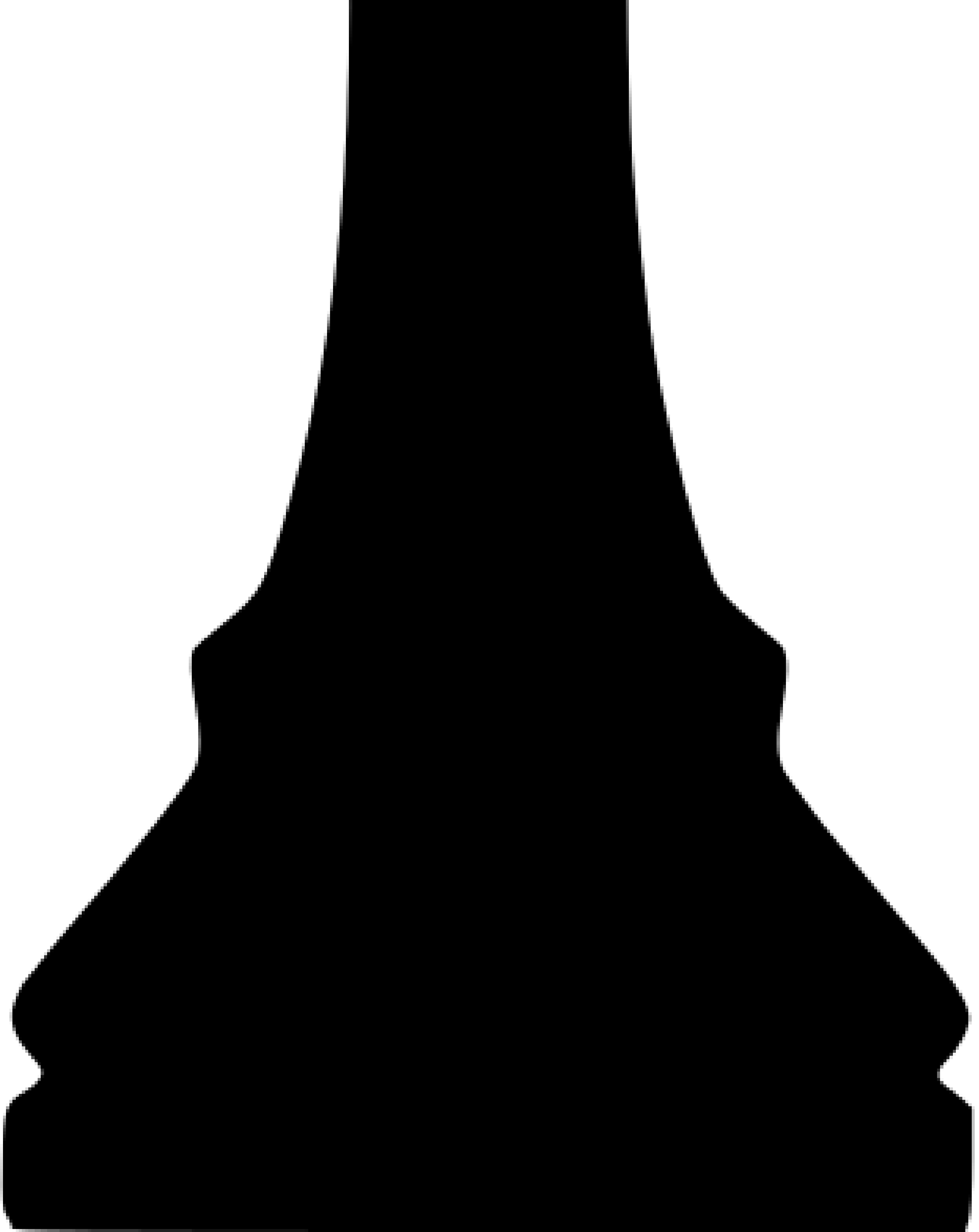
RL

Project

CHESS ENGINE

Michele Bartesaghi
Guido Giacomo Mussini

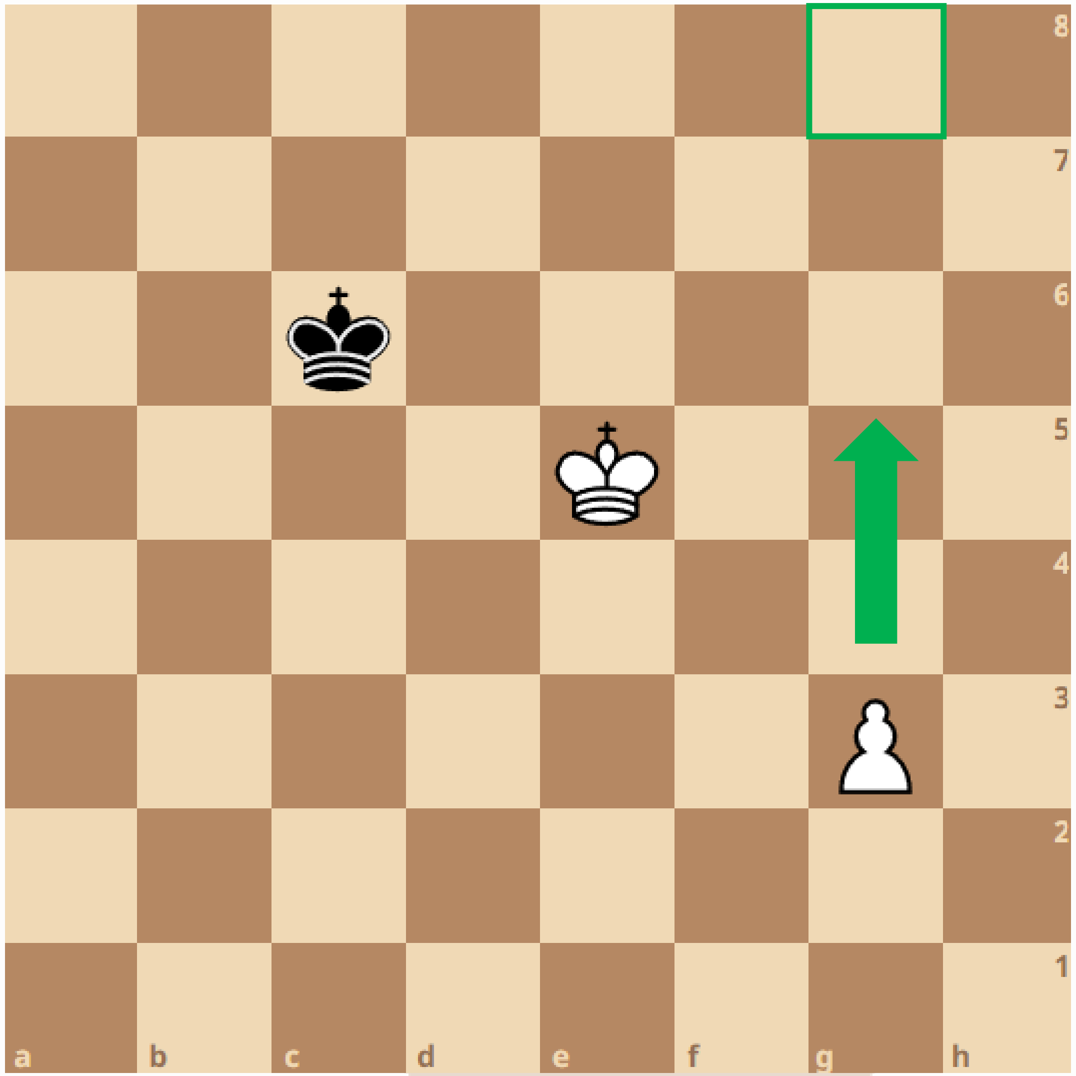


Outline

- Problem set
- Linear algebraic representation
- States
- Actions
- The Opponent: Stockfish

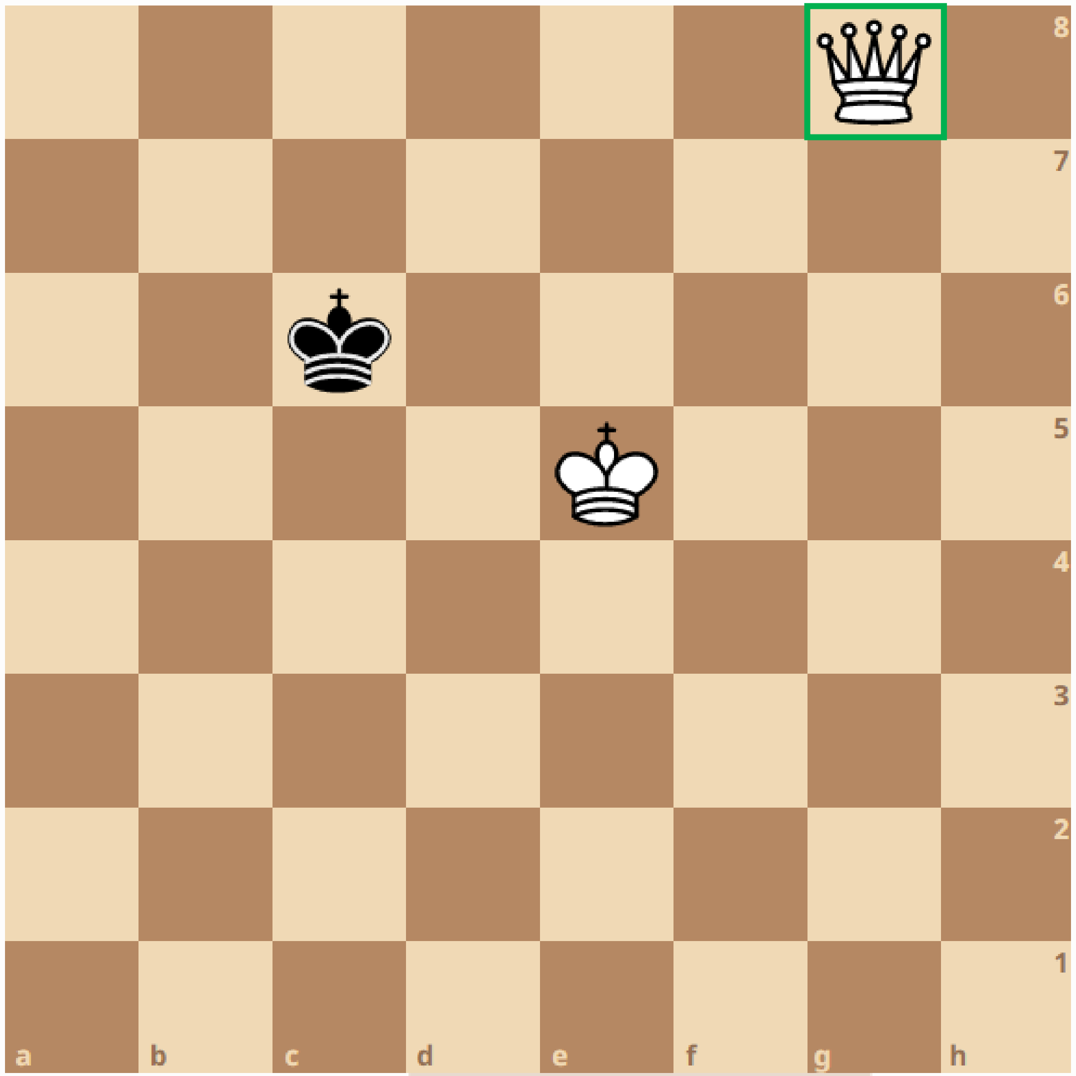
- S(c)arsa-λ
- ε-greedy (Q,E,N)
- Rewards
- Weights
- Results

PROBLEM SET (1)



- **GOAL: Win the game against the computer in a limited number of moves**
- A **game** is a random walk between different states (a state is a location matrix)
  - p = 1: probability of moving from a state to another, having picked a certain action (**deterministic**)
  - To each action there is an associated reward
- A game can be approximated by a **Markov chain**
  - The sequence of moves that led to the current location matrix is irrelevant: the only relevant aspect is performing the **best move from current position**.

PROBLEM SET (2)



- **Winning position for white**: if white agent plays a correct sequence of moves, it will win the game whatever the black does.

- **Easy to lose**: if you let the black king get close enough to the white pawn.

- **How to win**: reach the g8 square with the pawn.

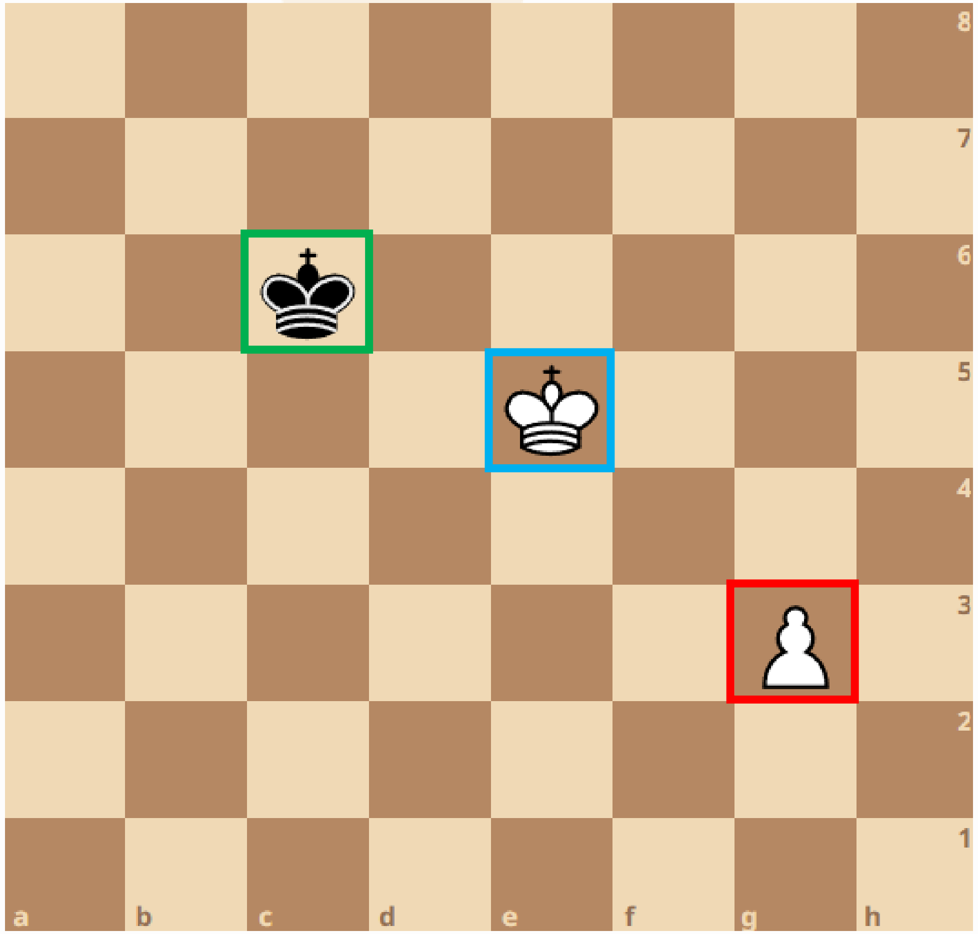
- **How to lose**:
  - let the pawn be captured.
  - not being able to win in 20 moves.

Linear algebraic representation (1)

Environment modelling from scratch with linear algebra

- **Location matrix L:** storing the coordinates of the pieces on the chessboard

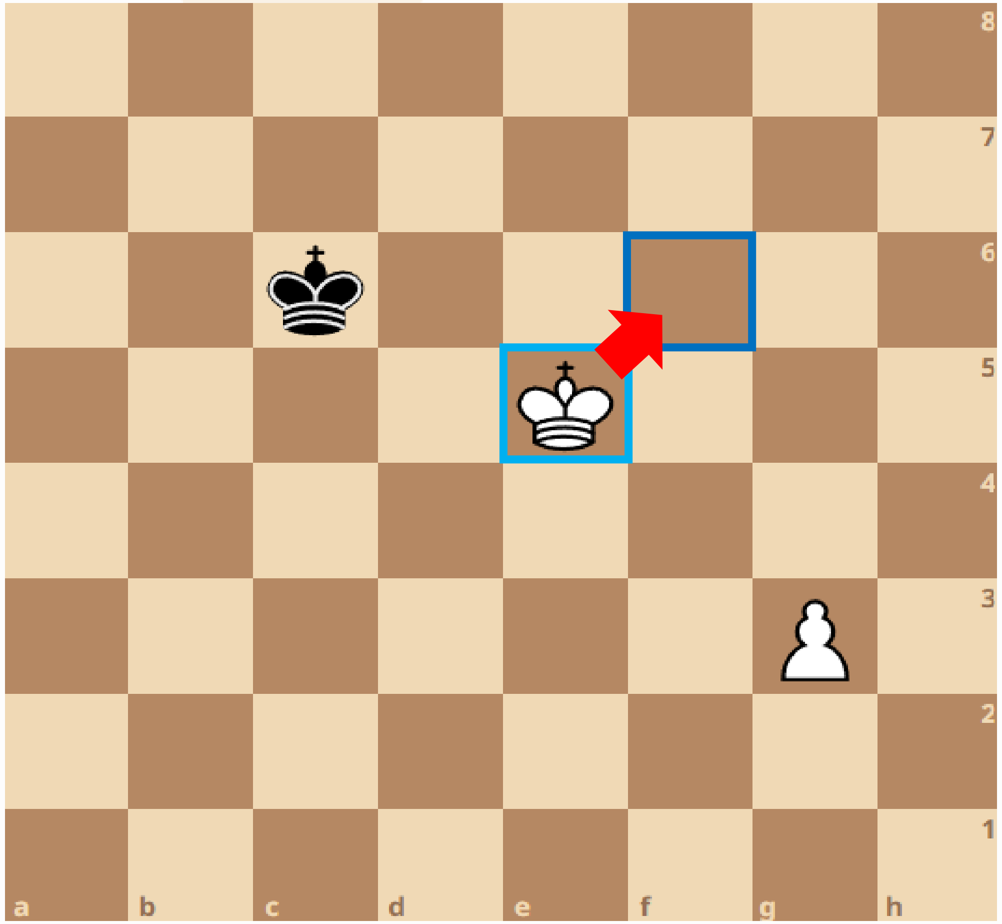
$$\begin{array}{l} \text{Rows} \\ \text{Columns} \end{array} \begin{bmatrix} 2 & 4 & 5 \\ 6 & 4 & 2 \end{bmatrix}$$



Linear algebraic representation (2)

- **Selection Matrix S:** $[1\ 0\ 0]$, $[0\ 1\ 0]$, $[0\ 0\ 1]$, used to select the piece from the location matrix
- **Moves:** dictionaries
  - Keys: string, name of the move (e.g. pfwd, kbl)
  - Values: vector e.g. $\begin{bmatrix}1\\0\end{bmatrix}, \begin{bmatrix}-1\\-1\end{bmatrix}$

$$\begin{bmatrix}2 & \boxed{\begin{matrix}4\\4\end{matrix}} & 5\\6 & & 2\end{bmatrix} + \underbrace{[0\ 1\ 0]}_{\text{white king selection}} \times \boxed{\begin{bmatrix}1\\1\end{bmatrix}} = \begin{bmatrix}2 & \boxed{\begin{matrix}5\\5\end{matrix}} & 5\\6 & & 2\end{bmatrix}$$



States

- Function to retrieve all the possible future locations after having performed an action (**legal_state**).

- **Algorithm**:
  - Store location matrices as **integers** to save space and make it easier to check conditions

  - **Reduced states**: location matrix without black king, i.e. black king is considered part of the environment (influencing possible future states)

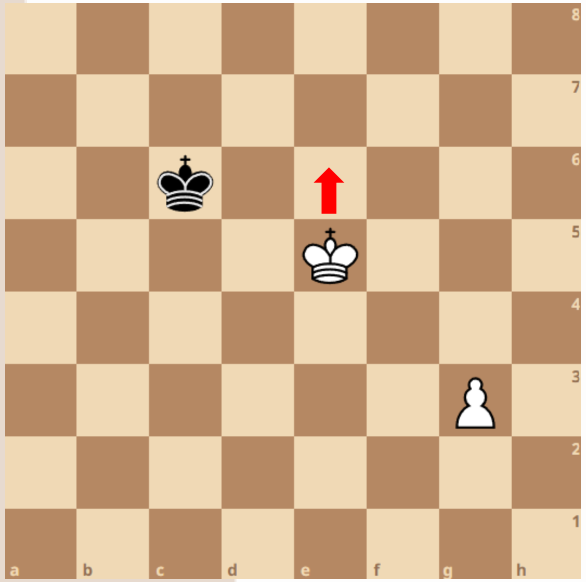
$$\underbrace{\begin{bmatrix} 2 & 4 & 5 \\ 6 & 4 & 2 \end{bmatrix} \longrightarrow 245642}_{\textbf{array\_to\_int}}$$

$$\underbrace{\begin{bmatrix} 2 & 4 & \boxed{5} \\ 6 & 4 & \boxed{2} \end{bmatrix} \longrightarrow \begin{bmatrix} 2 & 4 \\ 6 & 4 \end{bmatrix}}_{\textbf{reduce\_array}} \underbrace{\longrightarrow 2464}_{\textbf{array\_to\_int}}$$

Actions

- Function to retrieve all the possible moves (*actions*) for the white agent starting from the current location matrix (**legal_move**).
  - Encoding pieces as numbers on the chessboard
  - Matrix/vector addition

- **Algorithm**:
  - Store actions as **pairs of reduced states** in **integer form.**



'kfwd' ↔ (2464, 2564)

The Opponent: Stockfish

- **Black king:** controlled by *Stockfish 15.1*,
  one of the strongest chess engines with an estimated ELO rating of 3620.

- The highest ELO reached by a human is 2882, in 2014, by Magnus Carlsen.

- **White pieces:** moved by our algorithm.

- The agent decides at each turn whether to move the white king or the pawn.

- **Why SARSA-ʎ?**
  - **On-policy** vs off-policy: innumerable possible states and actions per state makes Q-learning a waste of time and resources

- **Change policy** as the training goes on (vs observing several policies that may be suboptimal)
  - After a certain number of games the agent will perform the best move given its position

S(c)arsa-ʎ

Input: Initial state (Starting location matrix)
- Set Q, N, E = 0

For Games:
- Draw an initial action (epsilon greedy)
- **For** Turns:
  - White move
  - Update N
  - Choose the move for the next turn (epsilon greedy)
  - Black move
- Assign the rewards
- Compute Delta
- Update Q, E

S(c)arsa-λ: Q, E, N

- **Q**: storing
  - State in form of an integer (e.g. 245642)
  - Action as a pair of reduced states (e.g. (2464, 3464))
  - Value (e.g. 10)
  - Updated at the **end of each game**, after computing delta

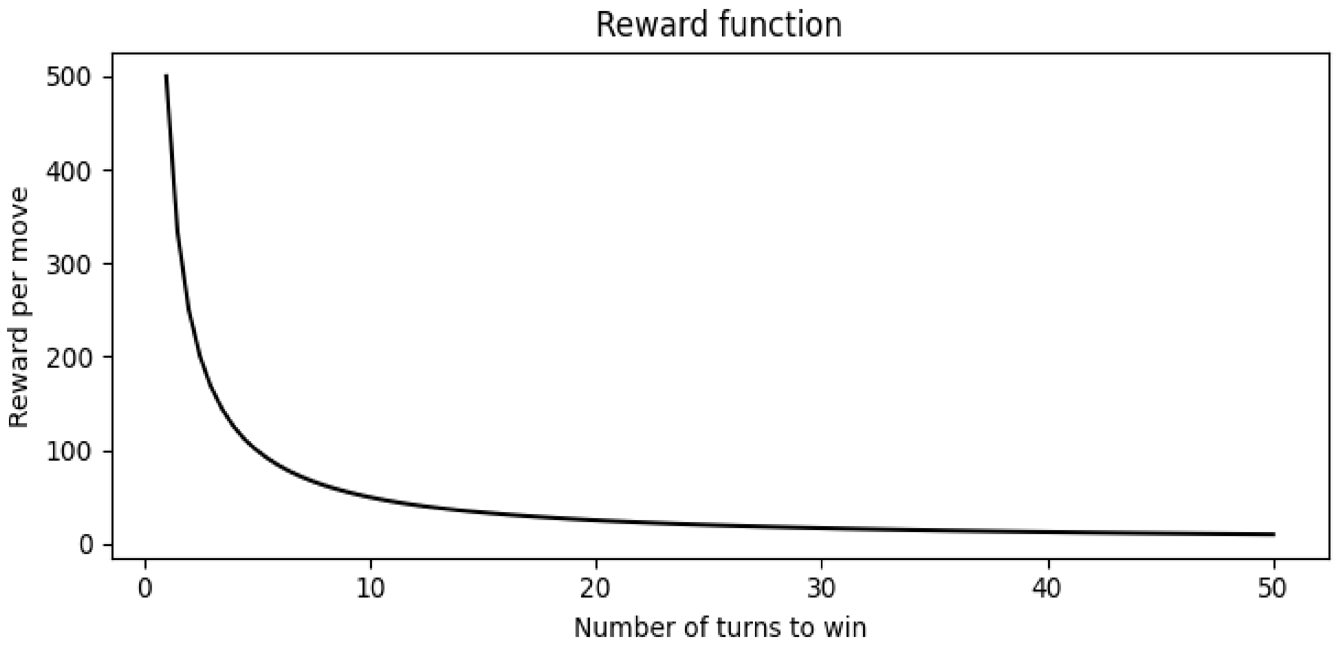
- **E (eligibility trace)**: same content as Q, but different update rule
  - Updated at the end of each game

- **N**: storing
  - Reduced state in form of integer (e.g. 2464)
  - Number of visits to that state (once again, the black king is considered each time as part of the environment)
  - Updated **after every white move**

S(c)arsa-λ: ε-greedy

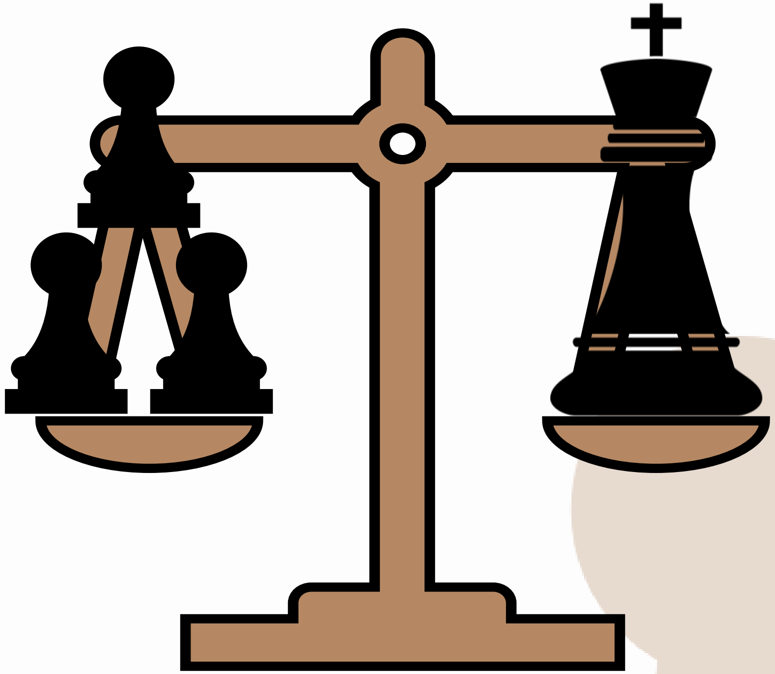
- **Next move:**
  - Random number generation x ϵ [0,1]
    - If x < $\frac{1}{N_i}$: pick a weighted random move, $N_i = \#\text{visits to state i}$
    - Else: pick the move that maximises the value of Q in that state
  - At the **beginning** the white agent **explores every move**, eventually getting a **negative reward**
    - Each reward in Q set to 0
    - For each state #visits is set to 1
  - After several games $\boldsymbol{P}\left(x < \frac{1}{N_i}\right) < \varepsilon$: if the agent **already visited the state** many times, it will **choose the move maximising its reward** in that state (greedy).

S(c)arsa-λ: Rewards



- **Rewards given to the single moves, at the end of each game.**
- **Positive Rewards** if the agent wins the game.
- **Higher rewards** if the agent wins in a shorter number of moves, as shown in the figure.
- **Negative rewards** if the Agent loses the game.

S(c)arsa-λ: Weights



- When the algorithm explores, we **weight the probability** of choosing the piece to move.
- At each move:
  - 50% of chance of moving the pawn
  - **fairly** divide the remaing 50% among all the king possible moves.

- Example:
  - Pawn: 1 possible move
  - King: 4 possible moves

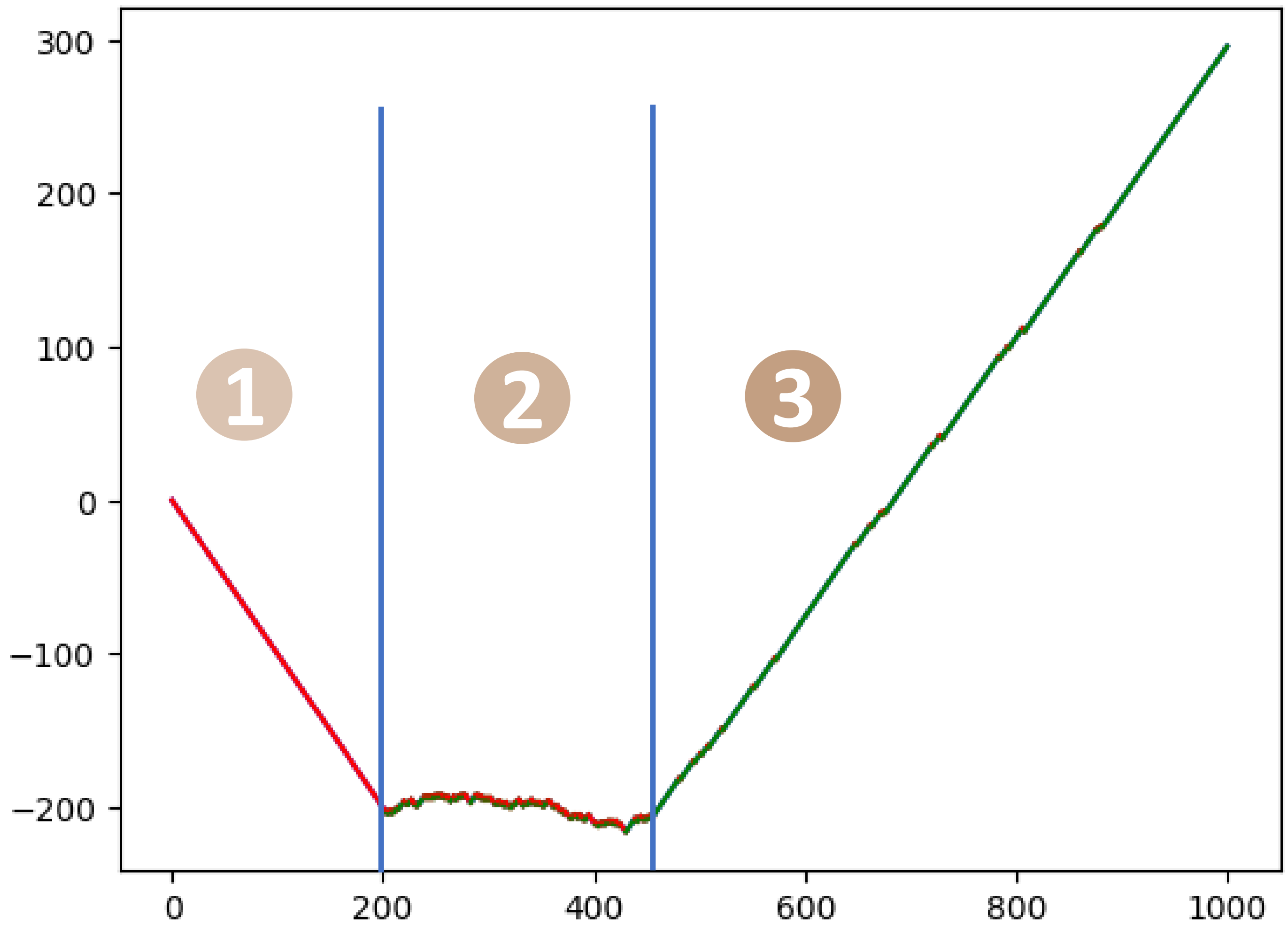
Weights vector →

$$[\underbrace{0.5}_{\text{Pawn move probability}} ; \underbrace{0.125 ; 0.125 ; 0.125 ; 0.125}_{\text{King moves probabilities}}]$$

Results [1]

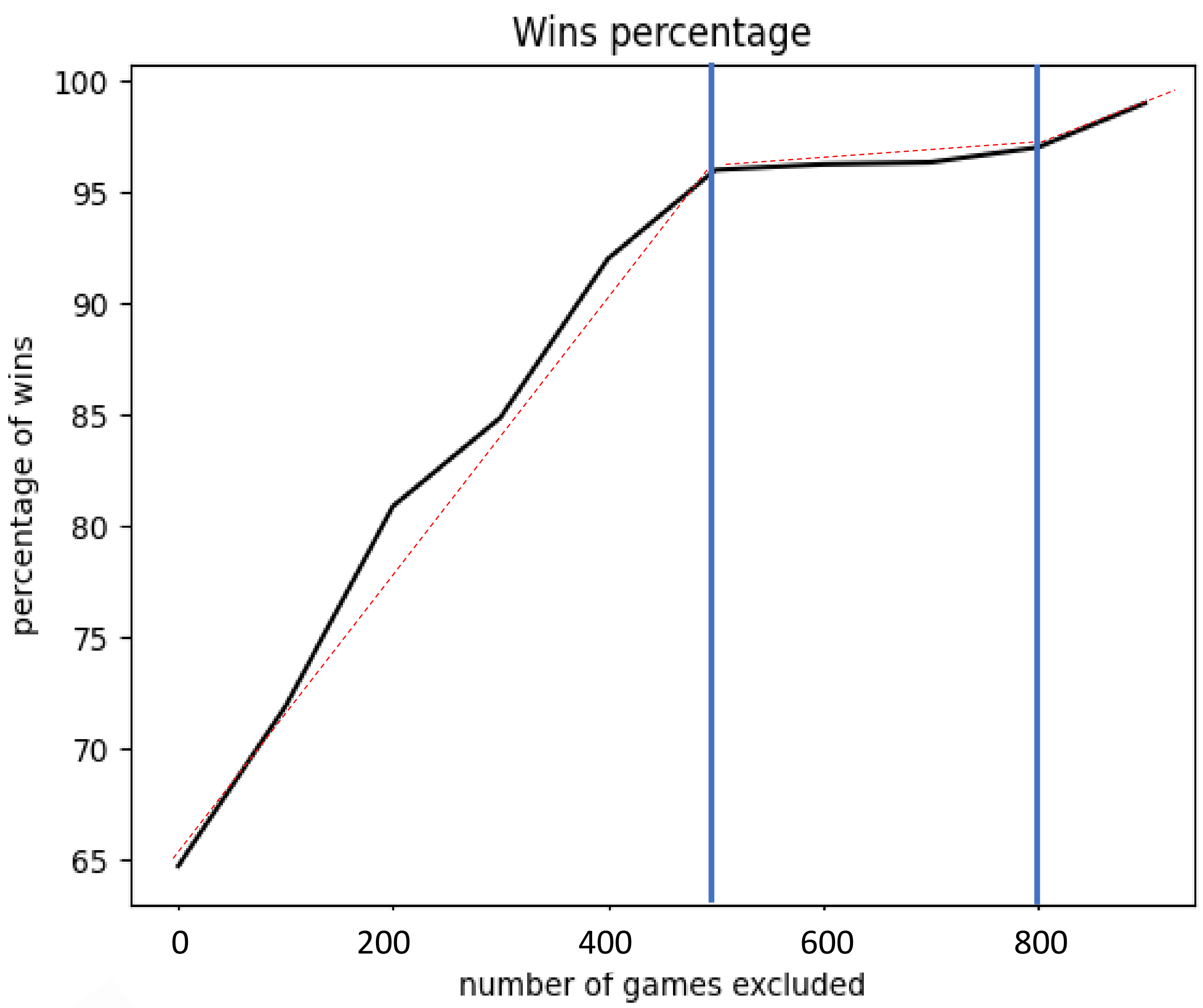
- The graph represent the score obtained:
  - + 1 if white agent wins the game
  - – 1 if white agent loses the game

- 3 main phases:
  - **First**: The algorithm is **totally explorative,** no winning sequence discovered.
  - **Second**: The algorithm starts to learn, but still **explores new paths**, that cause it to lose.
  - **Third**: The algorithm becomes (almost) **totally greedy**: it has learnt how to respond to the majority of the possible black moves.



Note that since the explorative phase has a **random component**, repeating the same experiment twice could provide different results (due to potentially different black response). Nonetheless, in the long run the algorithm will converge in any case

Results [2]



What is the slope of the learning?

- Taking into account **all the games**, the algorithm has won **64,38%** of the games
- Let's check **how the win rate changes**, (iteratively excluding 100 games)
  - The graph shows an **almost linear growth** in the win rate for the first 500 games
  - From game 500 to game 800 the win rate is stable at **95%.**
  - After game 800 it starts increasing again (**99.01%** of win rate in the last 100 games).


PLAY
AGAINST
US!


THANKS
FOR THE
ATTENTION

Questions?