An Interactive Collaborative Robotic System to Play Italian Checkers

An interactive collaborative robotic system to play Italian checkers

Giuliano Fabris, Lorenzo Scalera, and Alessandro Gasparetto

University of Udine, 33100 Udine, Italy,
{giuliano.fabris,lorenzo.scalera,alessandro.gasparetto}@uniud.it

Abstract. An interactive collaborative robotic system to play Italian checkers is presented in this paper. The system detects the state of the game using a camera, calculates the optimal move using a developed decision-making algorithm, and executes pick-and-place tasks to physically move the pieces on the board. The developed system is implemented in a real-world setup using a Franka Emika arm as pieces manipulator. The experimental results show the feasibility of proposed approach.

Keywords: collaborative robotics, computer vision, human-robot interaction, manipulation, artificial intelligence.

1 Introduction

Collaborative robotics describes how robots are integrated into modern manufacturing processes in a way that permits a safe human-robot interaction [1,2]. Nowadays, collaborative robotics founds applications not only in industry, but also in the fields of art and entertainment. Examples are given by robotic systems that can safely interact with humans while playing music [3], playing board games [4], or painting [5,6]. Among those, several robotic applications have been developed over the years to interactively play chess and checkers.

Actually, checkers and chess do not necessarily need a physical implementation of a robotic player (a GUI interface can be sufficient). However, designing efficient and competitive decision-making algorithms for such games is not a simple task as reported in [7], where the develop of a checkers playing GUI based on a *minimax* algorithm is presented, i.e., a game tree search algorithm in which the calculation time increases exponentially with the tree depth. In [8], the authors describe a hybrid tree search algorithm on parallel CPU and GPU with the aim of decreasing the calculation time. An alternative approach to select the best move among all possible ones is to use a neural network, as shown in [9].

Nevertheless, it is proved that a physical implementation of a robotic system improves the user experience and the game attractiveness to human players, making the game more realistic [10,11]. Moreover, manipulating non-instrumented arbitrary-shaped pieces on a board in changing environmental conditions is a complex task itself. The main challenge is to recognize the position of the pieces, mainly through computer vision, which is quite sensitive to lighting conditions.

To reduce this problem, the pieces, normally black and white like the squares, are usually colored in order to increase the contrast with squares [8,12,13]. This helps the movement detection, needed for knowing if the human player has made a move, for which it is sufficient to observe a change in the board image [14].

The recognition of the board, necessary in case the relative position between robot and board changes, is less problematic since many tested and robust algorithms are available to accomplish this task, like the Harris corner detection [15,16], the Canny edge detection, and the Hough Line Transform [17,18]. Furthermore, these vision algorithms require a proper calibration procedure that has to be done before starting to play. In order to make the board game design easier and fully independent from lighting conditions, some authors propose using an array of Hall sensors for detecting pieces on the board [9].

The other challenge of a physical implementation of the board games is controlling the robot in order to execute actions like moving pieces and removing them in case of capture. This also depends on the accuracy of the vision system that has to identify as precisely as possible the pieces position, allowing the robot to reach the correct target. Initially, for these applications robotic arms are mainly used, as described in [14,16,18]. Furthermore, the possibility of making the robot interact with the human player through facial expressions in order to enhance user experience has been also explored. An example is given by Baxter, a two-armed robot provided with a display that shows facial expressions related to its game state [13,17]. Recently, the researchers in [12,15] tried to make these applications even more realistic using alternative humanoid robots.

At University of Udine (Italy) we developed a collaborative robotics system for interactively play Italian checkers based on a Franka Emika manipulator and a vision system. The system detects the state of the game using a camera, calculates the optimal move using a developed decision-making algorithm, and executes pick-and-place tasks to physically move the pieces on the board.

The rules of Italian checkers [19] are briefly recalled in the following. The board has a size of 8 by 8 squares, alternately light and dark (usually white and black), playing on the dark ones, and players both start with 12 pieces, one with light ones and the other with dark ones. The player with light pieces starts to move. Pieces can be *men* or *kings*: each player starts with only men and they can become kings reaching the other side of the board. The difference between the 2 type of pieces is in their movements: men move one square diagonally forward, instead kings move one square diagonally both forward and backward. When a man (or a king) encounters a man (or a king) of a different color, with a free square behind it, it must capture it. Men can capture only men, while kings can capture both men and kings, and it is mandatory to capture as much pieces as possible in a single move. Finally, the player that captures or blocks all the opponent's pieces wins. If none of the players succeeds, the game is even.

This paper is organized as follows: Sect. 2 describes the algorithm for playing Italian checkers, Sect. 3 presents the experimental setup used for this system, and Sect. 4 shows the experimental results of this work. Finally, the conclusions and possible future improvements are given in Sect. 5.

2 The algorithm for playing Italian checkers

In this section, a general description of the individual components of the algorithm used to analyze the board and to decide the next move is presented. We first illustrate the computer vision approach used in this work, then the algorithm, based on AI, developed for computing the optimal moves.

2.1 The recognition algorithm

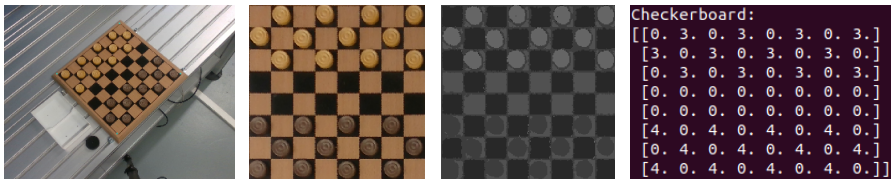
The first part of the system consists of the computer vision developed algorithm, which receives the image of the board (Fig. 1a) as input and identifies the positions and types of pieces. A function called *homography* is initially applied to the input image in order to switch from the plane of the image to that of the board and obtain the image of the board only (Fig. 1b), operation that requires the contribution of the operator who has to manually indicate the position of the board corners. For this operation it is not required a high precision because, even if the user does not indicate corners perfectly, this is corrected by taking advantage of the regularity of squares divisions.

After this, the setup of colors and types is performed, assuming that there are only four kinds of objects inside the board: white/black pieces and white/black squares. Due to this assumption, a very important aspect to consider is the illumination of the board, trying to avoid shadows and reflections in order not to interfere with the color identification. This problem is reduced by increasing the image resolution. All the pixels are organized in a list where each entry contains an RGB value, which is analyzed obtaining another list where each original RGB value is substituted with the color closest to it among the four predominant ones, identified with a value from 1 to 4. Then, to get a black and white image a shade of gray based on the values present in the list obtained previously is assigned at each pixel (Fig. 1c).

However, the four colors are initialized randomly, therefore not necessarily a lighter shade of gray represents white and a darker black, so it is necessary to go through a setup of the types. To do this, the user is asked to indicate the position in order of a white square, black square, white piece and black piece, so the program knows the location of an element for type of the four present in the board. From every element selected, an image of its square is cut out and analyzed, obtaining a histogram that represents how many times each color appears. From this histogram, the color with the highest number of occurrences is extracted. In this way it is possible to associate the real color indicated by the user to each randomly initialized color class.

Considering the colors of the pieces one at a time and applying to the respective pixels filters that eliminate the imperfections around the pieces, the developed computer vision algorithm is able to recognize the real center of the pieces with sufficient precision, saving their position through x and y coordinates in the system of the board normalized between 0 and 1, which is also used for the control of the robot.

Finally, the construction of the matrix containing positions and types of the pieces can be done, calculating for each piece the distance between its center and the center of closer black squares and associating piece to closest black square. Because the camera can only detect the color of the piece but not its type (man or king), a comparison between current and previous board images is required to determine it, knowing that if a man reaches the opposite side of the board, it becomes a king. After that, the respective identification numbers are assigned (0 for empty square, 3 for white man, 4 for black man, 5 for white king and 6 for black king) and the final matrix is created as in Fig. 1d. The flowchart shown in Fig. 2 summarizes the main steps of the described algorithm.



(a) (b) (c) (d)

Fig. 1: Image captured with corners indication (a), *homography* (b), black and white image (c), and final matrix (d).

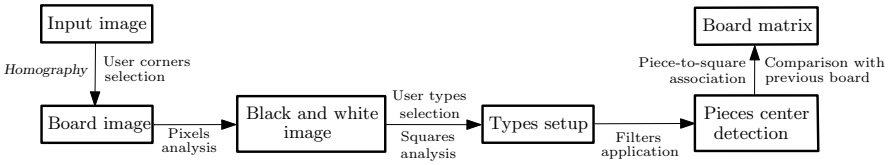


Fig. 2: Flowchart of computer vision software.

2.2 The gaming algorithm

After obtaining the matrix that represents the board, it is necessary to choose the move to perform, which must respect Italian checkers rules described in Sect. 1. The gaming algorithm, based on AI principles, has been developed as a *minimax* algorithm, a recursive algorithm often used in decision-making and zero-sum games (as Italian checkers), which allows the artificial player to choose the best move against the human player who is supposed to play optimally himself [7,20]. The algorithm implemented in this work is based on a recursive optimization of an evaluation function based on rewards and penalties determined by the game strategy.

The evaluation function (1) is applied to the considered board configuration and associates it with a score (i.e., the result of the evaluation function) that

depends on the number and position of own and opponent's men (m and om) and kings (k and ok), for example men close to becoming kings (i.e., with high m_{line} and om_{line}) or that are protected ($m_{protect}$ and $om_{protect}$) increase the score, positive for those of the robot and negative for those of the human. Changing weights (w_i), the game strategy can be changed. Among every possible move, only the five moves that lead to a board configuration with a higher score are considered to limit the tree breadth and reduce the calculation time, assuming that the others do not lead to favorable situations. For each considered move, the algorithm is repeated from the point of view of the human player, calculating its possible and better moves. By iteratively repeating this procedure up to the chosen depth, a search tree is built. If the search depth is set greater, artificial player performance increases, but at the expense of the calculation time.

$$\begin{aligned} \text{score} = {} & w_{man} \cdot (m - om) + w_{king} \cdot (k - ok) + \\ & + w_{protect} \cdot (m_{protect} - om_{protect}) + \\ & + w_{men\ position} \cdot \sum_{men} (m_{line}^2 - om_{line}^2) \end{aligned} \tag{1}$$

The developed gaming algorithm searches the move corresponding to its most convenient path, i.e., the one that leads it to better situations of the game. This is made crossing backward the tree starting from leaves and considering that the other player also plays optimally. If a parent node is encountered at a level corresponding to the human player turn, a value equal to the minimum of the child node values is assigned. Otherwise, if the node corresponds to a turn of the artificial player, a value equal to the maximum of the values associated with the children must be assigned. Finally, the developed gaming algorithm returns the best move to perform and sends a command to the robot to physically execute the move (Fig. 3).

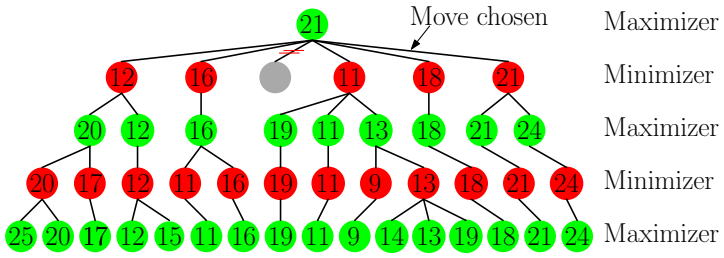


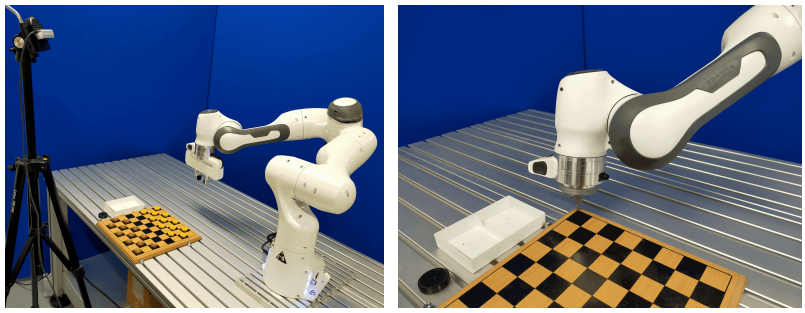
Fig. 3: Decision tree and path for the developed gaming algorithm.

3 Experimental setup

The robot chosen to perform this application is a Franka Emika manipulator with 7 degrees of freedom. The robot has an industrial-grade pose repeatability of $\pm 0.1\ mm$, a payload of $3\ Kg$, and a reach of $855\ mm$. It is equipped with a Franka hand as end-effector, a two-finger gripper used for pieces manipulation. The robot is controlled in Ubuntu 18.04 with Python 3 and ROS (Robot Operating System) Melodic. The gaming algorithm is also programmed in Python 3, instead the developed vision algorithm is implemented in GNU Octave and its functions can be easily called from the main program thanks to the `oct2py` library.

In order to simplify the algorithms (especially the computer vision one) and reducing calculation time, the relative position between robot and board is fixed so a continuously detection of the board is not required. Furthermore, we used a commercial checkerboard and pieces with standard and unmodified colors and shape in order to preserve the authenticity of the game.

For the vision system we used an Intel Realsense D435 depth camera with a resolution of 1920×1080 to acquire the board image. The camera is supported by a tripod and positioned above the board in a position that does not interfere with the robot motion. The experimental setup is shown in Fig. 4a.



(a) (b)

Fig. 4: Experimental setup (a), and robot calibration (b).

Before starting to play, it is necessary to calibrate the robot in order to obtain the position of the board with respect to the robot base reference frame. To do this, the robot has to touch the 4 corners of the board counterclockwise with a custom 3D printed tip as end-effector of the robot, starting from the corner to the left of the robot. However, only three points are strictly necessary for the robot calibration; the fourth is needed to compute the coordinates of the board center. The position of the corners in the Cartesian space is then obtained through direct kinematics (Fig. 4b). Remembering from Sect. 2.1 that the vision algorithm saves pieces position in the system of the board, knowing the real position of the corners allows us to define a transformation between the

two systems and calculate the real position of the pieces. In addition to this, a neutral pose has to be set from where the robot starts and where returns after the move, taking care that it does not interfere with the camera (Fig. 4a).

After the robot calibration, the camera setup is required. To do this, as mentioned in Sect. 2.1, the user first has to indicate on the image captured by the camera the position of board corners in the same order followed in the robot calibration, in order to match the real corners with the ones in the image. Then, the user checks if the *homography* and the black and white image are acceptable; if yes, the user continues with the indication in the image of the position of a white square, black square, white piece and black piece. Otherwise, the camera setup has to be restarted, trying to change lighting and/or camera position until these two images are acceptable. This camera-board calibration will be executed again only when the position of the camera is changed. Done this, the setup is finished and the game can start. For safety, the robot does not start its moves immediately after the human move, but it waits a keyboard signal. However, the robot motion is slow enough not to cause injuries in case of a potential collision with the human.

Moving on to the control and driving system of the robot, each trajectory is calculated using 10 waypoints between the start and the end position and interpolating with cubic splines, imposing the execution time and zero velocity and acceleration at the initial and final points. To sufficiently constrain the movements of the robot, during each trajectory the orientation of the end-effector is kept constant and the joints are moved as less as possible from the start to the end configurations, imposing the robot joints limits. In a standard movement, the robot starts from the neutral pose and picks the piece to be moved. From this state, the following moves depend on the type of play to be made:

- if it is a simple movement, the robot simply places its piece in the target square and returns to the neutral pose;
- in case of capture (even multiple), after placing its piece in the target square, the robot removes from the board the captured opponent's piece (or pieces);
- if its move brings a man to king, the robot removes its man from the board instead of placing it and picks up its king that the user prepared in a fixed position out of the board; this choice was made due to the difficulty of making the robot stack pieces, an operation that would require great precision because these must be fitted together rather than simply stacked.

4 Experimental results

Figure 5 shows the main actions performed during a game, i.e., pick (Fig. 5a), move (Fig. 5b) and place (Fig. 5c) a piece, pick a captured piece (Fig. 5d), remove a captured piece (Fig. 5e), and pick a king from its preparation point (Fig. 5f). The performance of the system is shown in a video available online [1].

[1] https://www.youtube.com/watch?v=KiR5qAI5S2M

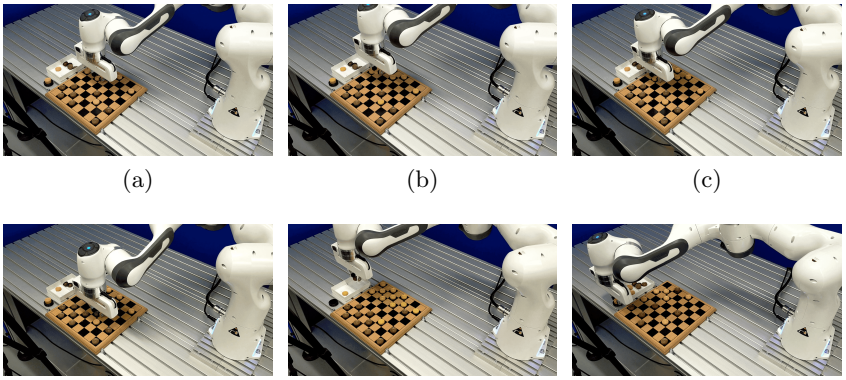


Fig. 5: Different game phases: pick a piece (a), move a piece (b), place a piece (c), pick a captured opponent's piece for removing it (d), remove a captured opponent's piece from board (e), and pick a king from its preparation point (f).

In the following, we examine the execution time of the entire algorithm, from image acquisition to selecting and performing the move. Considering the developed computer vision algorithm, its runtime is quite constant and it is about $1 - 2$ s. Moving to the gaming engine, as mentioned in Sect. 2.2, its runtime depends on the search depth and on the maximum number of moves considered (i.e., the tree breadth). Figure 6 shows the average time evaluation for different cases. For a fluid game play, a duration of about $1 - 2$ s could be considered adequate so, with 5 moves considered, a search depth of maximum 7 should be chosen. From the various games played we noted that against a not advanced player a search depth of 4 is enough to make the game challenging (most of human players think about the following $2 - 3$ moves at most), so the recommended maximum search depth will rarely be used.

In general, the majority of the algorithm execution time is spent on the physical execution of the move because the robot movements must be slow enough to pose no risk to the human player. Furthermore, this time depends on the type of move to be made (simple move, capture, etc.). We made several games in order to analyze them and the results are reported in Tab. 1. Adding together the times necessary to perform the various parts, a move by the artificial player lasts on average about 29 s.

Finally, we made a brief analysis of the success rate of the developed computer vision algorithm and the execution of the moves. As mentioned in Sect. 2.1, the first is sensitive to lighting conditions. For this reason, we implemented the system in the robotics laboratory of the University of Udine, where lighting is constant and this problem can be limited, providing a piece detection accuracy of 100% with standard light conditions. Instead, the second has a success rate of 98%, with rare failures mainly due to speed or acceleration limits violation.

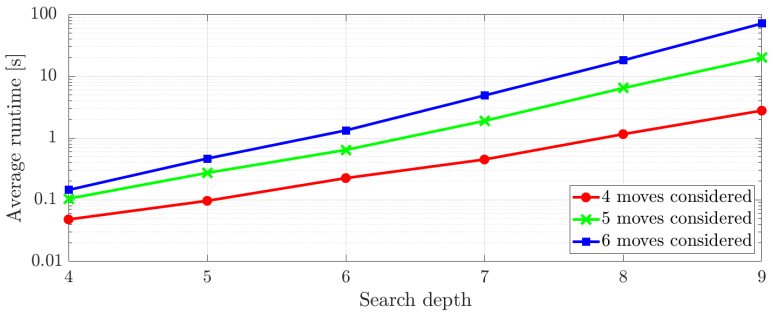


Fig. 6: Average runtime of developed gaming algorithm depending on the search depth and the maximum number of moves considered.

Move to execute	Simple move	Single capture	Double capture	Making king
Occurrences	261	52	18	31
Minimum duration	17 s	28 s	49 s	29 s
Maximum duration	21 s	34 s	54 s	34 s
Average duration	18.94 s	31.7 s	51.14 s	31.5 s

Table 1: Execution times of various types of moves.

5 Conclusion

In this paper, a robotic system playing Italian checkers has been presented. The proposed system is able to know the game state via a camera, evaluate the best move, and physically move pieces. In real-world application tests we were able to conduct full games without any errors, confirming the feasibility of proposed approach. Future work will focus on the improvement of all algorithm parts, making the developed computer vision approach more robust using lighting isolation and a dedicated lighting system, or moving to a solution based on model recognition. Furthermore, the alpha-beta pruning algorithm will be implemented to reduce the run time of the developed gaming engine and improving its performance. We will also speed up the moves in order to reduce execution time.

Acknowledgments

The authors would like to thank Diego Masotti, Jacopo Foltran and Alberto Ragazzon for their help with the development of this system. This research was partially supported by the Laboratory for Big Data, IoT, Cyber Security (LABIC) funded by Friuli Venezia Giulia, and the Laboratory for Artificial Intelligence for Human-Robot Collaboration (AI4HRC) funded by Fondazione Friuli.

References

1. L. Scalera, A. Giusti, R. Vidoni, and A. Gasparetto. Enhancing fluency and productivity in human-robot collaboration through online scaling of dynamic safety zones. *The Int. Jour. of Adv. Manuf. Tech.*, 121(9-10):6783–6798, 2022.
2. S. Seriani, P. Gallina, L. Scalera, and V. Lughi. Development of n-dof preloaded structures for impact mitigation in cobots. *J. of Mech. and Rob.*, 10(5):051009, 2018.
3. J.-Y. Lin, M. Kawai, Y. Nishio, S. Cosentino, and A. Takanishi. Development of performance system with musical dynamics expression on humanoid saxophonist robot. *IEEE Rob. and Autom. Lett.*, 4(2):1684–1690, 2019.
4. L. Carbonari, M. Forlini, C. Scoccia, D. Costa, and M.-C. Palpacelli. Disseminating collaborative robotics and artificial intelligence through a board game demo. In *Int. Conf. on Mech. and Emb. Syst. and Appl.*, pages 1–5. IEEE, 2022.
5. L. Scalera, S. Seriani, A. Gasparetto, and P. Gallina. Non-photorealistic rendering techniques for artistic robotic painting. *Robotics*, 8(1):10, 2019.
6. L. Scalera, S. Seriani, P. Gallina, M. Lentini, and A. Gasparetto. Human–robot interaction through eye tracking for artistic drawing. *Robotics*, 10(2):54, 2021.
7. E. R Escandon and J. Campion. Minimax checkers playing GUI: A foundation for AI applications. In *XXV Int. Conf. on Electr., Electr. Eng. and Comp.*, pages 1–4. IEEE, 2018.
8. A. A Elnaggar, M. Gadallah, M. A. Aziem, and H. Aldeeb. Autonomous checkers robot using enhanced massive parallel game tree search. In *9th Int. Conf. on Inf. and Syst.*, pages PDC–35. IEEE, 2014.
9. E. E Kopets, A. I Karimov, G. Y Kolev, L. Scalera, and D. N Butusov. Interactive Robot for Playing Russian Checkers. *Robotics*, 9(4):107, 2020.
10. F. J Rodrıguez-Sedano, G. Esteban, L. Inyesto, P. Blanco, and F. J Rodrıguez-Lera. Strategies for haptic-robotic teleoperation in board games: playing checkers with Baxter. *Strategies*, 31, 2016.
11. L. Carrera, F. Morales, J. Tobar, and D. Loza. Marti: A robotic chess module with interactive table, for learning purposes. In *World Congress on Eng. and Comp. Sci.*, pages 25–27, 2017.
12. E. I Barakova, M. De Haas, W. Kuijpers, N. Irigoyen, and A. Betancourt. Socially grounded game strategy enhances bonding and perceived smartness of a humanoid robot. *Conn. Sci.*, 30(1):81–98, 2018.
13. D. J Brooks, E. McCann, J. Allspaw, M. Medvedev, and H. A Yanco. Sense, plan, triple jump. In *Int. Conf. on Tech. for Pract. Rob. Appl.*, pages 1–6. IEEE, 2015.
14. H. M. Luqman and M. Zaffar. Chess brain and autonomous chess playing robotic system. In *Int. Conf. on Aut. Rob. Syst. and Comp.*, pages 211–216. IEEE, 2016.
15. L.-H. Juang. Humanoid robots play chess using visual control. *Mult. Tools and Appl.*, pages 1–22, 2022.
16. P. Kołosowski, A. Wolniakowski, and K. Miatliuk. Collaborative robot system for playing chess. In *Int. Conf. Mech. Syst. and Mat.*, pages 1–6. IEEE, 2020.
17. A. T.-Y. Chen and K. I-K. Wang. Robust computer vision chess analysis and interaction with a humanoid robot. *Comp.*, 8(1):14, 2019.
18. C. del Toro, C. Robles-Algarín, and O. Rodríguez-Álvarez. Design and construction of a cost-effective didactic robotic arm for playing chess, using an artificial vision system. *Electr.*, 8(10):1154, 2019.
19. Italian draughts. https://www.ludoteka.com/clasika/italian_draughts.html. 2023.
20. S. G. Diez, J. Laforge, and M. Saerens. Rminimax: An optimally randomized minimax algorithm. *IEEE Trans. on Cybern.*, 43(1):385–393, 2012.