Prof. Eleni Vasilaki

Reinforcement Learning Assignment

March 6, 2023

1. Overview

In this individual assignment, you will implement a deep reinforcement learning agent that learns to play chess. Please submit a four pages report containing a GitHub link to your code.

2. Assignment description

The task consists of a chessboard 4x4. Three pieces, **1x King (1)**, **1x Queen (2)** and **1x Opponent's King (3)**, are placed in a random location of the board. In the initial positions, the pieces are not causing any threats.
Assuming the role of player 1 (white), you aim to checkmate the Opponent's King (player 2, black). The game progresses as follows:

1. Player 1 moves the King or the Queen, aiming checkmate.
2. Player 2 randomly moves the Opponent's King to a safe position (i.e. where Player 1's King or Queen doesn't threaten it).
3. Repeat the above sequence until:
   - **Checkmate**. Player 2 plays. Its King is threatened. There are no squares that the King can move.
   - **Draw**. Player 2 plays. Its King is not threatened. There are no squares that the King can move.

Please note that the King cannot move to a square where he is threatened.

Your task is implementing a deep neural network capable of learning to perform checkmate. The states' features are denoted by a vector $\mathbf{s}(t)$, whose dimensionality is $3N^2+10$, where $N \times N$ is the board size, and 3 is the number of the pieces involved in this task. Thus, the position of each piece is decoded by a binary matrix C_{ij} (the chessboard), where $c_{ij} = 1$ if that particular piece is in the position ij and $c_{ij} = 0$ otherwise. The $+10$ term in the dimension of $\mathbf{s}(t)$ contains: (i) the degree of freedom of the opponent King (eight options in 1-hot-encoding), i.e. the number of available squares where the Opponent's King is allowed to move, and (ii) whether the Opponent's King is threatened or not (two options in 1-hot-encoding).

We provide you with documented code implementing the environment (in Python). You only need to implement the updated rules for the backpropagation and TD algorithms

(please see appendix A). You can use (modified where appropriate) code from earlier Labs you developed independently or from the Lab solutions we provided. You are also allowed to use libraries, subject to describe them in a short section of the report's appendix. But you must interface your code with the environment we provide you.
Your specific tasks for this assignment are:

1. Describe the algorithms Q-learning and SARSA within the context of Deep Reinforcement Learning. Explain the differences and the possible advantages/disadvantages between the two methods. No coding is needed to reply to this question.

2. We provide you with a template code for the chess game. Fill the gaps in the program file *chess* implementing either Deep Q-Learning or SARSA; detailed instructions are available inside the file. We provide indicative parameter values. Produce two plots that show the reward per game and the number of moves per game vs training time. The plots will be noisy. Use an exponential moving average.

3. Change the discount factor γ and the speed β of the decaying trend of ϵ (for a definition of β please see code). Explain these parameters and how they should affect your results (you can reply to this question without running simulations). Analyse your results in the context of the varying parameters.

4. Implement another algorithm (it could be SARSA if you chose before Q-Learning and vice versa, but you can also make a different choice). Discuss how you expect them to differ. Compare the new results with your previous results.

3. Report

Your report should adhere to scientific standards, i.e. a journal paper-like format with sections: introduction, methods, results, and conclusions. We recommend you search the web for relevant journal papers and mimic their structure. For instance, you can use the IEEE templates `https://www.ieee.org/conferences/publishing/templates.html`. Explain and justify your results. Figures should have captions and legends. Plots require error bars (where appropriate). The individual reports should **NOT exceed four pages** (excluding Appendix and Cover Page). We recommend a two-column format. The minimum font size is 11pt. Margins should be reasonable. Kindly note that we consider the readability and clarity of your document. Your report should include the following:

1. Reply to the questions.

2. An Appendix with snippets of your code referring to the algorithm implementations, with comments/explanations, where you would like to highlight any novelties you have introduced (i.e. your ideas that go beyond the Lab code).

3. A description of how to reproduce your results, see also "Important Note".

Important Note. Please make sure that your results are reproducible. You can, for instance, initialise a random seed to ensure you always get the same results. Together with the assignment, please provide code well commented on. Provide sufficient information so that we can quickly test your results. If your results are not reproducible by your code, the assessment cannot receive total points. If you do not provide your code, the submission is incomplete.

4. Suggested language

The recommended programming language is Python.

5. Marking

To maximise your mark, please make sure you explain your model. Please provide evidence of your results (e.g., plots with captions, scientific arguments). Figures should be combined wherever appropriate to show a clear message. Any original additions you have employed should be highlighted and well explained.

6. Submission

The **deadline** is **Friday, 19th May 2023, 23:59.** (in PDF format). Your report should include a link to your GitHub project of the assignment. Please keep your GitHub repository private, and add the Lecturer and Teaching assistants as collaborators so we can assess your work. The GitHub usernames are: *eleni-vasilaki*, *ChaoHan-UoS*, *mtrkhan854* and *iLackImagination*.

7. Plagiarism, and Collusion

You are permitted to use Python/Matlab code developed for the lab as a basis for the code you produce for this assignment or other code that inspired you with **appropriate acknowledgement**. Any sources of information that you use should be cited. You may add a section in the appendix for additional information. Libraries and toolboxes are allowed, but if you choose to implement your code based on the lab material, it will give you a **20% bonus** on your mark. All marks are upper bounded by 100 and subject to scale according to the departmental board.

You may discuss this assignment with other students, but the work you submit must be your own. Do not share any code or text you write with other students. Please refer to the guidance on “Collaborative work, plagiarism and collusion” in the Undergraduate or MSc Handbook.

A. Chess

Chess is a two-player strategy game on a 64 square board (chessboard) arranged in an 8x8 grid. Each player is given 16 pieces falling under six types, and each type has a unique set of moves. The game aims to **checkmate** the opponent's king by placing it into a position where it is threatened by a piece and cannot perform any action to escape the threat. Another way that a chess game can end is by **draw**, which happens if the remaining pieces in the chessboard are unable to perform a checkmate or the king *(a)* is not threatened, *(b)* is the only piece that can perform an action and *(c)* any action will cause him a threat. More information on chess and how each piece moves can be found online (e.g. here); for your assignment, you will use a simplified version of the chess with only three total pieces on the chessboard board.

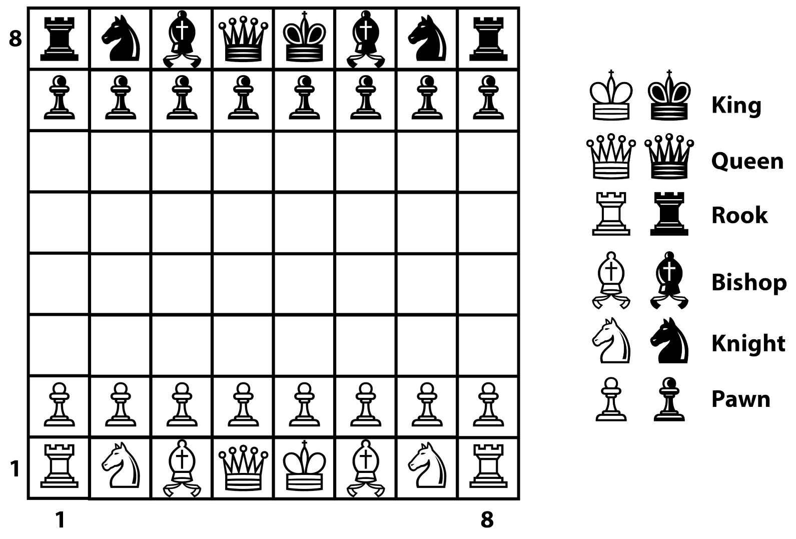

Figure 1: A game of chess and the name of each piece.

Learning to checkmate only with King and Queen

For your assignment, you will be given a reduced chessboard with a **4x4 grid** and three pieces that are generated in random locations: **1x King (1)**, **1x Queen (2)** and **1x Opponent's King (3)**. Your task is to use reinforcement learning to teach your pieces (White, King and Queen) to perform checkmate to the Opponent's King (Black). *For illustrating purposes in these examples, we will use an 8x8 grid.*

General movement rules

- The King can move one block at a time.
- The Queen can move in any one straight direction: forward, backward, right, left, or diagonally, as far as possible, as long as she does not move through any of the other pieces on the board.
- None of the pieces is allowed to exit the board.

Refer also to figure 3.

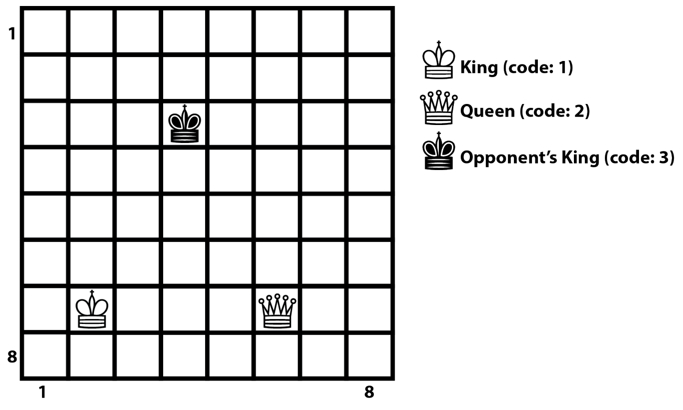

Figure 2: Simplified chess.

Hard-coded movement rules

To simplify the game even more and make the learning process faster, some constraints have been added:

- The King will never move to a location that will cause it to be threatened by the Opponent's King.
- The Queen will never move to a location that will cause it to be threatened by the Opponent's King except if the King is nearby to protect.
- The Opponent's King will always pick a random action that will not cause him to be threatened. If such action is not available, then:
  - If it is already threatened, it's a **checkmate**.
  - If it is not already threatened, it's a **draw**.

Refer also to figure 4.

End of game conditions and example of a complete game

The game can end either in a **checkmate**, where the Opponent's King cannot move and is threatened (refer to figures 5(a) and 6) or in a **draw**, where the Opponent's King cannot move but it is not threatened (refer to figures 5(b) and 7).

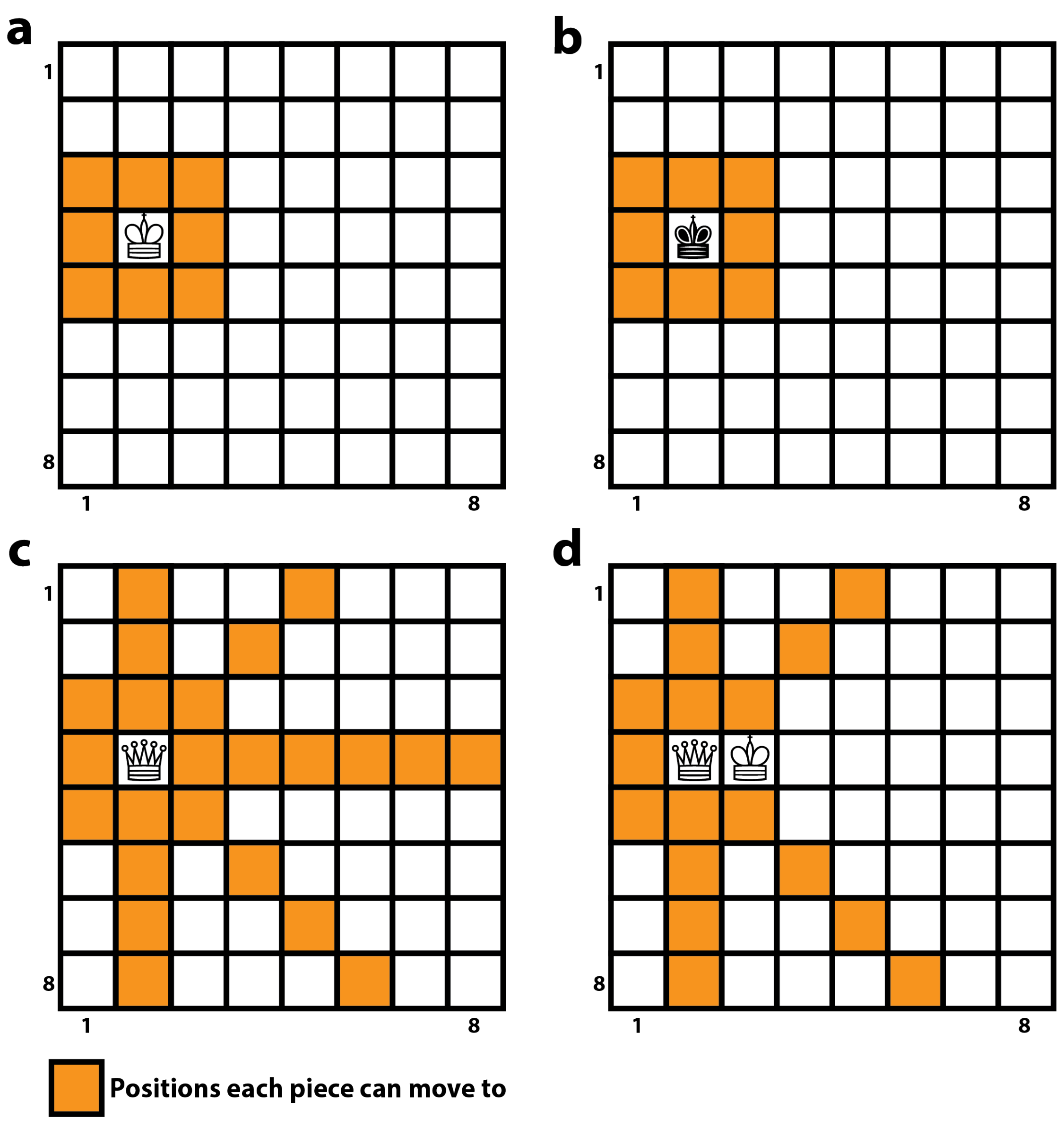

Figure 3: Movement pattern of each piece. **(a)** King, **(b)** Opponent's King, **(c)** Queen, **(d)** Queen blocked by King.

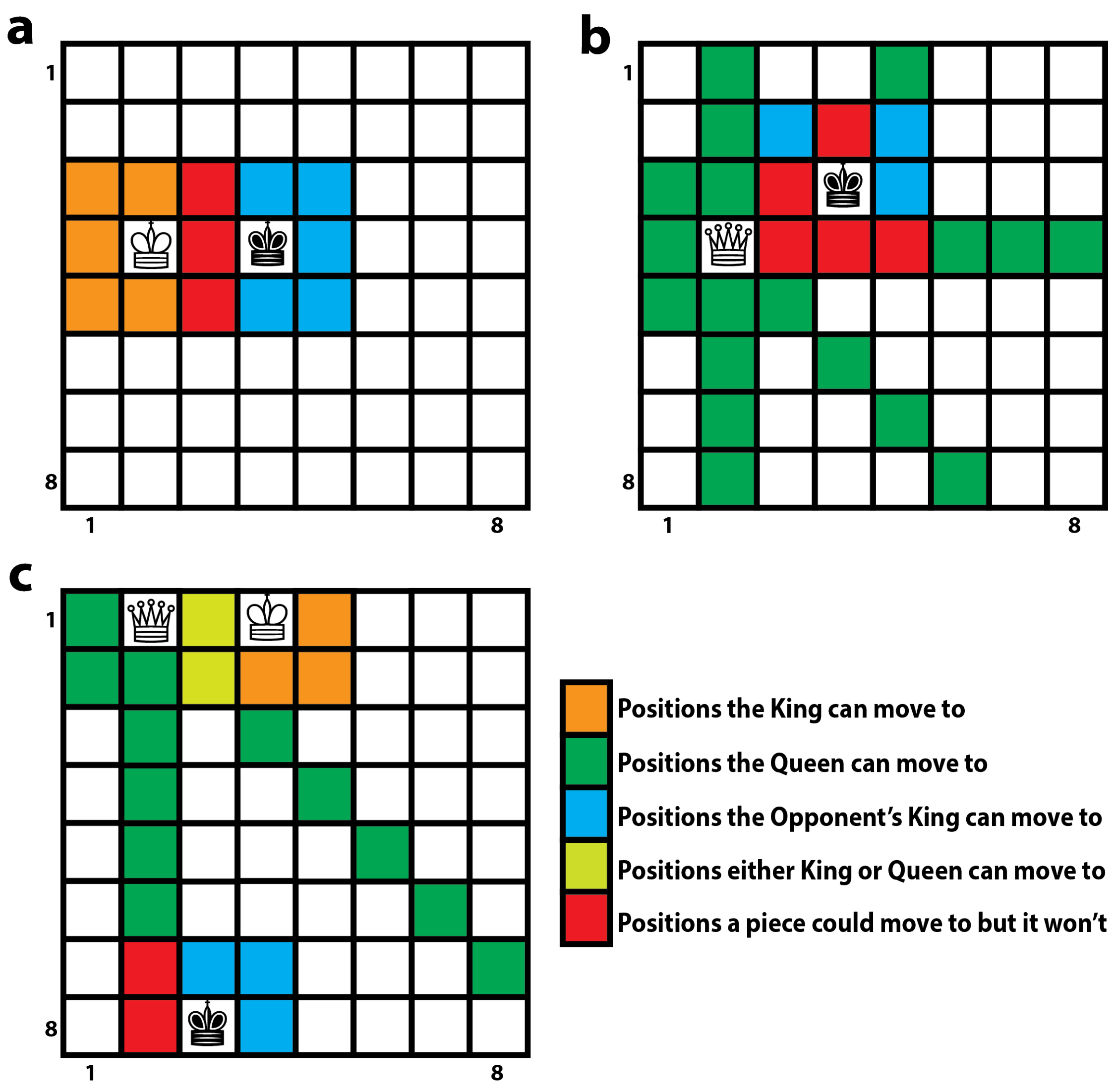
a
b
c
1
8
1
8
Positions the King can move to
Positions the Queen can move to
Positions the Opponent's King can move to
Positions either King or Queen can move to
Positions a piece could move to but it won't

Figure 4: Constrained movement pattern of each piece. **(a)** The King or the Opponent's King will not select to move in the blocks marked with red because, on the next turn, one of them will be captured. **(b)** The Queen or the Opponent's King will not select to move in the blocks marked with red because, on the next turn, one of them will be captured. **(c)** The Queen or the Opponent's King will not select to move in the blocks marked with red because, on the next turn, one of them will be captured.

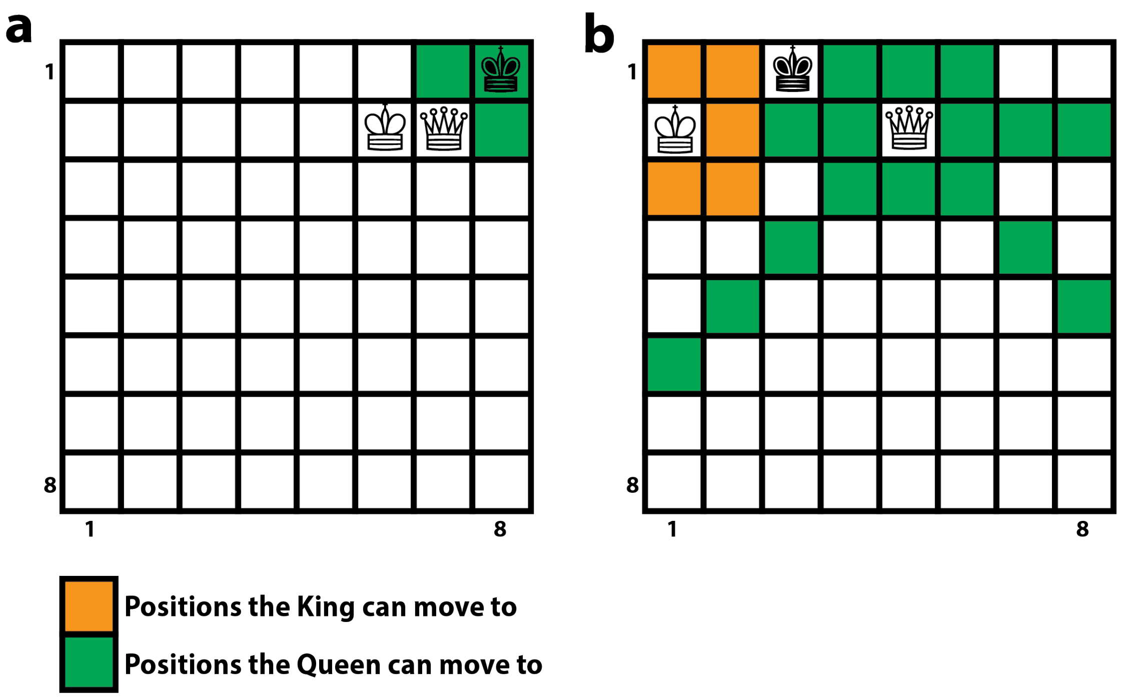
a
b
1
8
1
8
1
8
1
8
Positions the King can move to
Positions the Queen can move to

Figure 5: Ending of a game. **(a)** It is the Opponent's King's turn, the Queen threatens the Opponent's King, and it cannot perform any action to escape the threat. It also cannot capture the Queen because then it will be captured by the King. It is a **checkmate**. **(b)** It is the Opponent's King turn, it is not threatened, but it cannot move anywhere since all the available positions will cause it to be threatened. It is a **draw**.

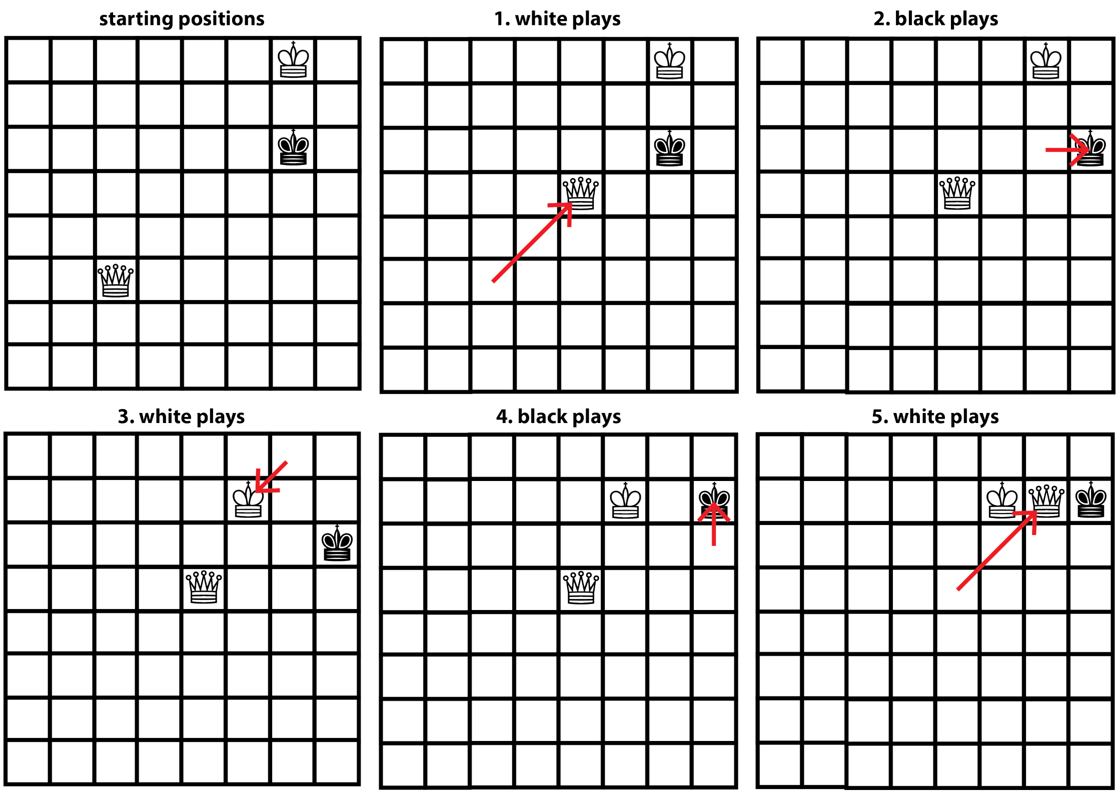

Figure 6: Game ends with a checkmate. The pieces are randomly placed on the chessboard without causing any threats. **(1)** Player 1 (white) plays first and moves the Queen. **(2)** Player 2 (black) moves the Opponent's King to a random position, given that it will not be threatened. **(3)** Player 1 (white) moves the King. **(4)** Player 2 (black) moves the Opponent's King. **(5)** Player 1 (white) moves the Queen. The game ends in a **checkmate** because the Queen threatens the Opponent's King, and it cannot perform any action to escape the threat.

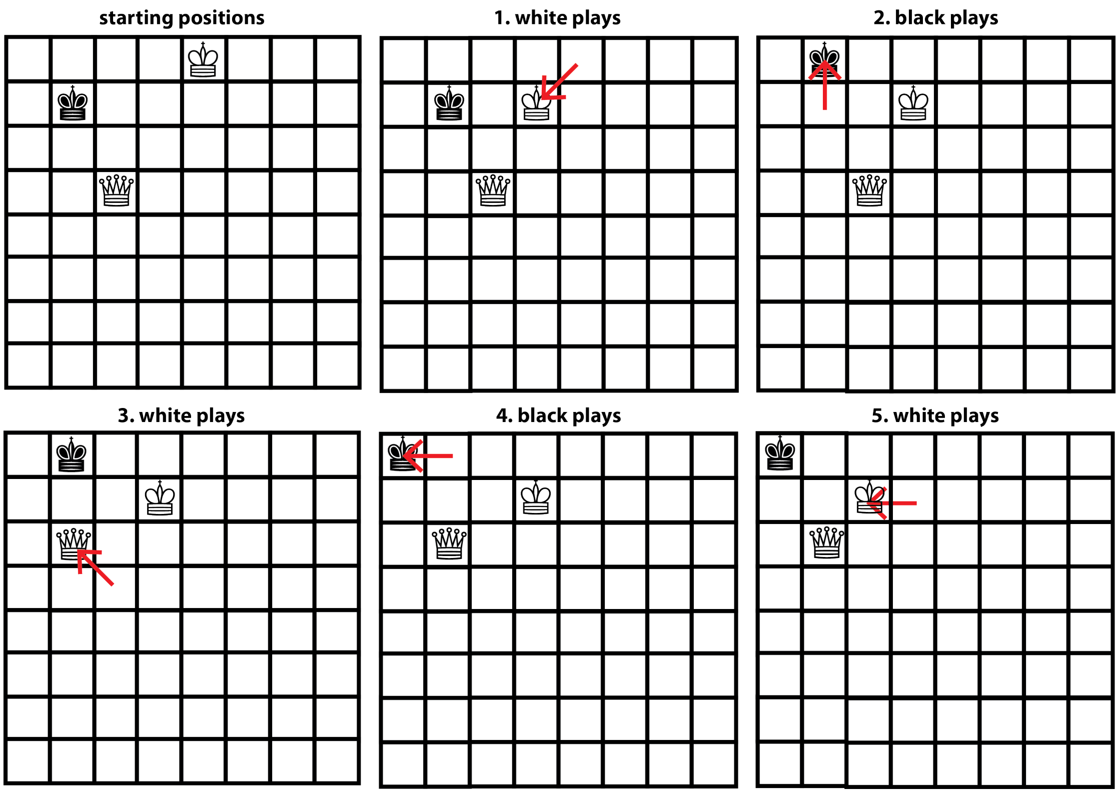

Figure 7: Game ends with a draw. The pieces are randomly placed on the chessboard without causing any threats. **(1)** Player 1 (white) plays first and moves the King. **(2)** Player 2 (black) moves the Opponent's King to a random position, given that it will not be threatened. **(3)** Player 1 (white) moves the Queen. **(4)** Player 2 (black) moves the Opponent's King. **(5)** Player 1 (white) moves the King. The game ends in a **draw** because the Opponent's King is not threatened and cannot perform any action since every available action will cause it to be threatened.