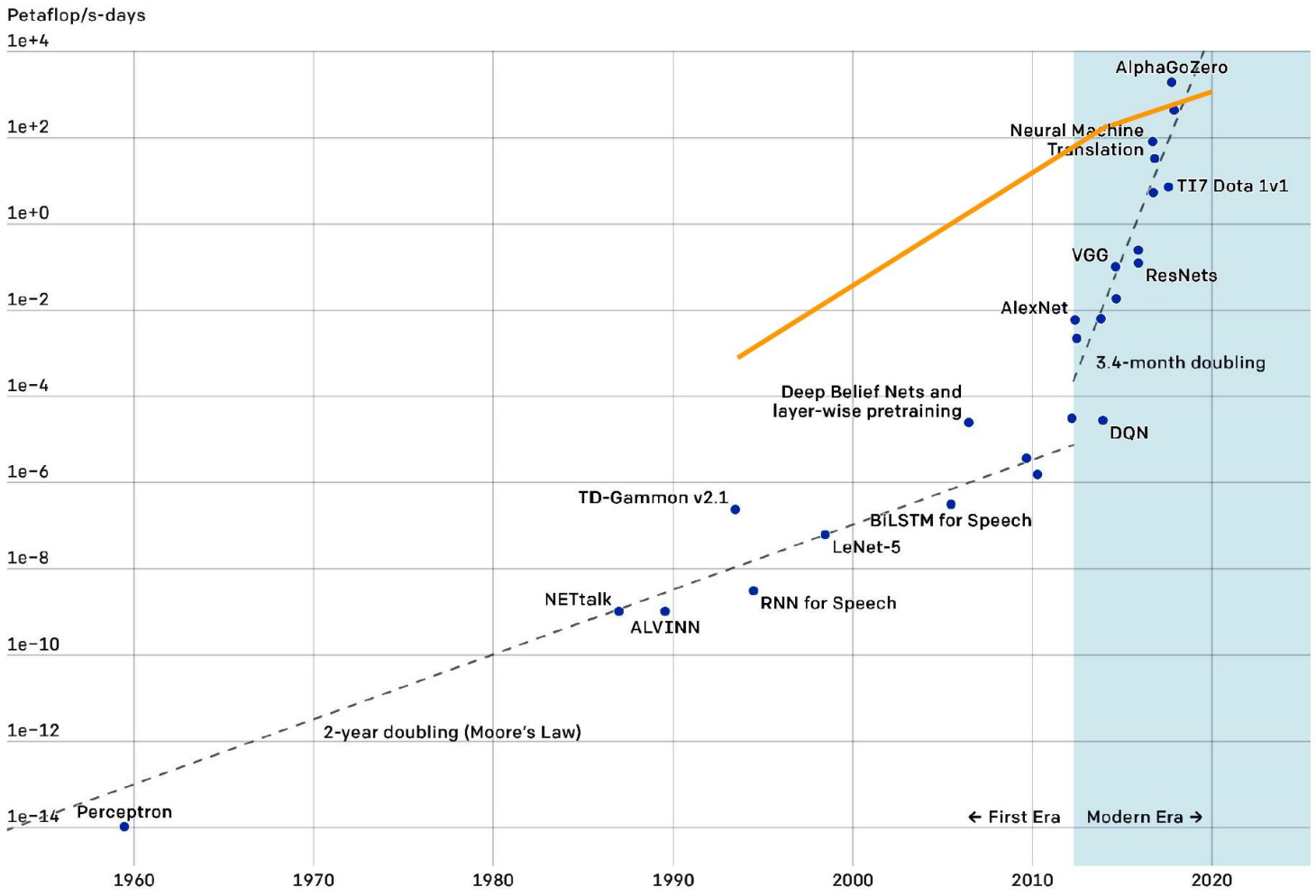
Alpha Star

- https://www.youtube.com/watch?v=DMXvkbAtHNY&ab_channel=TwoMinutePapers

MuZero

Translates the pixel space into an abstract space used for planning,
Dynamics model to predict next state

- https://www.youtube.com/watch?v=L0A86LmH7Yw&ab_channel=JulianSchrittwieser



All supercomputers on the top-500 list running for 1 day: 1600 Petaflop-days.

Image: OpenAI

Appendix

- https://techcommunity.microsoft.com/t5/azure-ai/introducing-reinforcement-learning-on-azure-machine-learning/ba-p/1403028
- https://github.com/Azure/MachineLearningNotebooks/tree/master/how-to-use-azureml/reinforcement-learning
- https://github.com/ray-project/ray
- http://karpathy.github.io/2016/05/31/rl/
- https://cs.stanford.edu/people/karpathy/reinforcejs/gridworld_dp.html

Tutorials

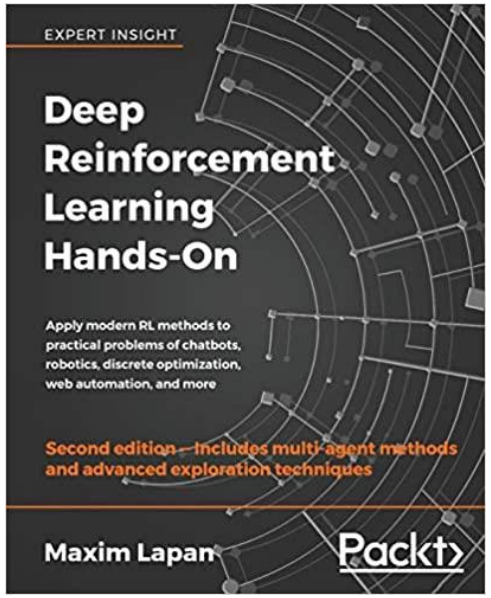
- O’Reilly AI tutorial: Scalable AI and reinforcement learning with Ray

Literature

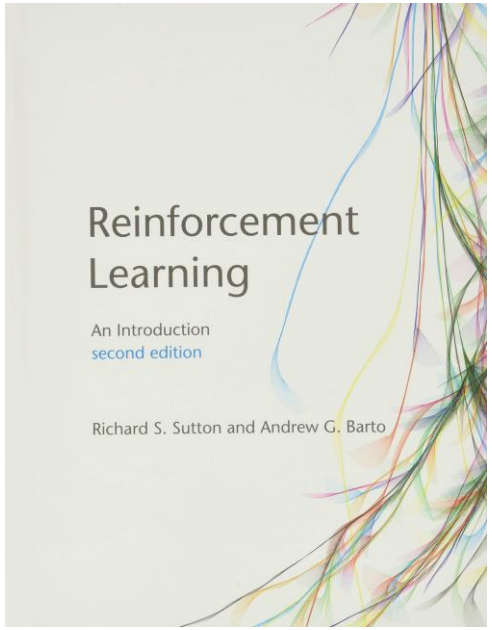
Learn about Ray/Rllib

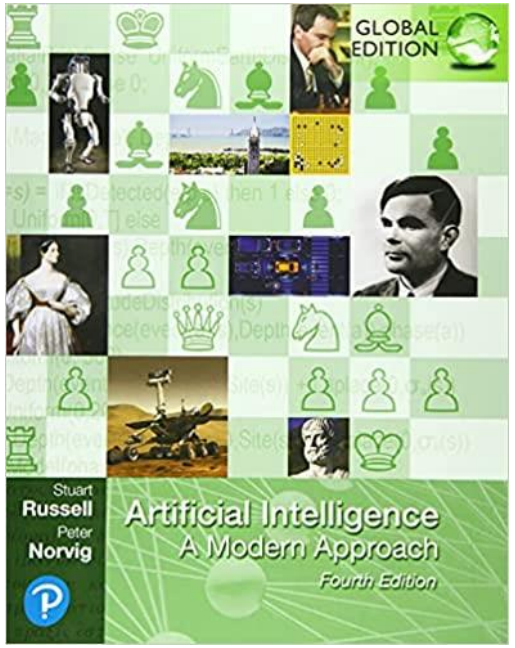
O'Reilly AI tutorial: Scalable AI and reinforcement learning with Ray

`http://aka.ms/mslibrary`



Easy, with code examples to reproduce





The standard textbook for AI

http://incompleteideas.net/book/RLbook2018.pdf

Math heavy but well explained here:

https://www.youtube.com/watch?v=4SLGEq_HZxk&t=0s&ab_channel=HenryAILabs