DEEP CHESS

https://github.com/S-parera/RL-chess_aidl

Students:	Sergi Sellés
	Pol García
	Sergi Parera
	Borja García
Advisor:	Dani Fojo

Motivation of the project

Initial Hypothesis & Key objectives

Critical Review Goals	✓	The algorithm is able to finish games.
	✓ ✗	The algorithm is able to beat a random-move player.
	✗	The algorithm is able to beat a 1000-elo player.
	✓	The algorithm runs smoothly (is optimized).
Extra Goal	✓ ✓	The algorithm is able to solve simpler environments.

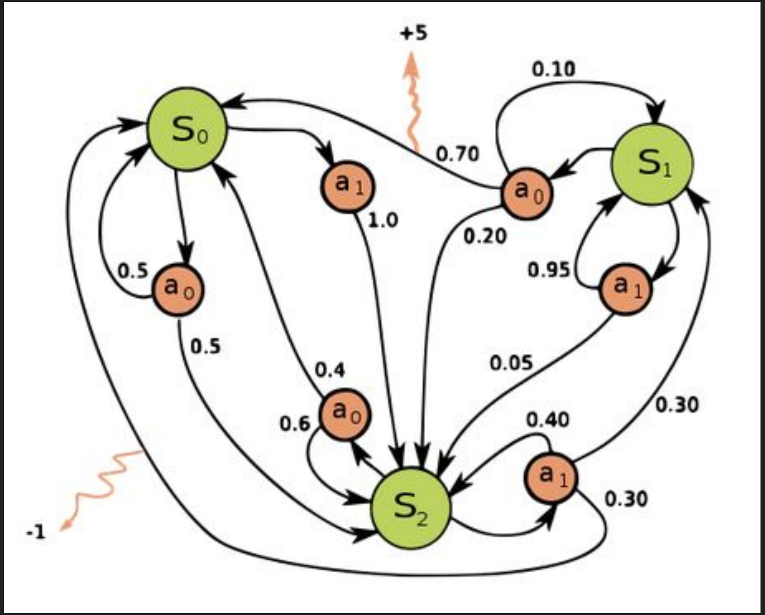
Brief introduction to RL

Markovian Decision Process

MDP=$\langle S,A,T,R,\gamma\rangle$

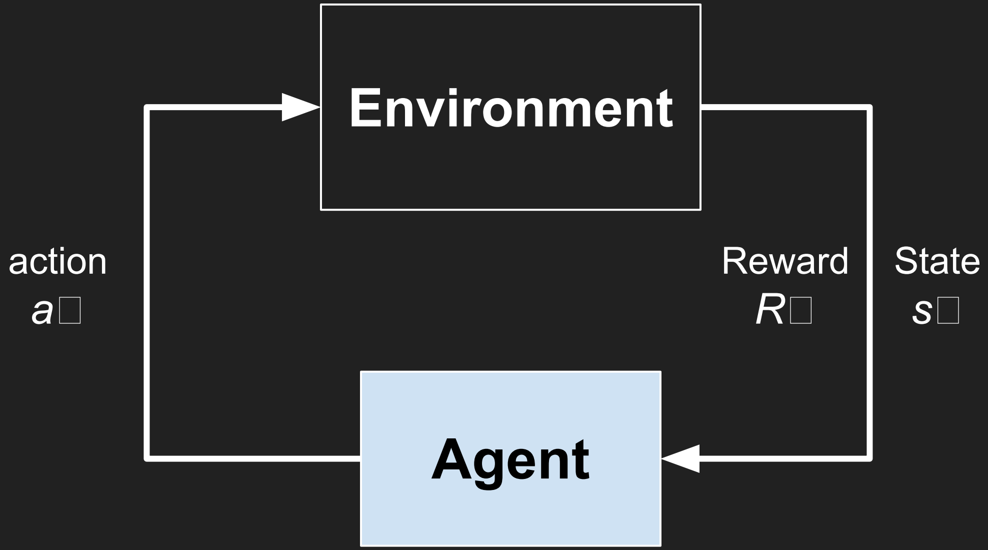
Learning < Decision Making

No expected outcome, just **REWARD**

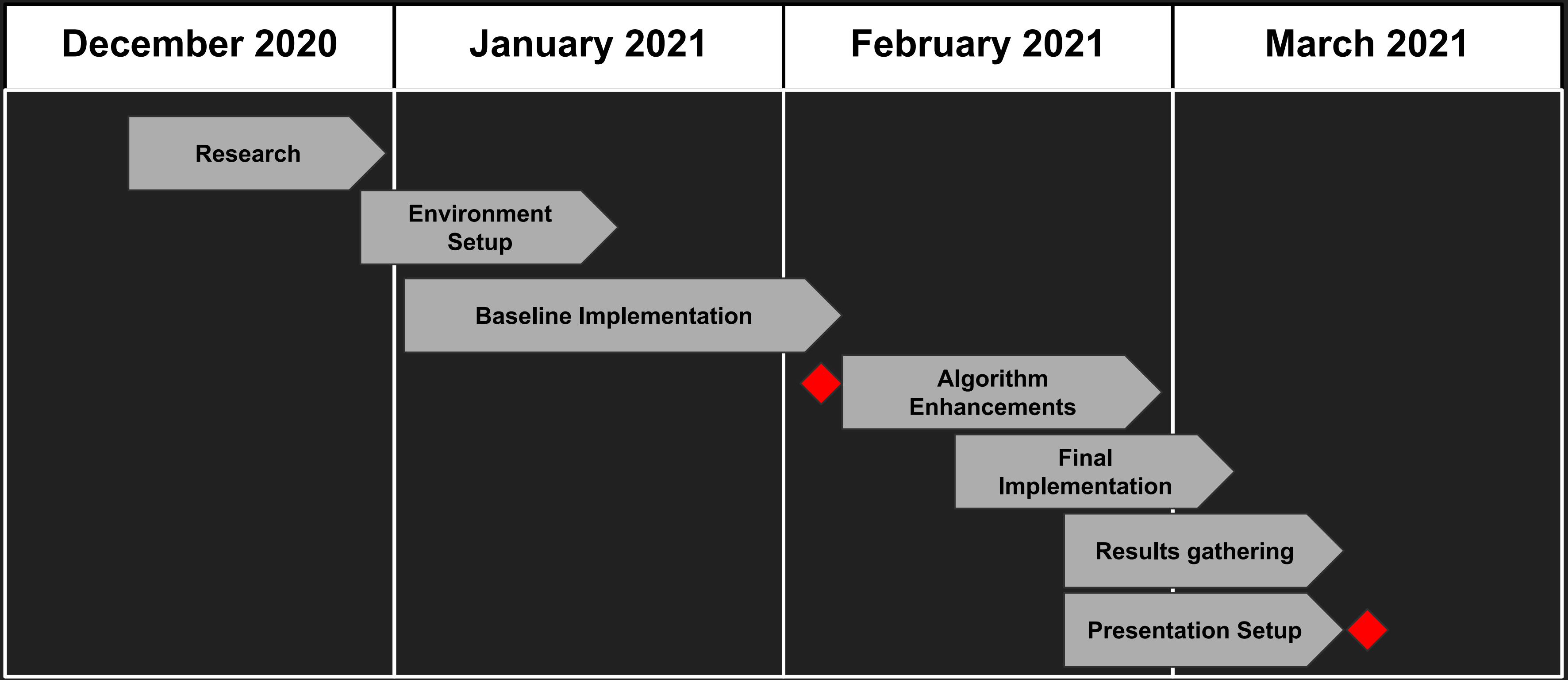


Brief introduction to RL

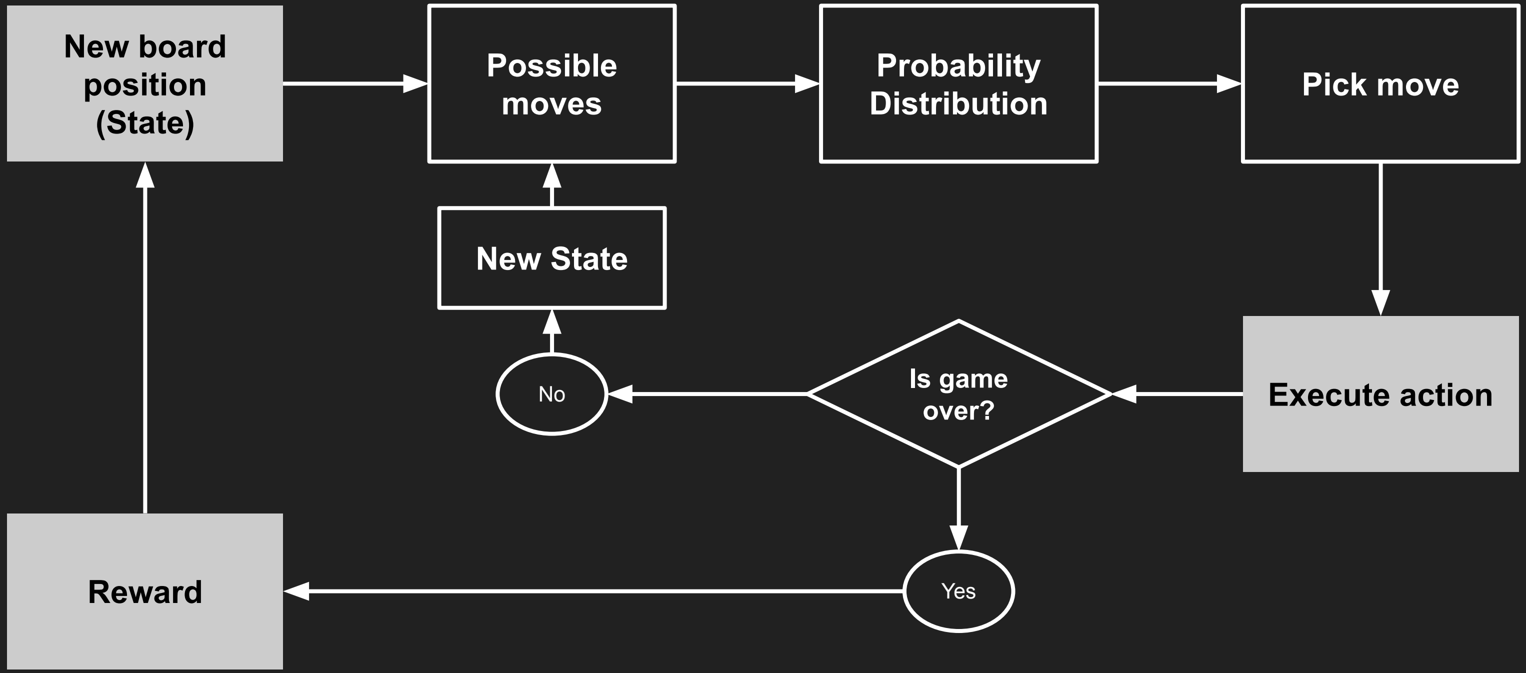
There is an **agent** in an unknown **environment** that gets **rewards** by performing **actions** in it. The main goal for the agent is to **maximize** the overall reward score.



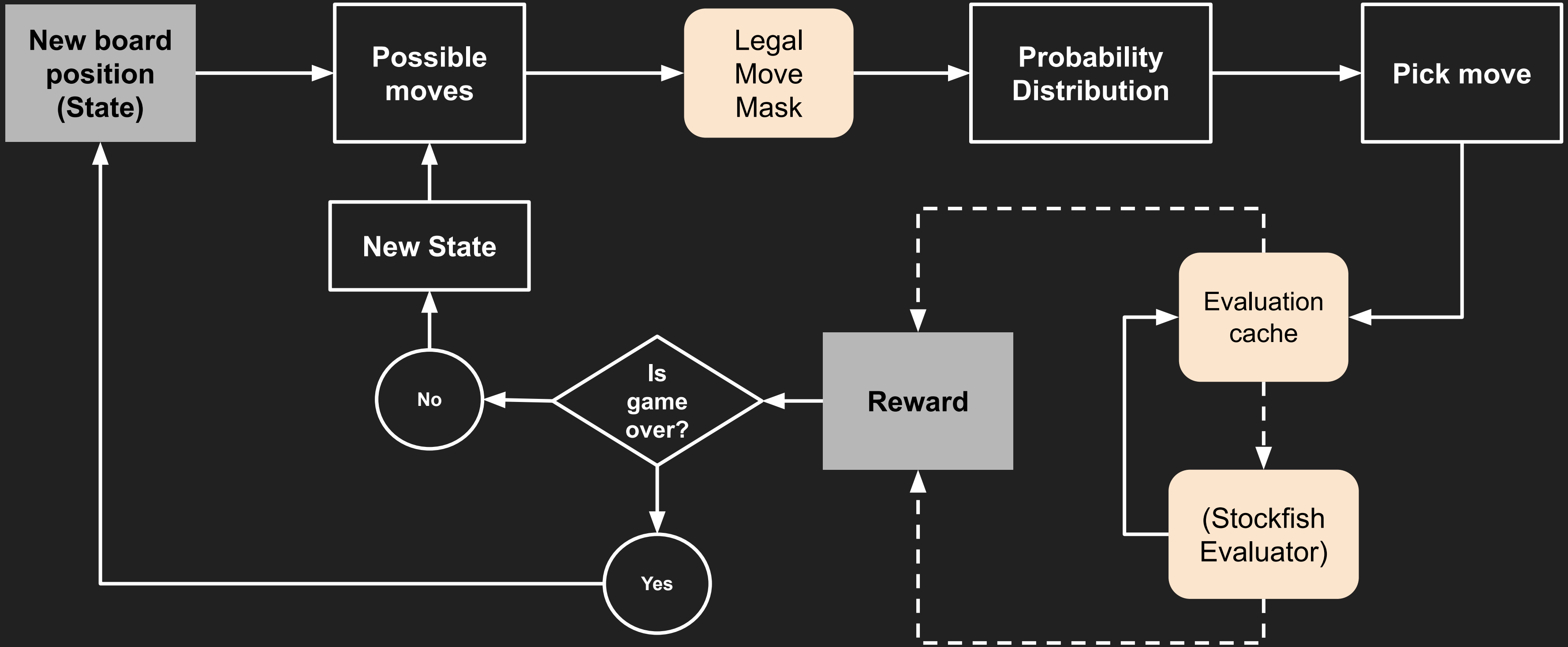
Final Project Plan



Algorithm Workflow (baseline)

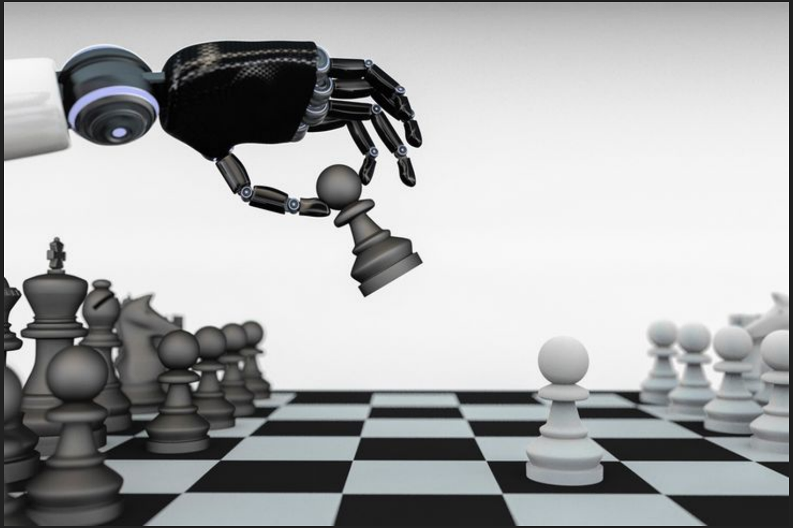


Algorithm Workflow (Enhanced)

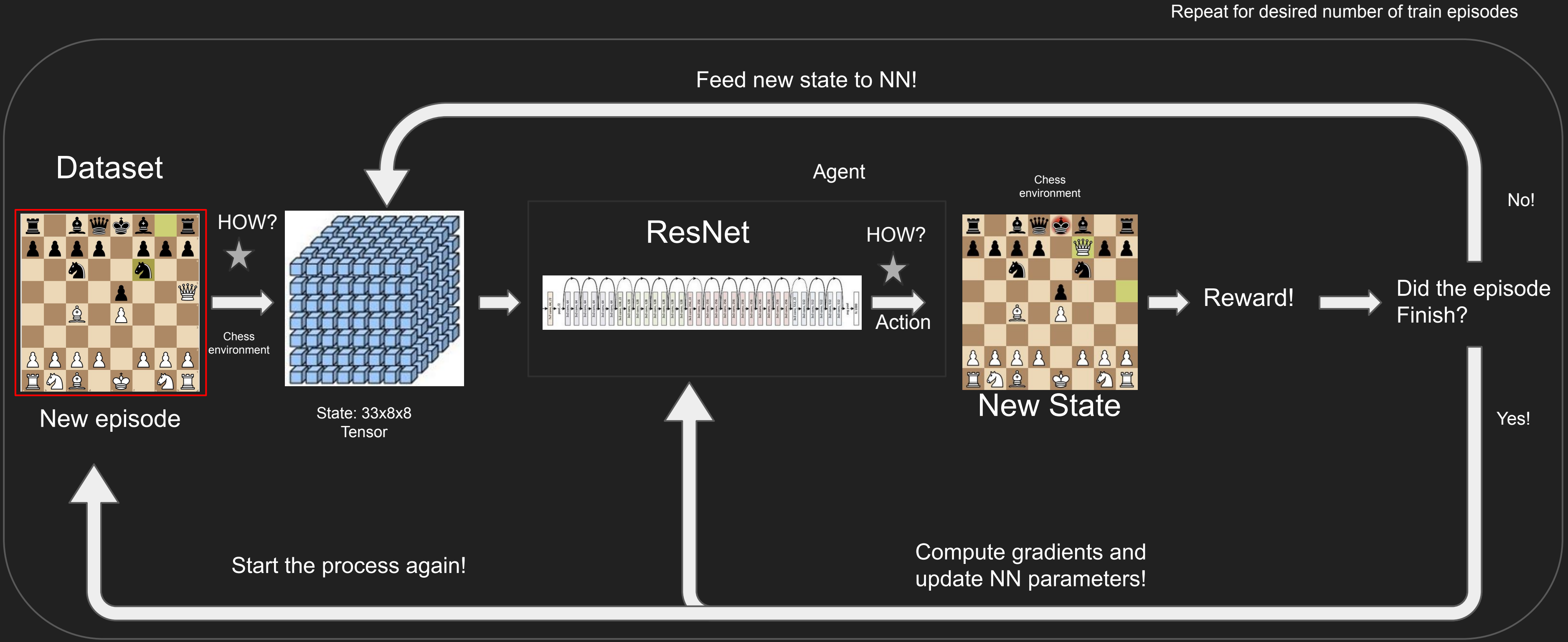


Experimental Environment: Training idea

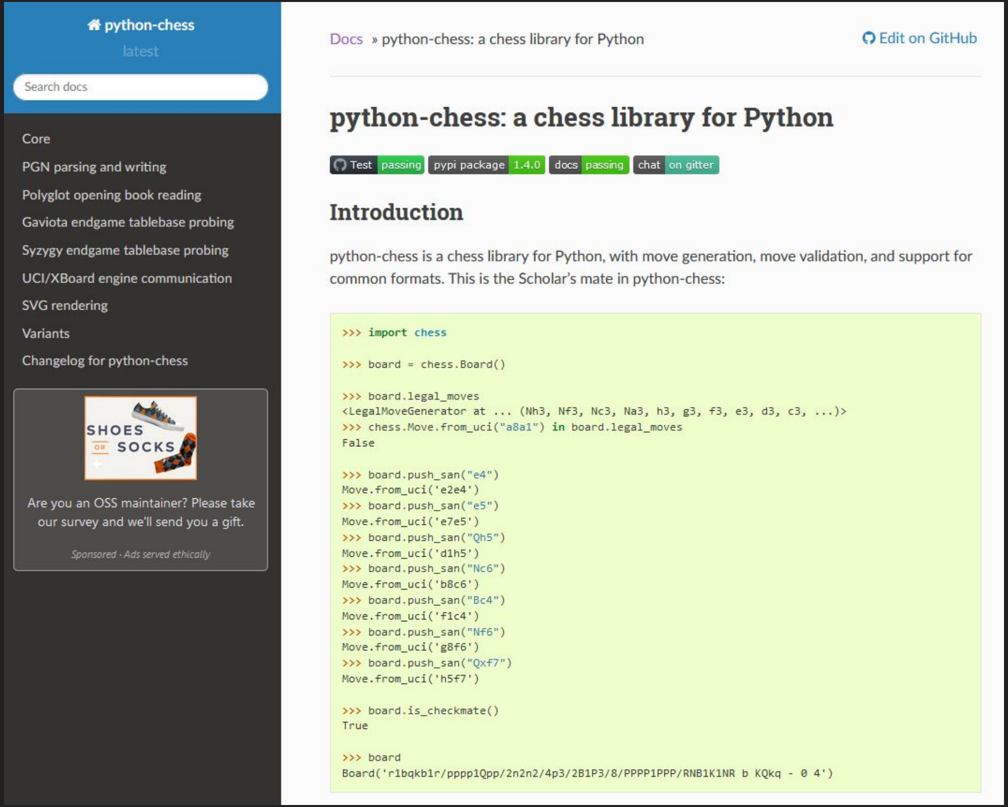
- Have the machine play against itself
- If checkmate → reward
- When average reward > threshold → Training is finished



Experimental Environment: Training pipeline



Experimental Environment: Python chess!


python-chess
latest
Search docs
Core
PGN parsing and writing
Polyglot opening book reading
Gaviota endgame tablebase probing
Syzygy endgame tablebase probing
UCI/XBoard engine communication
SVG rendering
Variants
Changelog for python-chess
SHOES OR SOCKS
Are you an OSS maintainer? Please take our survey and we'll send you a gift.
Sponsored · Ads served ethically
Docs » python-chess: a chess library for Python
Edit on GitHub
Test passing
pypi package 1.4.0
docs passing
chat on gitter

python-chess: a chess library for Python

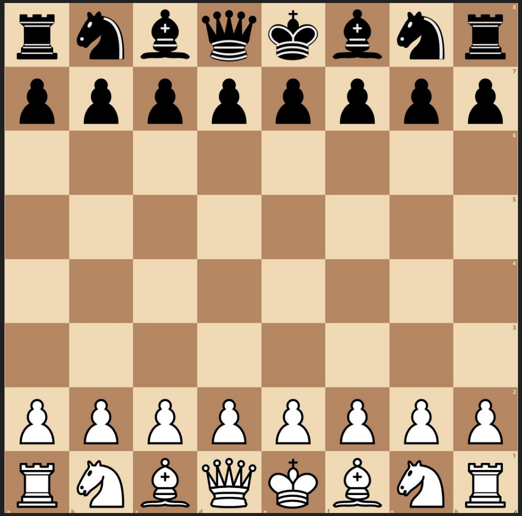
Introduction

python-chess is a chess library for Python, with move generation, move validation, and support for common formats. This is the Scholar's mate in python-chess:

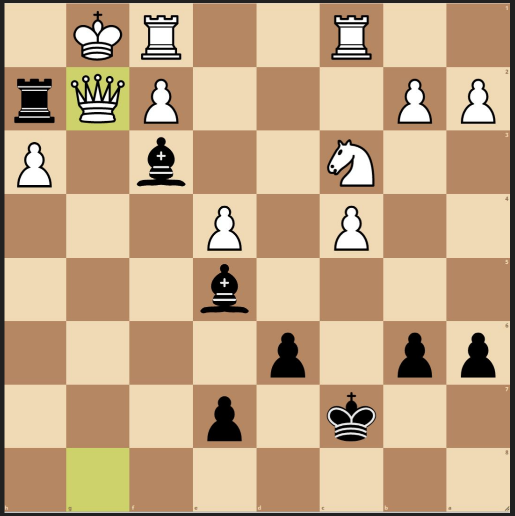
```
>>> import chess

>>> board = chess.Board()

>>> board.legal_moves
<LegalMoveGenerator at ... (Nh3, Nf3, Nc3, Na3, h3, g3, f3, e3, d3, c3, ...)>
>>> chess.Move.from_uci("a8a1") in board.legal_moves
False

>>> board.push_san("e4")
Move.from_uci('e2e4')
>>> board.push_san("e5")
Move.from_uci('e7e5')
>>> board.push_san("Qh5")
Move.from_uci('d1h5')
>>> board.push_san("Nc6")
Move.from_uci('b8c6')
>>> board.push_san("Bc4")
Move.from_uci('f1c4')
>>> board.push_san("Nf6")
Move.from_uci('g8f6')
>>> board.push_san("Qxf7")
Move.from_uci('h5f7')

>>> board.is_checkmate()
True

>>> board
Board('r1bqkb1r/pppp1Qpp/2n2n2/4p3/2B1P3/8/PPPP1PPP/RNB1K1NR b KQkq - 0 4')
```

Experimental Environment: Starting position

Initial chess board (20% of times)



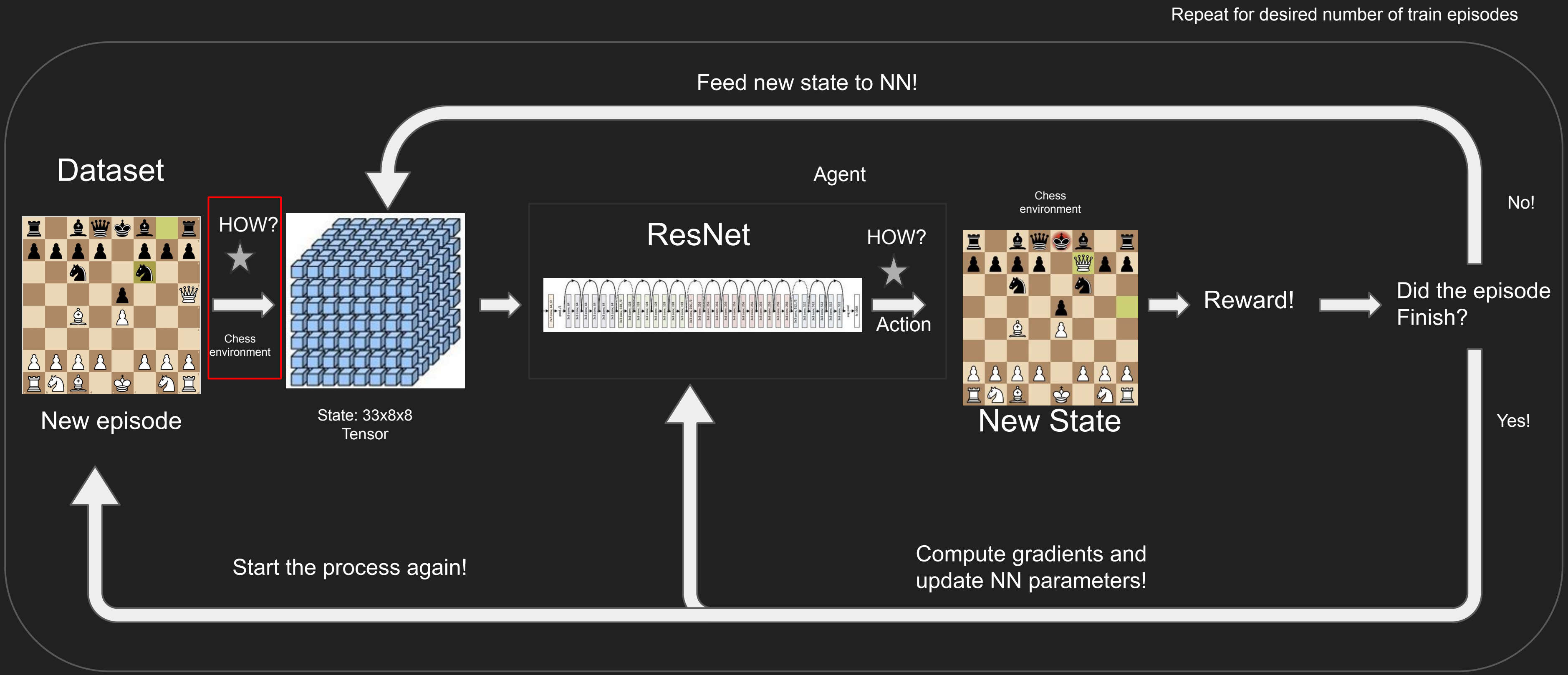
Tactical position (80% of times)



lichess.org

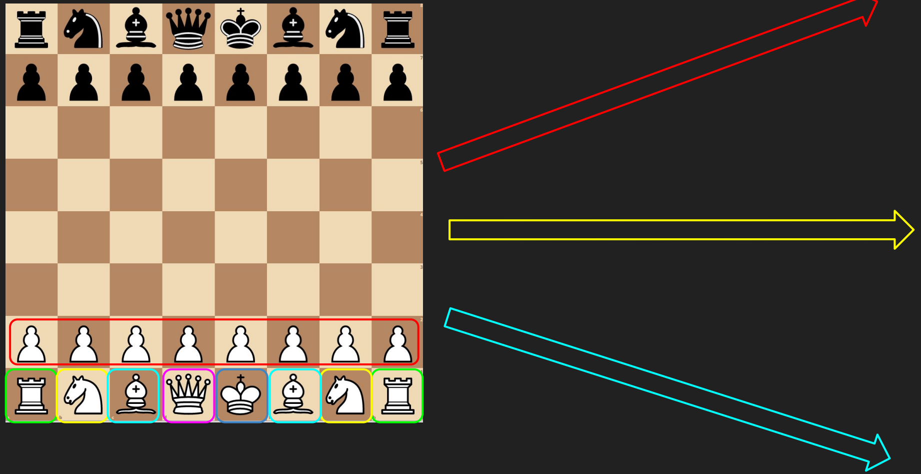
Dataset: 1 million tactical positions

Experimental Environment: Training pipeline



Experimental Environment: From board to Tensor 33x8x8

Chess Board



```
[[0 0 0 0 0 0 0 0]
 [1 1 1 1 1 1 1 1]
 [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 1 0 0 0 0 1 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]]
```

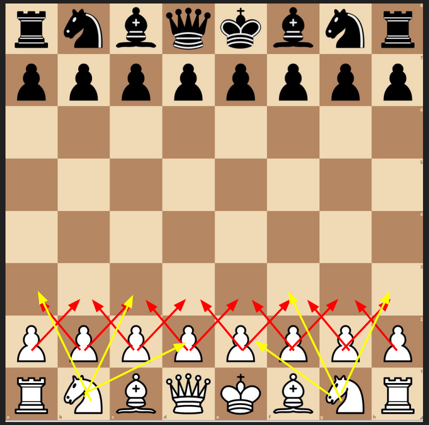
```
[[0 0 1 0 0 1 0 0]
 [0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0]]
```

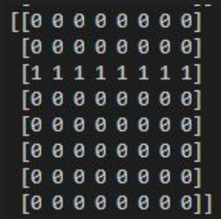
Boards 0:5 Where is each white piece type?

0: Pawns
1: Knight
2: Bishop
3: Rook
4: King
5: Queen

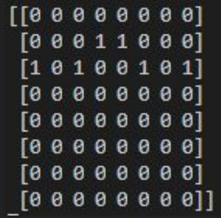
Experimental Environment: From board to Tensor 33x8x8

Chess Board





```
[[0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0]
 [1 1 1 1 1 1 1 1]
 [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]]
```



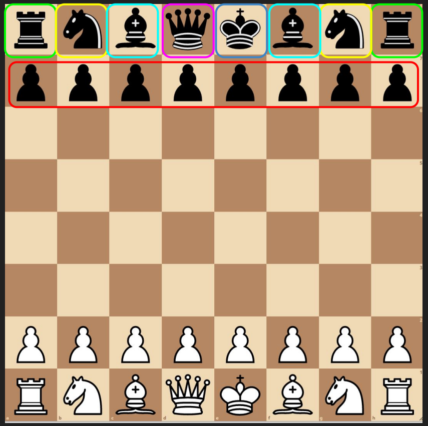
```
[[0 0 0 0 0 0 0 0]
 [0 0 0 1 1 0 0 0]
 [1 0 1 0 0 1 0 1]
 [0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0]]
```

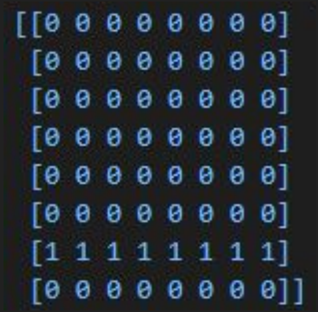
Boards 6:11: Which squares are white pieces attacking?

6: Pawns
7: Knight
8: Bishop
9: Rook
10: Queen
11: King

Experimental Environment: From board to Tensor 33x8x8

Chess Board





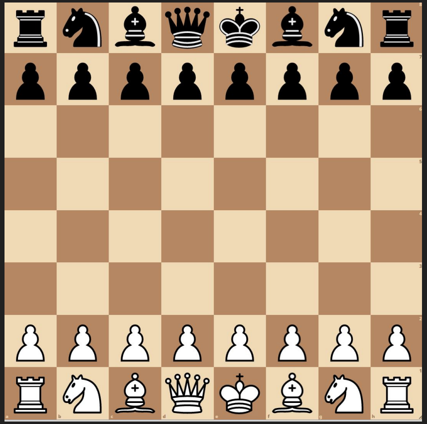
```
[[0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0]
 [1 1 1 1 1 1 1 1]
 [0 0 0 0 0 0 0 0]]
```

Boards 12:23 Same as 0:11 but for black pieces

12: Pawns
13: Knight
14: Bishop
15: Rook
16: Queen
17: King
18: Pawns attack squares
19: Knight attack squares
20: Bishop attack squares
21: Rook attack squares
22: Queen attack squares
23: King attack squares

Experimental Environment: From board to Tensor 33x8x8

Chess Board



Boards 24:32 encode other chess features

24: 1st board repetition
25: 2nd board repetition
26: turn (white or black)
27: Total move counter
28: Castling rights
29: Castling rights
30: Castling rights
31: Castling rights
32: Halfmove clock

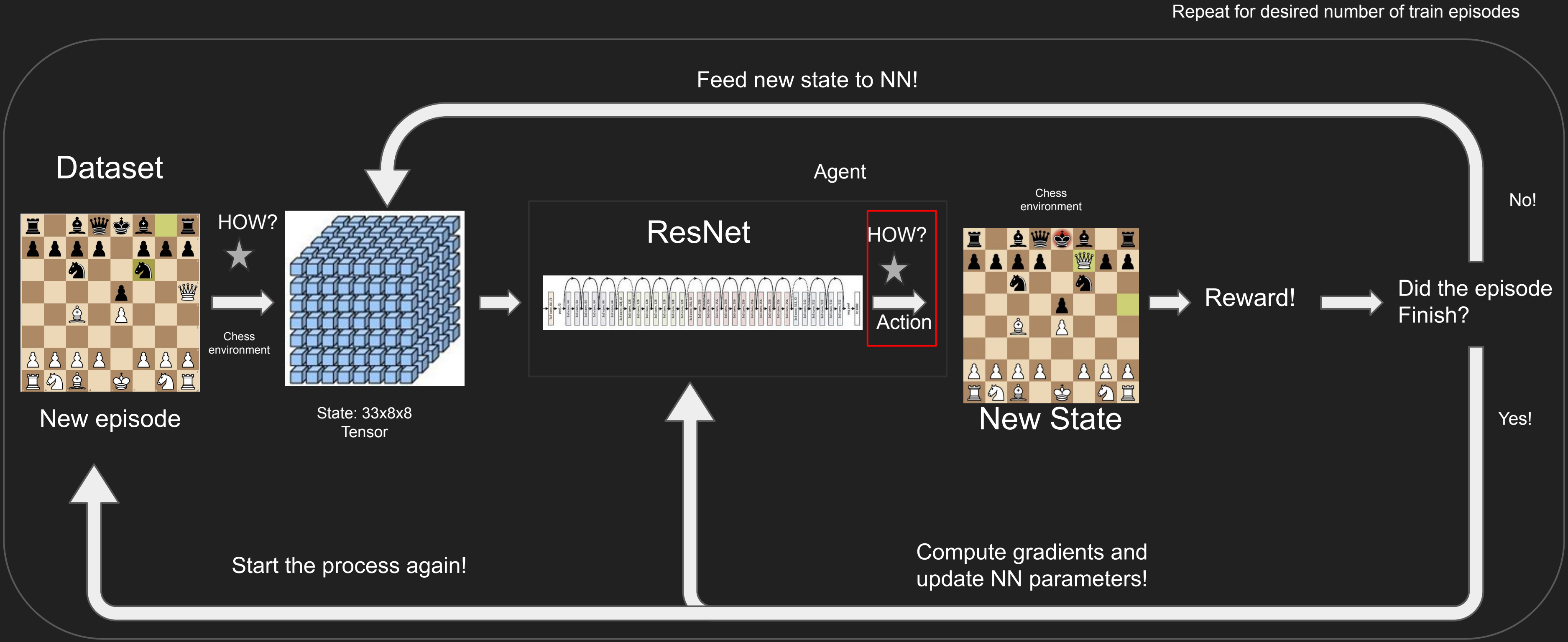
Black to move

```
[[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]]
```

White to move

```
[[1 1 1 1 1 1 1 1]
 [1 1 1 1 1 1 1 1]
 [1 1 1 1 1 1 1 1]
 [1 1 1 1 1 1 1 1]
 [1 1 1 1 1 1 1 1]
 [1 1 1 1 1 1 1 1]
 [1 1 1 1 1 1 1 1]
 [1 1 1 1 1 1 1 1]]
```

Experimental Environment: Training pipeline



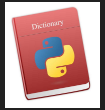
Experimental Environment: Movement encoding

Idea: Pick a piece in one square of the board (8x8) and leave it in another square of the board (8x8):

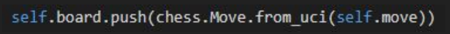
Totally 4272 movements are possible:

- 4096 possible moves
- 176 promotion moves

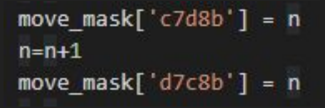
Dictionary



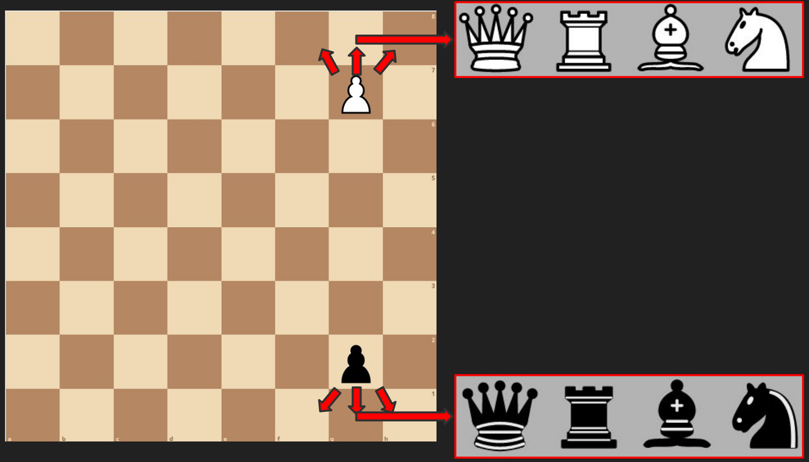
Python chess move



```
self.board.push(chess.Move.from_uci(self.move))
```

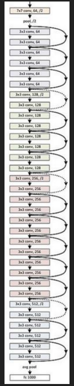


```
move_mask['c7d8b'] = n
n=n+1
move_mask['d7c8b'] = n
```



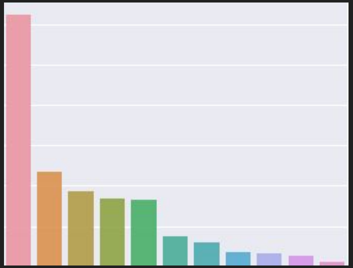
Experimental Environment: Movement & Back to board!

Neural Network outputs 1x4272 Tensor (logits)!



Mask out Ilegal moves

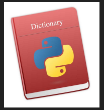
Convert to Categorical Distribution



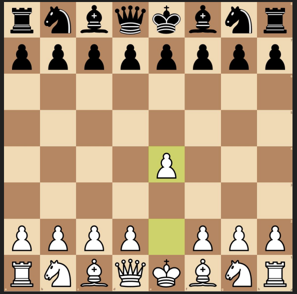
Choose action

m.sample()

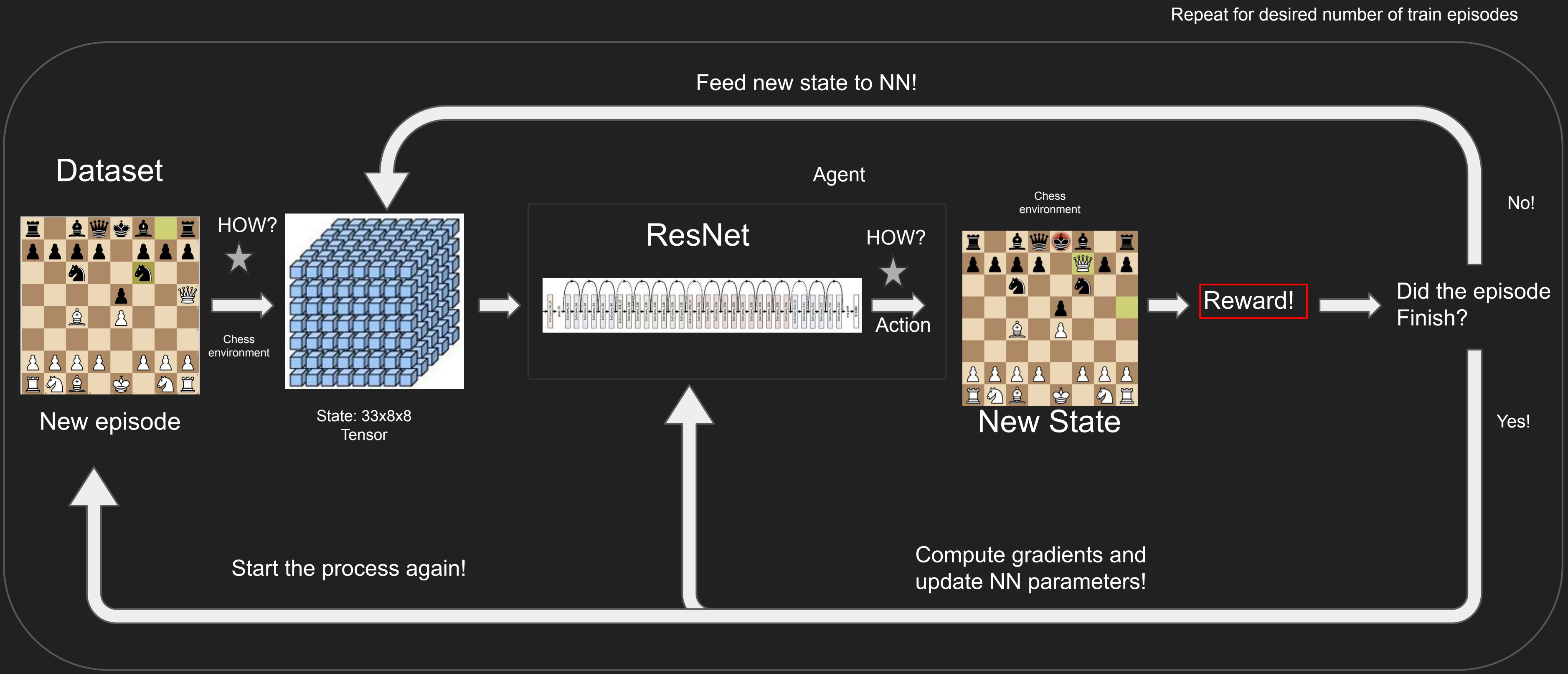
Dictionary



Move with Python chess!



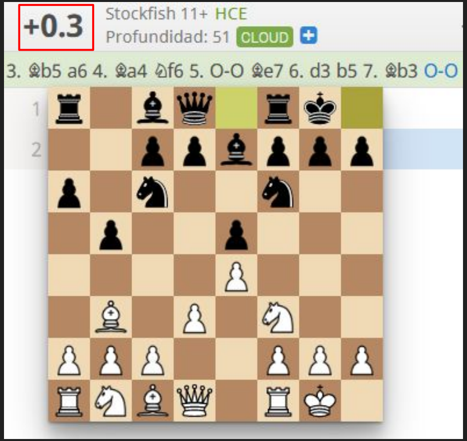
Experimental Environment: Training pipeline



Experimental Environment: Reward

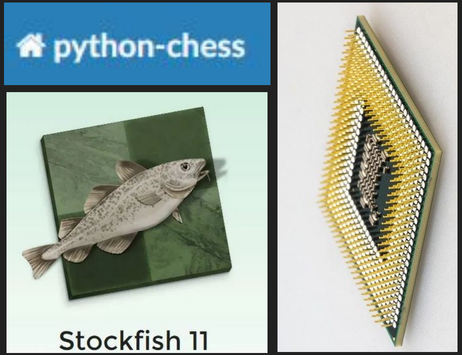
Was last movement good or bad?

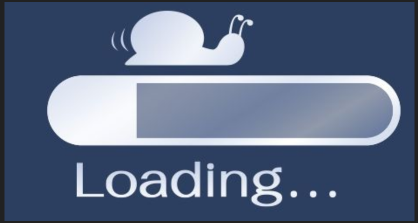
Reward = -abs(tanh(stockfish_t)-tanh(stockfish_t-1))



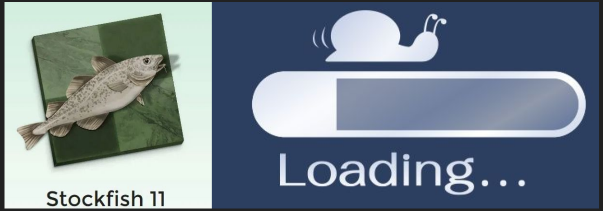
We will also give +1 reward if checkmate state is reached!

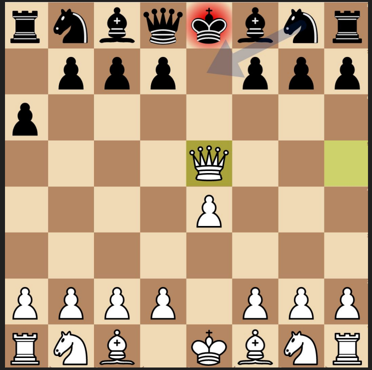
Experimental Environment: Problems





Experimental Environment: Problems (Cache/Hash Table)

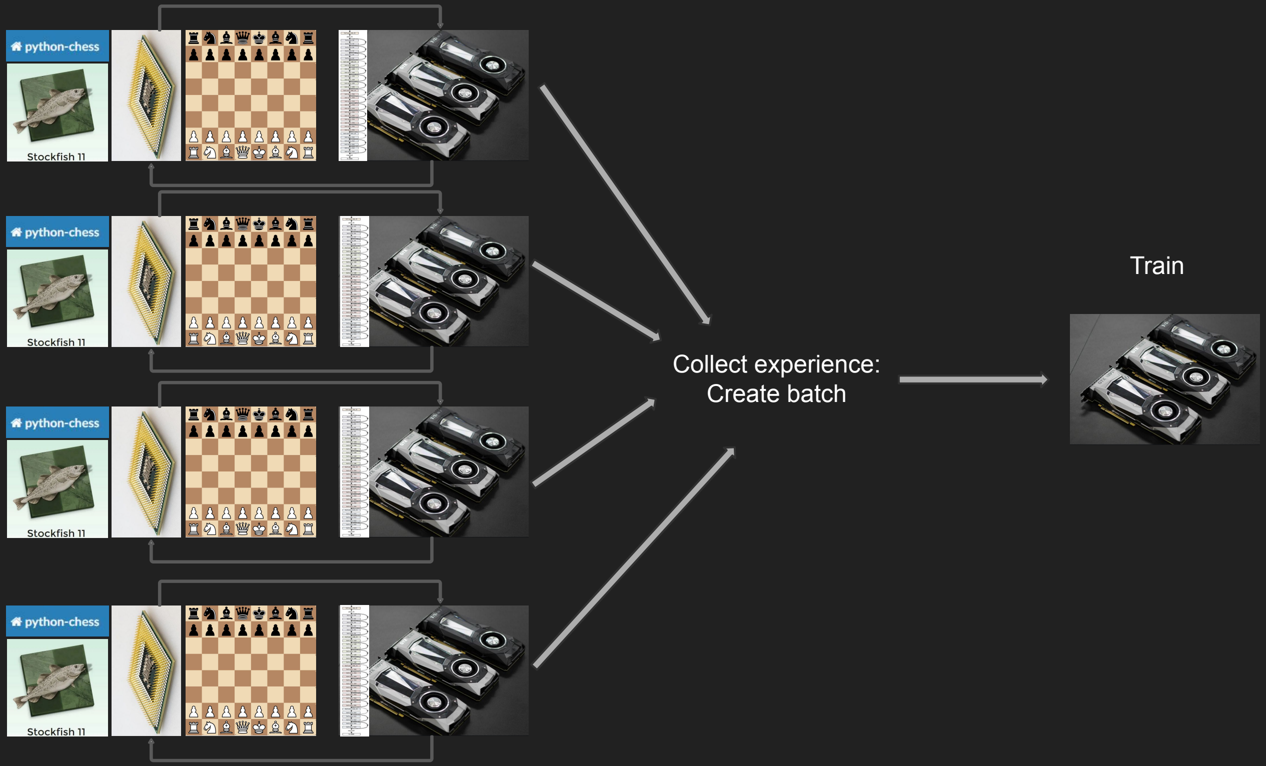




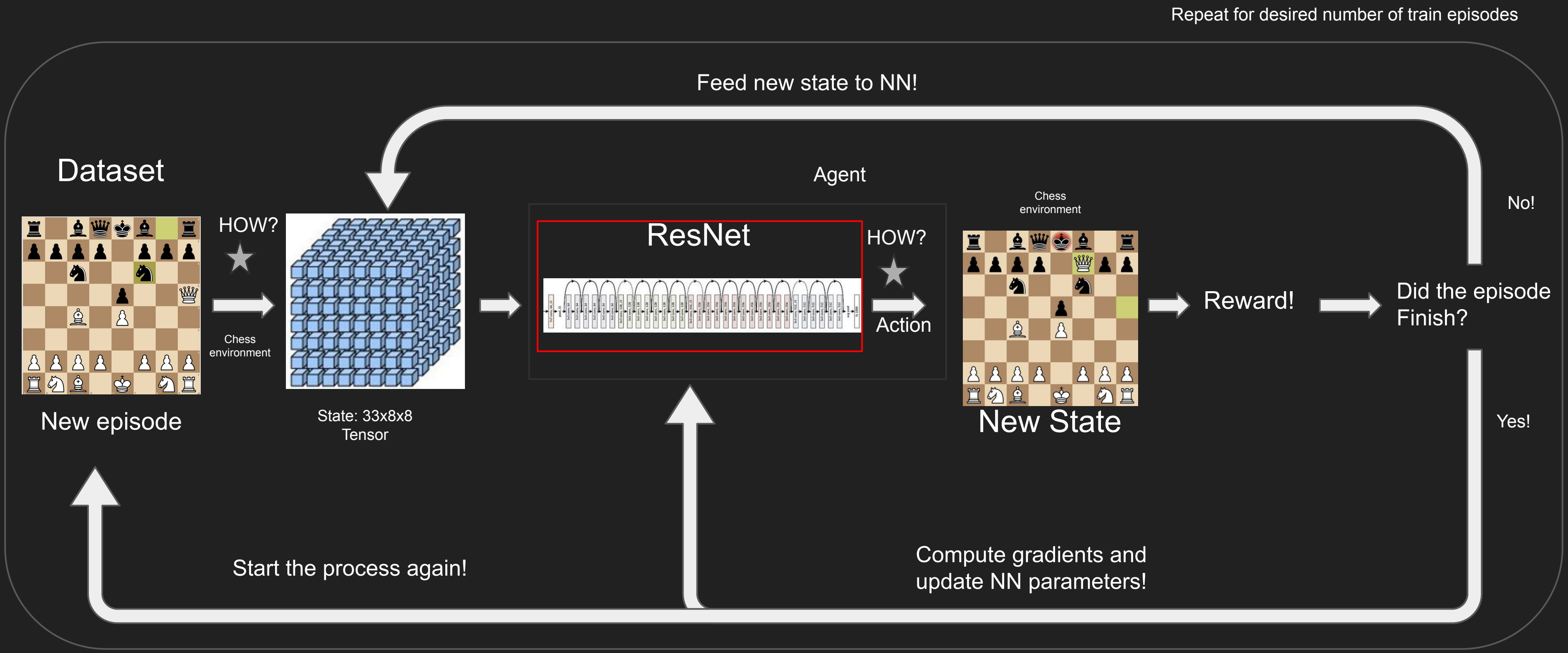
Save each position as {FEN: Value}

{rnbqkbnr/1ppp1ppp/p7/4Q3/4P3/8/PPPP1PPP/RNB1KBNR b KQkq - 0 3:+1.2}

Experimental Environment: Problems (Multithreading)

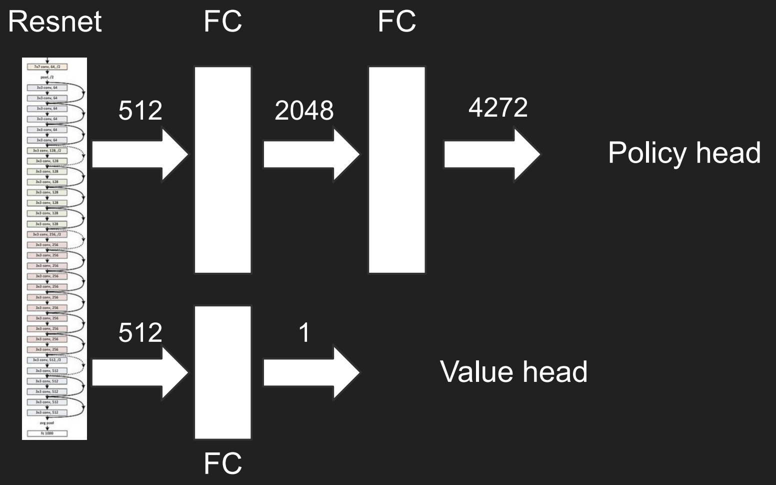


Experimental Environment: Training pipeline

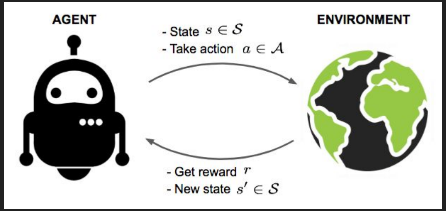


Model: NN topology

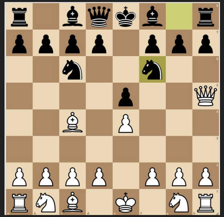
We tried many different topologies… Final one was:



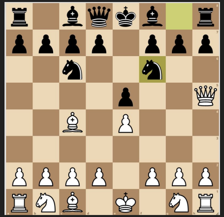
Algorithm's evolution


AGENT
ENVIRONMENT
- State $s \in \mathcal{S}$
- Take action $a \in \mathcal{A}$
- Get reward r
- New state $s' \in \mathcal{S}$

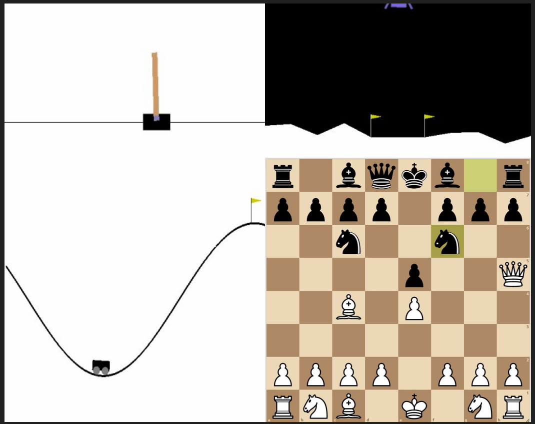
Policy Gradient



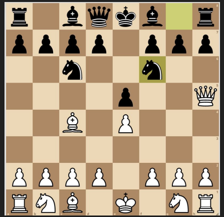
DQN



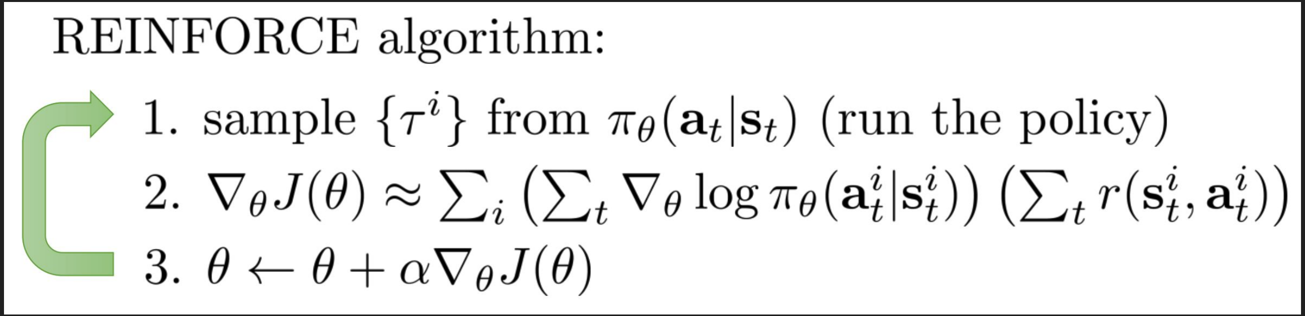
PPO



Supervised Learning

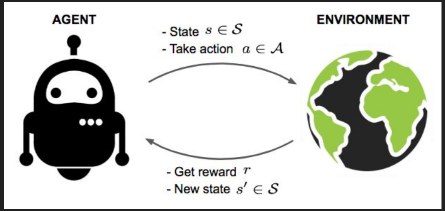


Algorithm’s evolution: Policy Gradient

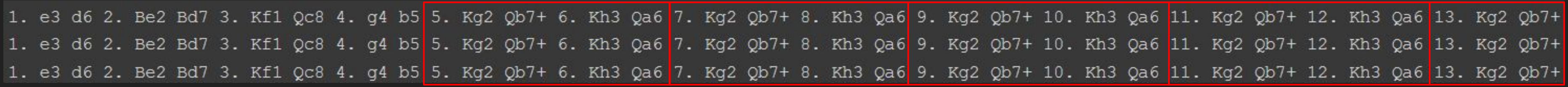


REINFORCE algorithm:

1. sample $\{\tau^i\}$ from $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ (run the policy)
2. $\nabla_\theta J(\theta) \approx \sum_i \left(\sum_t \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i)\right) \left(\sum_t r(\mathbf{s}_t^i, \mathbf{a}_t^i)\right)$
3. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$


AGENT
ENVIRONMENT
- State $s \in \mathcal{S}$
- Take action $a \in \mathcal{A}$
- Get reward r
- New state $s' \in \mathcal{S}$

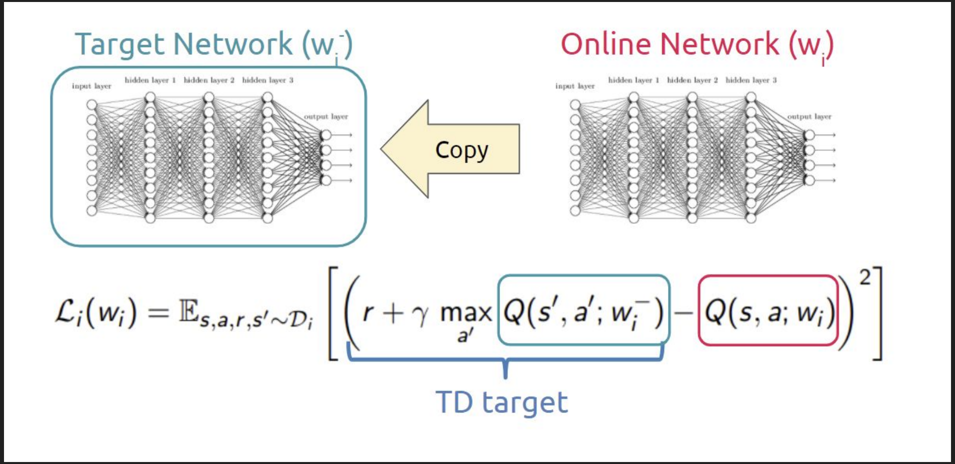
- Achievements: memorize legal moves and entire games as soon as quick way to reward is found



```
1. e3 d6 2. Be2 Bd7 3. Kf1 Qc8 4. g4 b5 5. Kg2 Qb7+ 6. Kh3 Qa6 7. Kg2 Qb7+ 8. Kh3 Qa6 9. Kg2 Qb7+ 10. Kh3 Qa6 11. Kg2 Qb7+ 12. Kh3 Qa6 13. Kg2 Qb7+
1. e3 d6 2. Be2 Bd7 3. Kf1 Qc8 4. g4 b5 5. Kg2 Qb7+ 6. Kh3 Qa6 7. Kg2 Qb7+ 8. Kh3 Qa6 9. Kg2 Qb7+ 10. Kh3 Qa6 11. Kg2 Qb7+ 12. Kh3 Qa6 13. Kg2 Qb7+
1. e3 d6 2. Be2 Bd7 3. Kf1 Qc8 4. g4 b5 5. Kg2 Qb7+ 6. Kh3 Qa6 7. Kg2 Qb7+ 8. Kh3 Qa6 9. Kg2 Qb7+ 10. Kh3 Qa6 11. Kg2 Qb7+ 12. Kh3 Qa6 13. Kg2 Qb7+
```

Algorithm's evolution: DQN

- DQN was trained to generate legal moves but the learning rate was too slow.
- PPO has better performance than DQN.



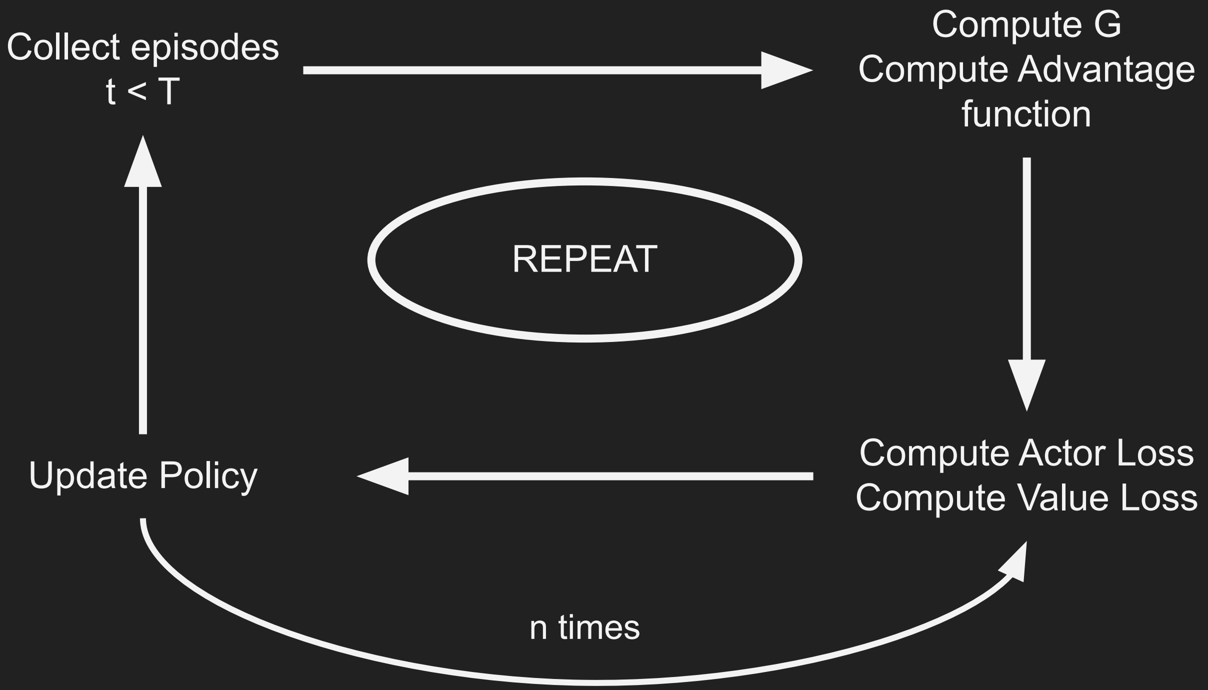
$$\mathcal{L}_i(w_i) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}_i}\left[\left(\underbrace{r + \gamma \max_{a'} \boxed{Q(s', a'; w_i^-)}}_{\text{TD target}} - \boxed{Q(s, a; w_i)}\right)^2\right]$$

Algorithm’s evolution: PPO

- Is an on-policy algorithm
- Continuous / discrete actions
- Limits the improvement not to worsen performance

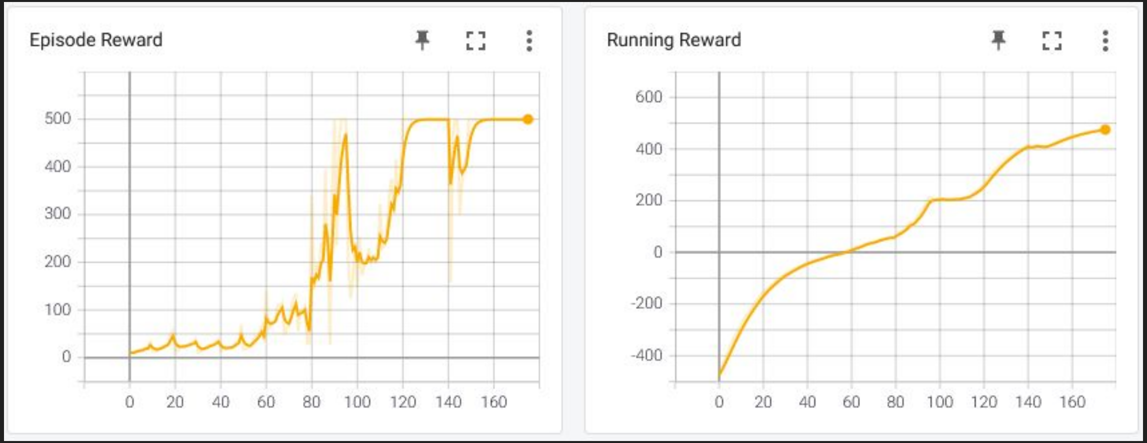
$$L(s, a, \theta_k, \theta) = \min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s, a), \;\; \text{clip}\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, 1 - \epsilon, 1 + \epsilon\right) A^{\pi_{\theta_k}}(s, a)\right),$$

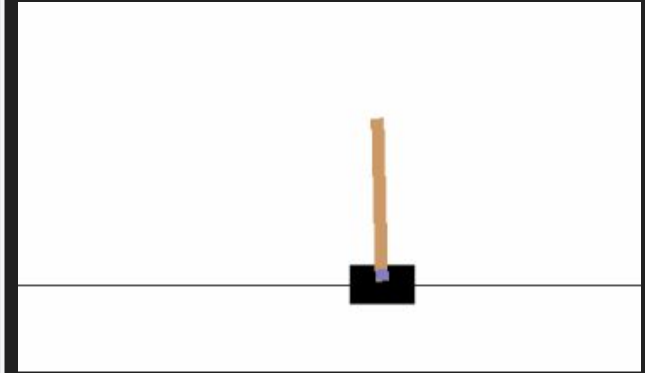
Algorithm's evolution: PPO



Algorithm’s evolution: PPO

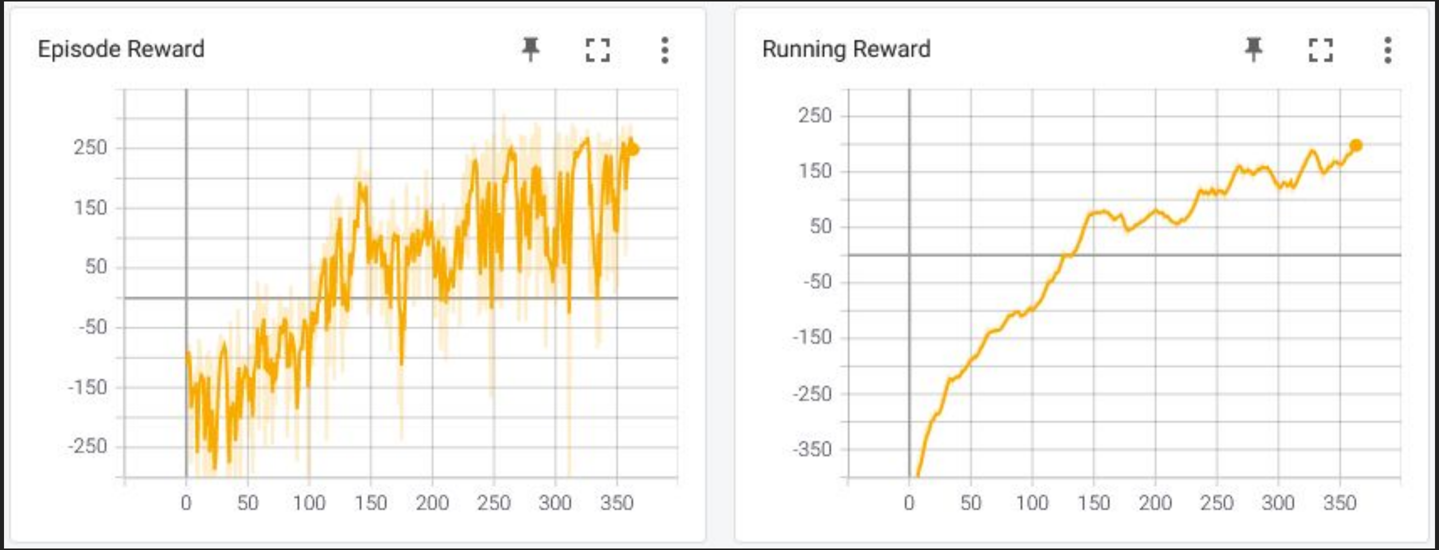
Results: Cart Pole





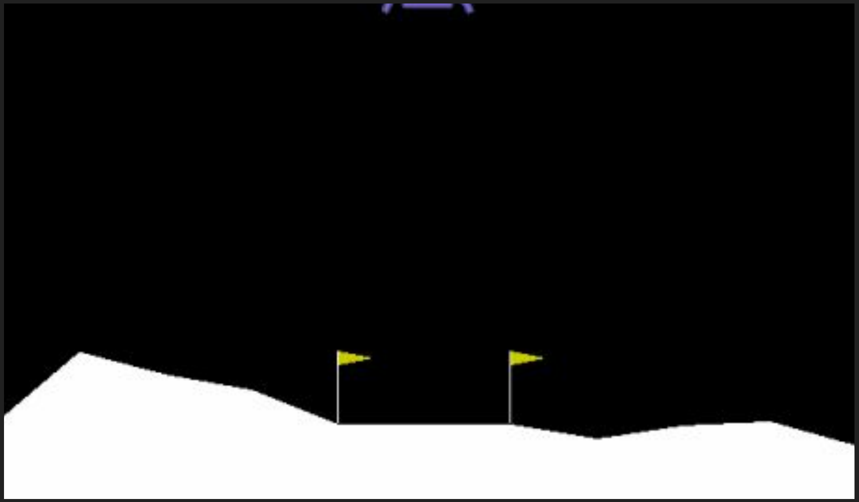
Algorithm's evolution: PPO

Results: Lunar Lander

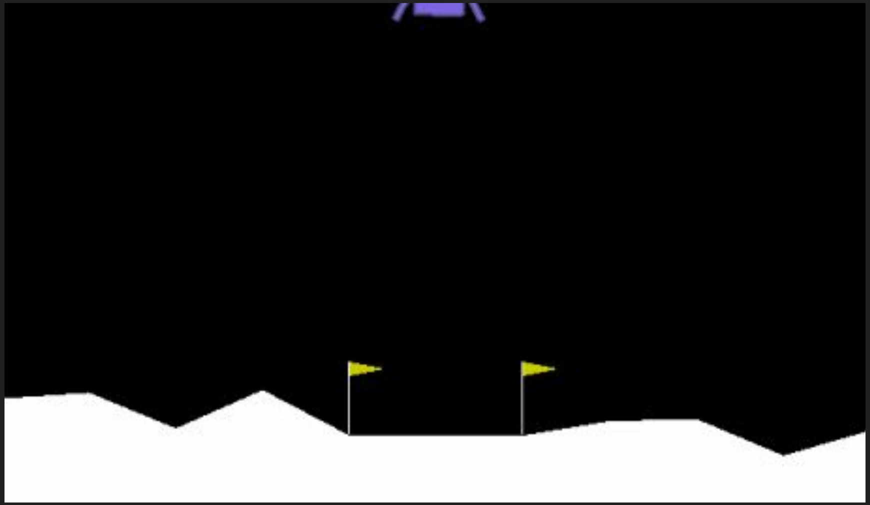


Algorithm’s evolution: PPO

Results: Lunar Lander



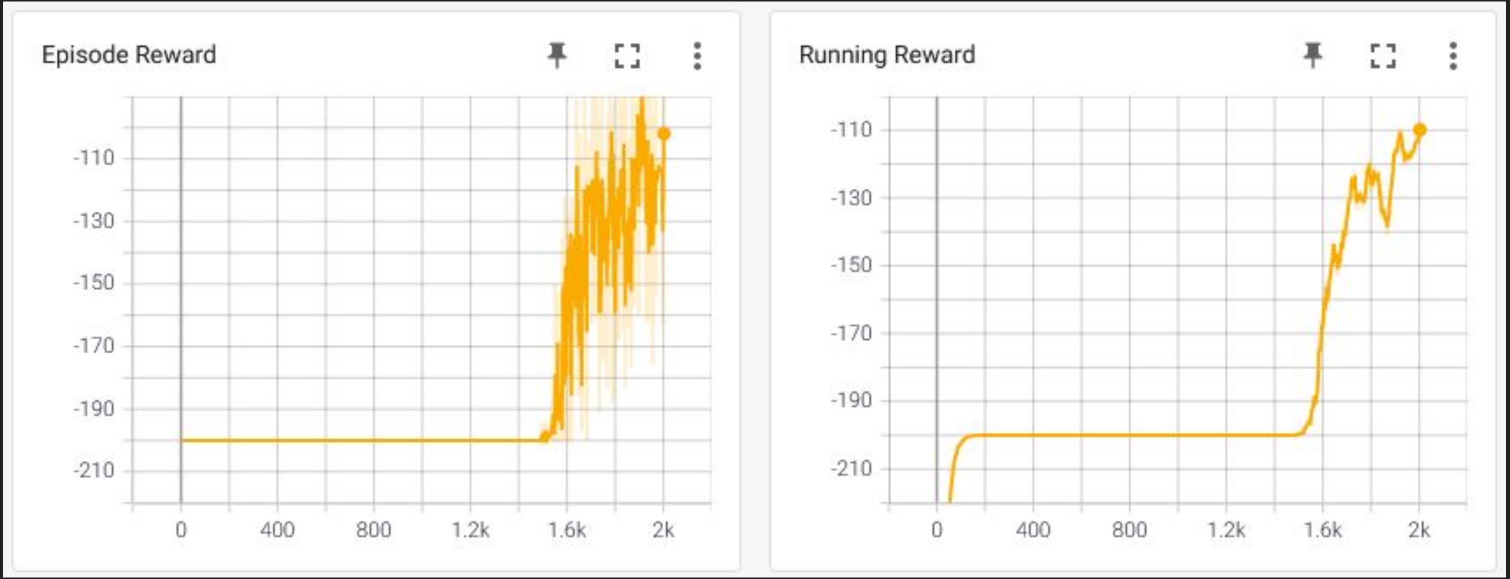
Untrained



Trained

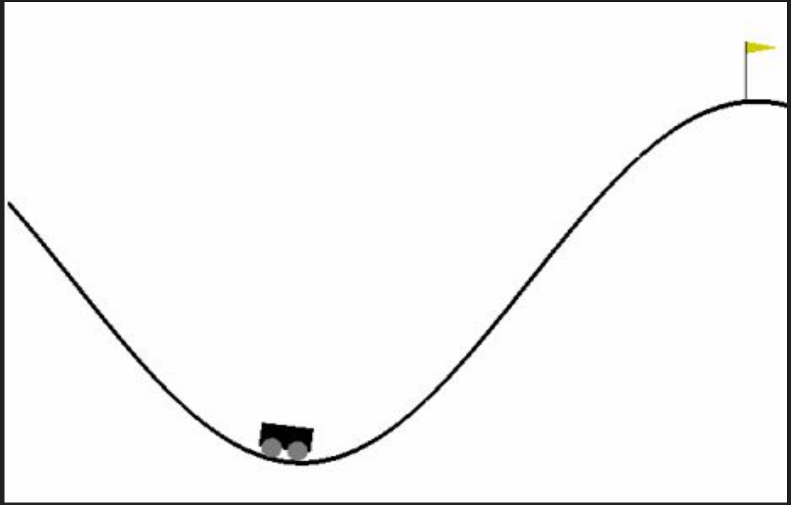
Algorithm’s evolution: PPO

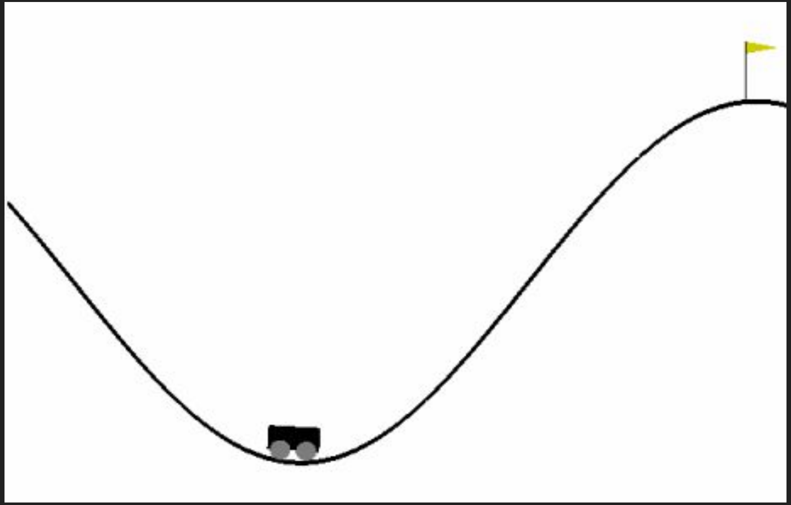
Results: Mountain Car



Algorithm’s evolution: PPO

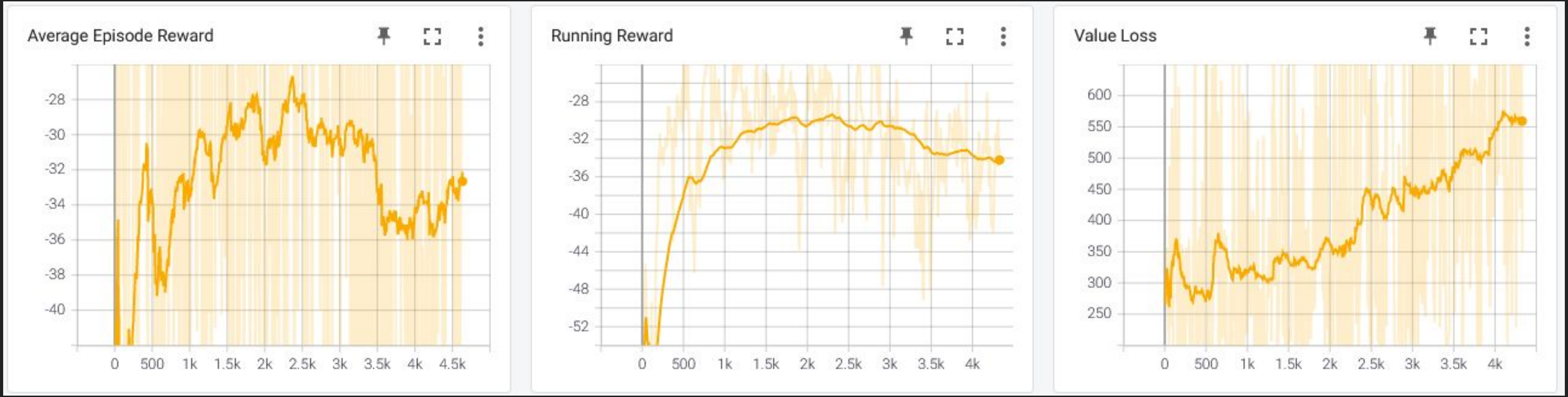
Results: Mountain Car


Untrained


Trained

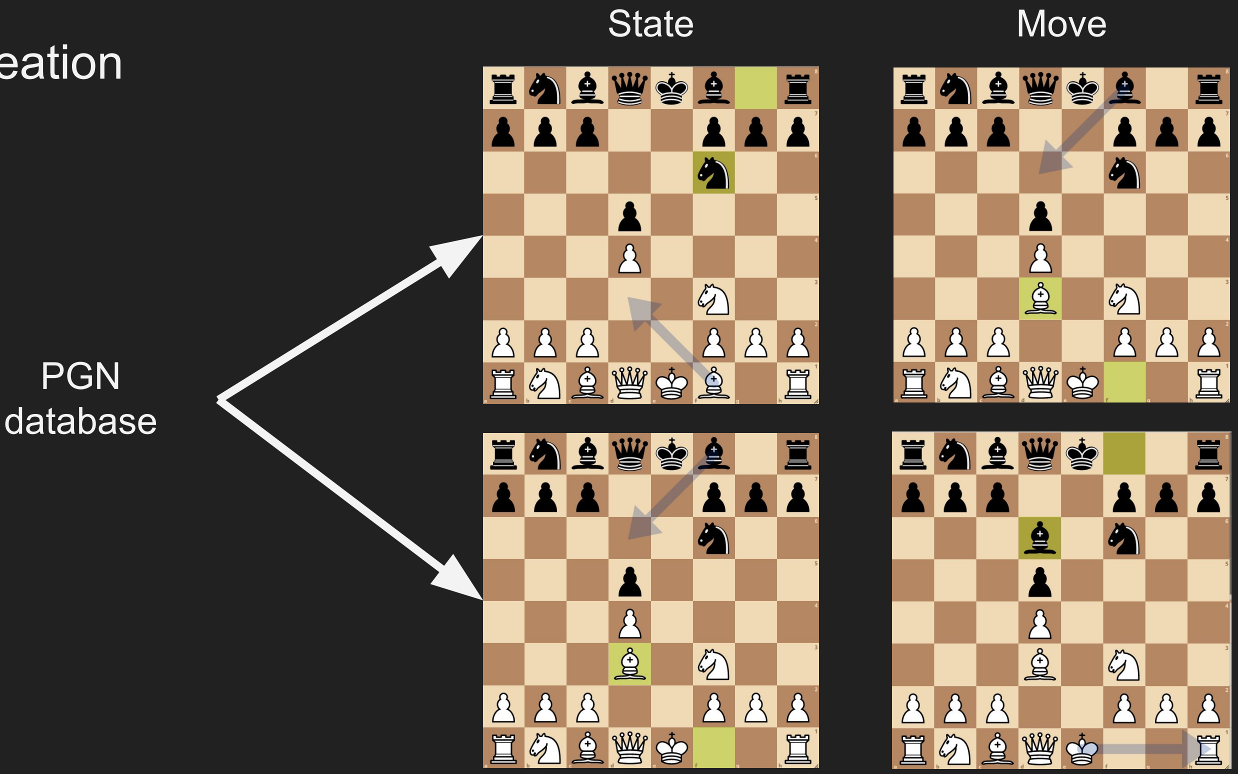
Algorithm’s evolution: PPO

Results: Chess



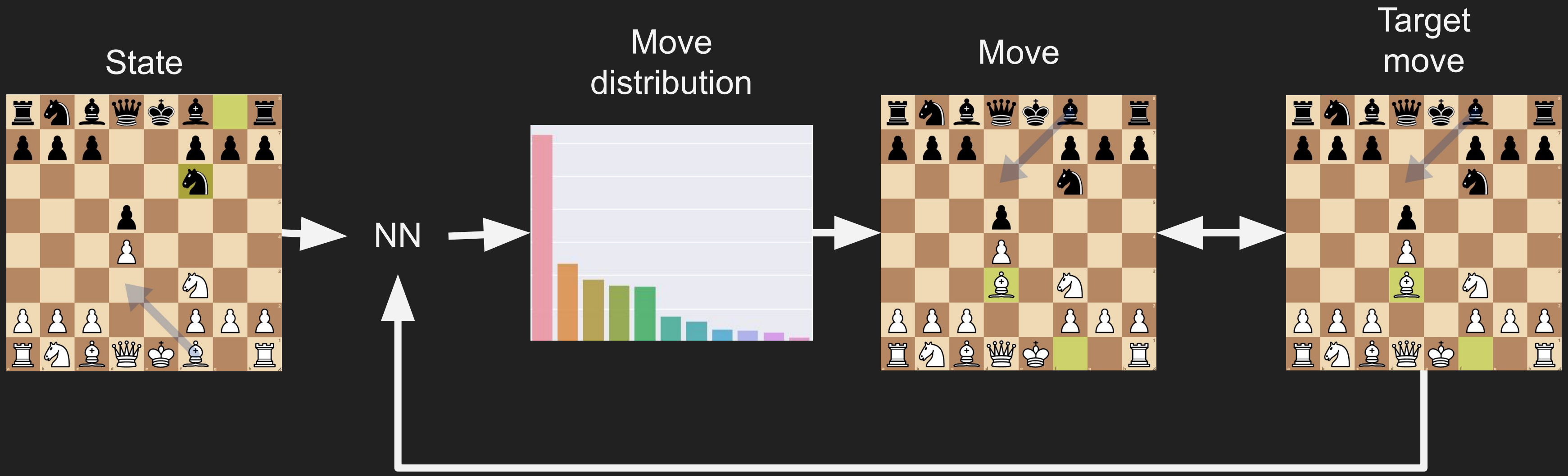
Algorithm's evolution: Supervised Learning

Dataset creation



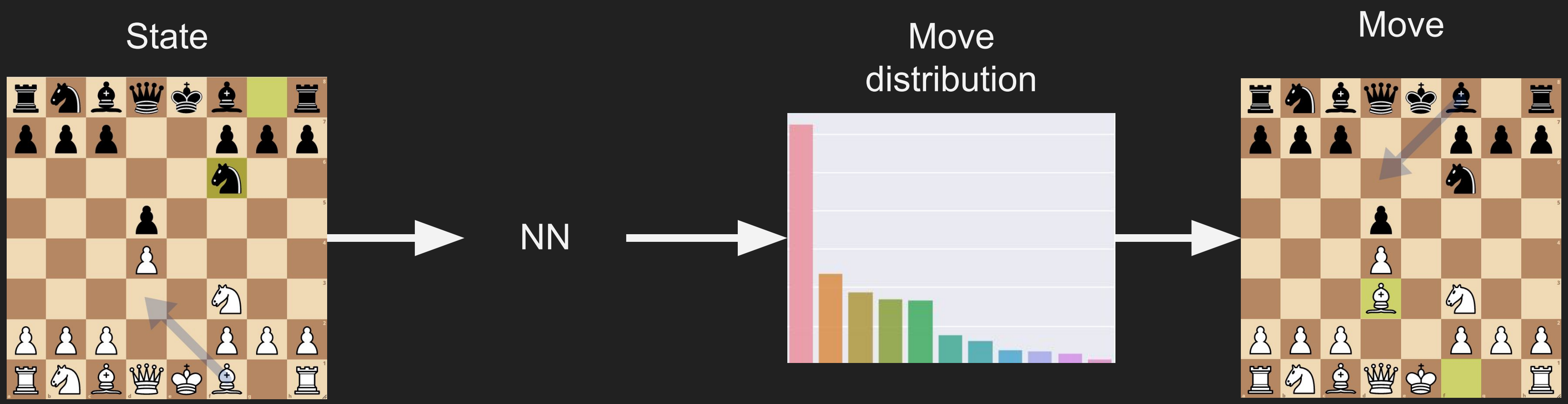
Algorithm’s evolution: Supervised Learning

Training



Algorithm's evolution: Supervised Learning

Accuracy



Algorithm’s evolution: Supervised Learning

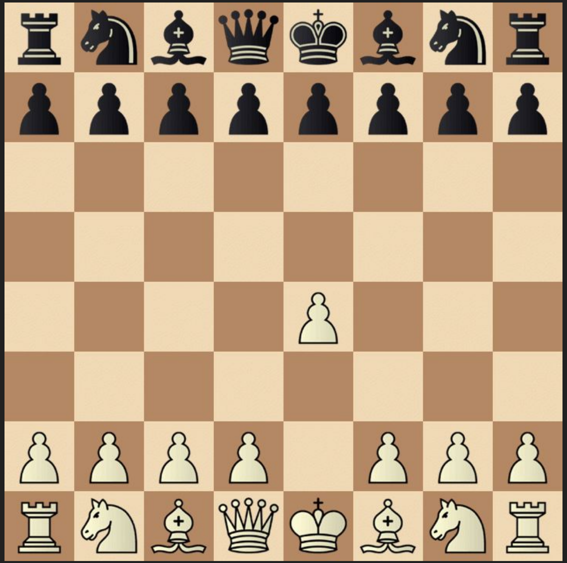
Results

White (network) wins: 11

Black (random) wins: 0

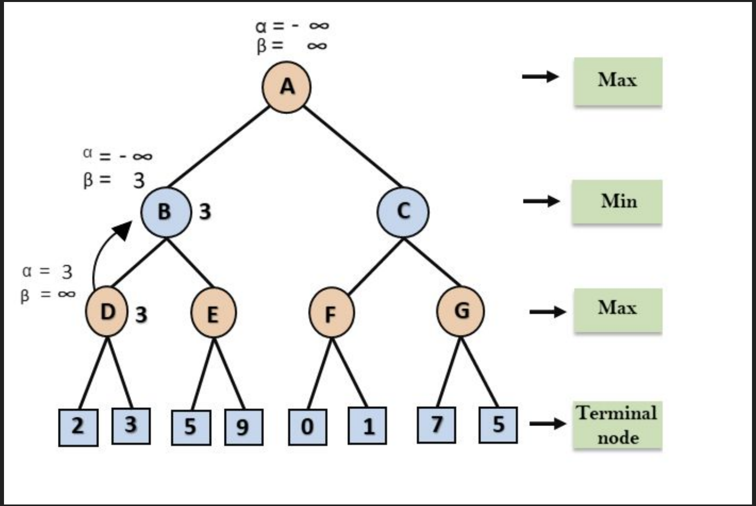
Draws: 4

Timeouts: 5



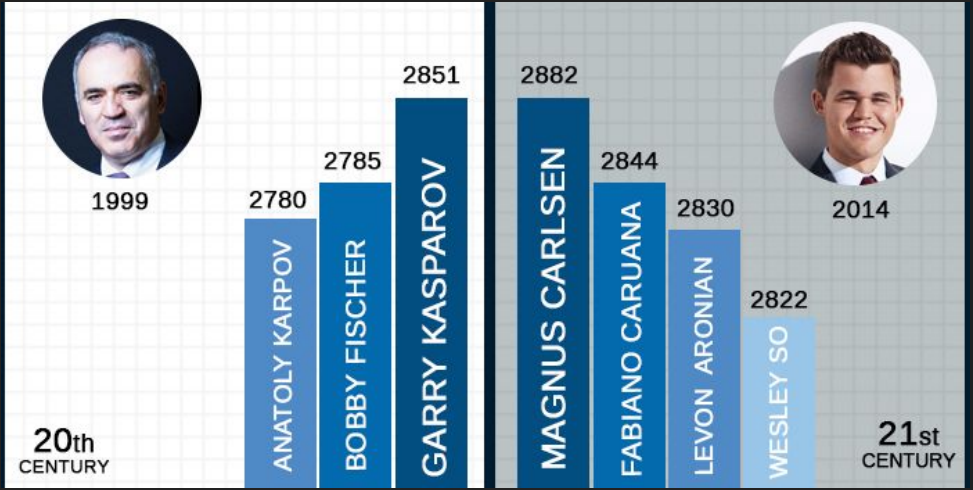
Evaluation

- Our trained policy/value net play games against another chess engine.
- At play the best move from the policy is selected with Alpha-beta tree search.



Evaluation

- ELO calculates the relative skills of players.
- ELO rating system implemented in the evaluation.



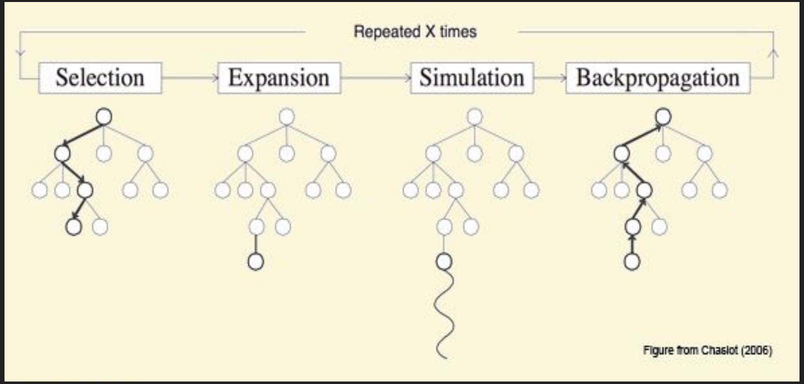
Results

- Policy Gradient: memorizes legal moves and entire games as soon as quick way to reward is found.
- DQN: managed to make +10 legal moves in a row. Very slow learning.
- PPO: solved Cartpole, Lunar Lander, Mountain Car.
- Supervised Learning: shows chess knowledge. It can beat a random-move player.
- PPO-chess: too slow progress in training chess. It requires implementing MCTS.

Results

- Monte Carlo Tree Search (MCTS) is implemented by Alpha Zero and others to select the best possible move in training and also at play.

Chess complexity = 10 ^123 !!!



- PPO-chess: MCTS prepared but no time to complete run.

Conclusions

- The algorithm Works:

  The PPO algorithm works and is able to play games. It solves Lunar Lander and Mountain Car.

- The algorithm is not just random / aims to win:

  When trained with supervised learning, it can beat a random player and shows chess knowledge.

- Training our chess engine with PPO alone is not enough. MCTS is necessary.

Next Steps

- PPO training with MCTS.
- Use computing resources in Google Cloud Platform:
    - VM instance prepared but no time to run MCTS.

THANK YOU!