Backgammon 4 Real

A program for playing Backgammon vs. a computer opponent on a real Backgammon board

Final Project: Or Hayat & Oz Mishli

Introduction to Computational and Biological Vision, 2021

Department of Computer Science, Ben-Gurion University

March 7th, 2021

Introduction

The game

Backgammon is one of the oldest strategy games known, dating back to 5,000 years ago. It involves both strategy and luck, stemming from rolling two dice as part of the game. Generally, it is a two-player game where each player has 15 checkers at a certain color (typically, black and white or black and red, but other colors exist). The board is divided into 24 triangle-shaped cells in two alternating colors, and the checkers are moving around them based on the results of a dice roll (one roll per turn). The objective of each player is to move all her/his 15 checkers off the board, and the first player to achieve this wins the game.

For further information on the game and its rules, please refer to this How to Play Guide or to the corresponding Wikipedia entry.

The game has been studied extensively in computer science, mostly from the artificial intelligence (AI) point of view, with the goal of developing algorithms that would beat world class human players (e.g. TD-Gammon which was developed in 1992 by IBM).

Project Goals

The goal of this project is to develop a fully functional software program that allows a human player to play backgammon versus an AI-powered computer opponent on a physical board game, while delivering very good user experience. The setup should be simple, ideally supported on a typical room/office environment with a standard laptop and USB webcam for image acquisition.

It is important to note that this project mostly focuses on the acquisition and analysis of the physical backgammon board from a standard web camera, integrating it flawlessly into an existing Backgammon AI framework and while keeping performance and user experience as good as possible.

Given the immense variety of colors, types and sizes of backgammon boards (see figure 1), we decided to keep the project scope reasonable by focusing on a specific, standard Backgammon board and checkers that are common in Israel (figure 2). Having said that, it should be possible to adapt the software rather easily to different Backgammon boards, mostly by parameter tuning of the corresponding HSV filters.

Figure 1: A few examples[1] of different Backgammon boards and checkers. Note the sheer differences in colors and shapes in this very small sample.

Figure 2: The Backgammon board and checkers selected for this project.

[1] Images taken from (starting from top-left, clockwise): https://www.amazon.com/, http://www.jamesrobertwatson.com/backgammon.html, https://thinkcomputers.org/., https://en.wikipedia.org/wiki/Backgammon,

System Overview

This section describes the use cases and the system design in a fair level of detail, and includes a few sections covering high level use cases, the system design and its modules. It will be followed by a detailed section covering the computer vision aspect, which is the major focus of the project.

Use Cases

Environment setting

The system is designed to be used as follows:

1. A standard laptop used in a typical setting on a desk.
2. A Backgammon board is laid out open on the desk (or another flat surface) either to the left or right of the laptop, at the user's convenience (other board placements such as in-front or at the back of the laptop are theoretically possible, however not recommended due to degraded user experience).
3. A standard USB-webcam[2] connected to the laptop, placed opposing to the player. There are no assumptions on the camera angle and position, as long as the board is set on a flat surface and fitted entirely in the image.

Game Scenario

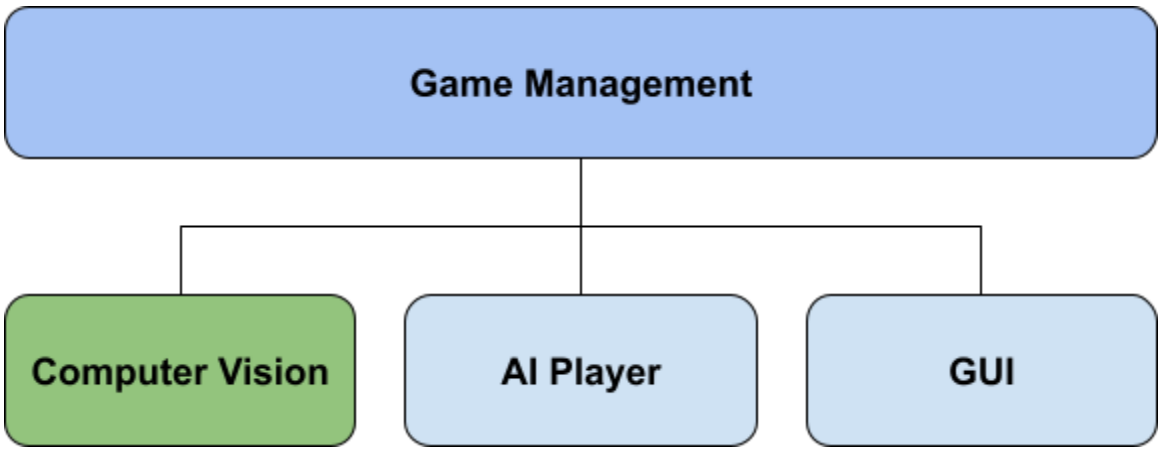
1. Setup:
    a. After launching the program, the user has to choose between player types for both players. Three options are available:
        i. Human player.
        ii. AI-powered computer player (difficult).
        iii. Random computer player (easy, for beginners).
    b. Once game setup is done, the program needs to perform an initial calibration process. This process is performed only once per game. It is straightforward, and requires taking a picture of the empty Backgammon board (i.e., with no checkers).
2. Game
    a. The game begins, with alternating turns between the players, until the end.
    b. The game is accompanied by a visual representation of the Backgammon board on the screen, which is correlated with the physical board.

[2] While it is theoretically possible to use a built-in webcam of the laptop, in practice it is discouraged and highly inconvenient for the user as it opposes the user but should also capture the board.

c. Rolling the dice is performed automatically for both players (human and computer) and is represented visually on the screen, i.e. no physical dice rolling is required. We figured out that this will provide a smoother and more intuitive user experience, as the human player won't have to roll the dice twice per round (once for himself, once for the computer), or alternatively to roll differently for the user vs. the computer.
d. The system enforces game rules, and refuses continuing the game if a certain move is illegal, or e.g. if the user moved the computer's checkers not as instructed.
e. On the computer turns, the user will be asked to move the computer's checkers according to instructions on the screen. Once the correct moves have been completed the user will be able to click a button to continue to the next instruction or to roll the dice for the next player.

Modules

This section briefly describes the main four modules of the system, as illustrated in figure 3 below.



Game management

The game management system that we built was based on OpenAi gym toolkit. This toolkit includes many AI environment descriptions for various games, including Backgammon specifically. We leveraged their Backgammon environment and extended it to fit our needs for the project. In a nutshell, the core environment describes the game programmatically: it holds the current player, the board state, the bar state and the offbar state for each player. It has four main functions:

1. Reset: reset the game state into the initial state (the game starting position).
2. Step: this function gets the set of moves that the agent wishes to perform, and changes the environment based on those actions.
3. Get_valid_actions: this function provides all the available legal moves for the current player, based on the current board state and the dice roll.
4. Get_opponent_agent: this function ends the player's turn and returns the move of the opposing agent.

As mentioned above, we had to extend the OpenAI system with various adaptations, such as fixing its ability to render the game state to a window and adding the ability to break a full computer turn to a few single moves, for a better user experience.

In addition to the functionality described above, this module also interacts with all other modules, orchestrating between the GUI, the AI and the computer vision modules for creating a smooth game experience.

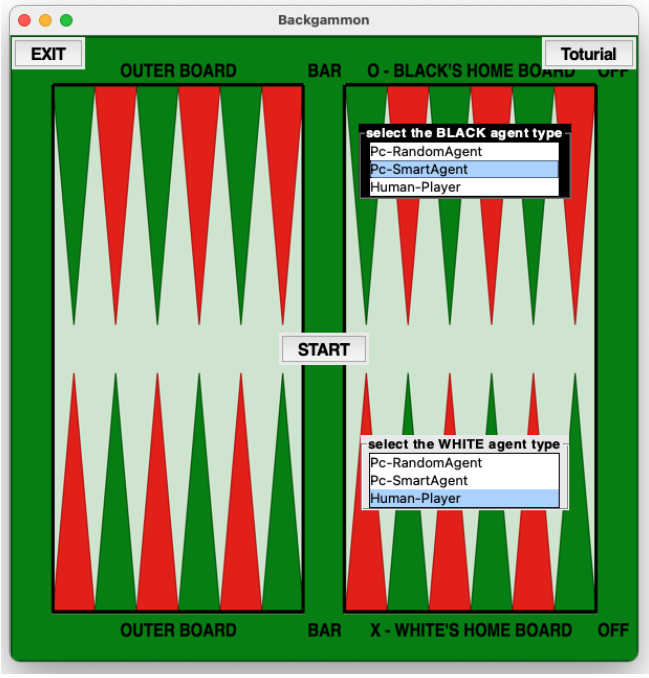
GUI

The GUI was built using the Tkinter library, and it is the entrance point of the game, immediately when launching the executable. It is based on a GUI provided by OpenAi gym, with significant adaptations and modifications for our project. The game starts by offering the user to choose the agent type for each color: the white player agent type and the black player agent type.
Then, the player needs to perform the calibration process with the empty board. The process is smooth, with informative error messages and images if the calibration process fails.
Once the calibration is successful, the GUI presents the initial game state that the board needs to be in. After the player prepares the initial game board she/he will click on the button to check that the board is set up correctly to start the game.

The first player is selected randomly, and then the game is running, turn after turn. We print the selected player and the roll dice to the console and also render the dice roll in the UI. If the current agent is the computer agent, then it will split the selected action to discrete, single moves. Each such move is rendered on the screen, waiting for the player to play that move on the real board, then click on a “Next” button after checking that the expected board state is detected by the camera.

For the human player turns, the player will make the move on the board and then click a “Next” button. Then, the system will check the physical board state and verify whether its state is one of the possible states that are available from the last verified game state (taking into account the last board position and the rolled dice).

Figure 4: A snapshot of the GUI presenting the opening screen, where the player selects the agent types for each color.



The Game Agents

The system includes 3 agents with different move selection policy:

1. **Random computer agent:** the agent selects a random move from its available list of moves, as per the current board state and the dice roll. This is considered an easier setting for the human player rival.
2. **Td-Gammon agent:** in order to have an agent that performs far better than a random one, we trained a neural network to predict a score for each move for the WHITE AGENT to win following that move.
   Then, the white agent will use this network and attempt to maximize the score, while the black agent will use it and try to minimize the score. Essentially, the network was trained with two computer agents playing against each other.
   After the training process was completed, the network was evaluated, scoring 100% win rate against the random agent, and close to 50% win rate playing against another Td-Gammon agent.
3. **Human agent:** A human player that performs all the moves of the game, excluding the dice roll (that takes place using the computer, as mentioned above).

Computer Vision

The computer vision module has been developed from scratch by the project team. The module is responsible for the following:

1. Webcam connection and image acquisition.
2. Detecting the board in an image and performing calibration.
3. Detecting the board including checkers on it, acquiring current board status.
4. Translate the visual board status to an object describing the game status, as required by the game management module.

Computer Vision Module

This section describes in detail the process we have gone through while developing the computer vision module, which was indeed the greatest challenge of this project. We outline the challenges and the solutions systematically, as we encountered them during the R&D process.

Early in the process, during system design we have realized the challenging nature of capturing Backgammon checkers and board due to the following reasons:

1. There are many checkers (30), they are relatively small and have no fixed placeholders on the board (i.e. there are no physical boundaries, apart from the triangles drawn on the board). However, those cannot even be considered as boundaries; when stacked, a couple of checkers on a triangle may completely hide it.
2. The black checkers are literally in the same color of the black triangles, making it difficult even for a human eye to notice their presence on a black triangle in a picture taken by a standard webcam.
3. The board is made of mediocre quality wood, it's surface and edges/bars aren't smooth, and have color/texture variations. Moreover, the board color is not easily distinguishable from its surrounding environment.
4. As checkers cover important visually-recognizable elements of the board (e.g. the red triangles) during the game, it is very hard (if not impossible) to use these for accurate board detection, or alternatively to easily capture board status in a stateless fashion, without relying on previously captured images.

A photo of the board and checkers chosen for this project can be found at figure 2 above.

Thus, we have taken the following approach for accurate detection:

1. The detection of the locations of the checkers on the board during the game will be done in respect to an already existing image of the empty board captured during initialization, relying solely on the pixel coordinates of the checkers for their placement in a pre-captured map (that contains multiple containers in respective locations on the board where checkers can be placed).
2. As per #1, it is clear that the board detection and alignment must be very accurate in order to correctly report the checkers in the right location -- during the entire game.
3. Due to the challenges in colour distinction in some areas, visual cues may be used where needed to ensure sufficient accuracy of detection.
4. The detection of the board should rely on its boundaries, as its visual elements may be covered by checkers during the game.

The following sections will describe all process steps in detail.

Board Detection

The first challenge we faced was the accurate detection of the board area. Most of the experiments we conducted were on a table which was in a color quite similar to the board color. Initially, we have tried multiple techniques for accurately detecting the board, including using various HSV color filters, along with blurring and erosion/dilation. Unfortunately, all of our experiments failed after investing days in attempting to detect the contours of the board. As predicted, the color of the board, being not very much distinguishable from the table, was a challenge. See figure 5, comparing the image of the board vs. the one of the best HSV filters we could apply empirically.

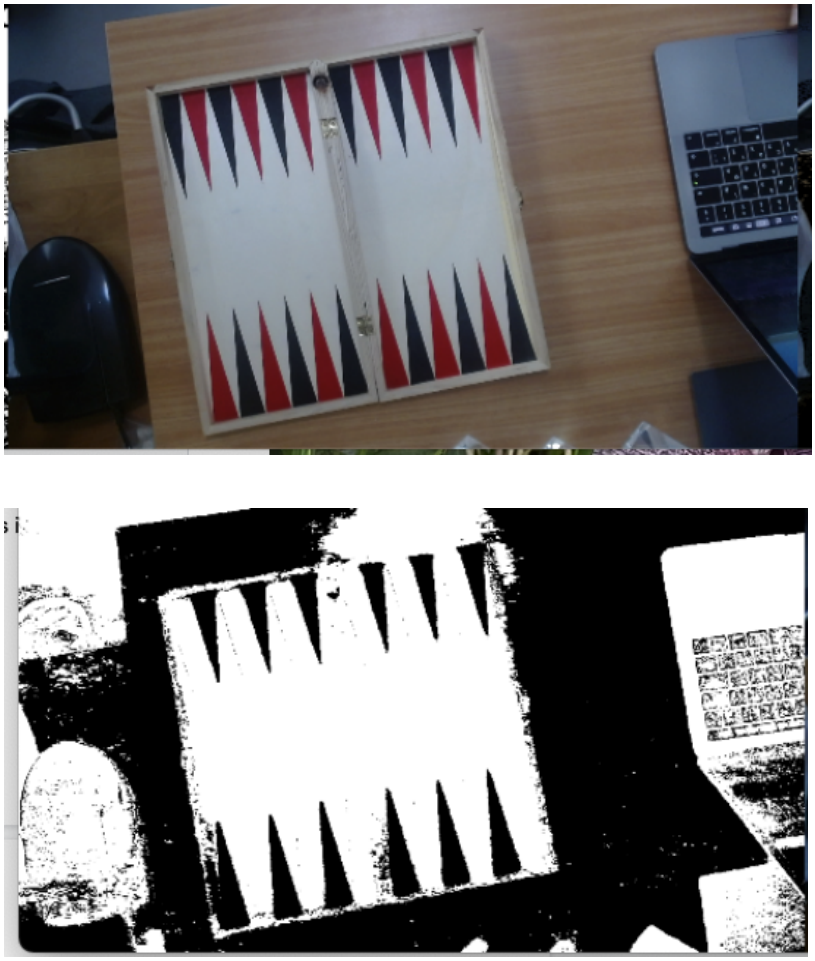


Figure 5: The original image acquired by the webcam (top) and the seemingly best performing HSV filter (bottom). Note the light reflection on the table near the top right corner of the board; it made the detection of a square contour around the board virtually impossible, even after blur and/or erosion and dilation.

After consultation, we've figured out that adding a minor visual cue would be necessary for getting high reliability board detection. Thus, we have attached four small green circle stickers, one in each corner of the board. They can be observed in figure 6 below, along with how they were detected using the HSV filter:

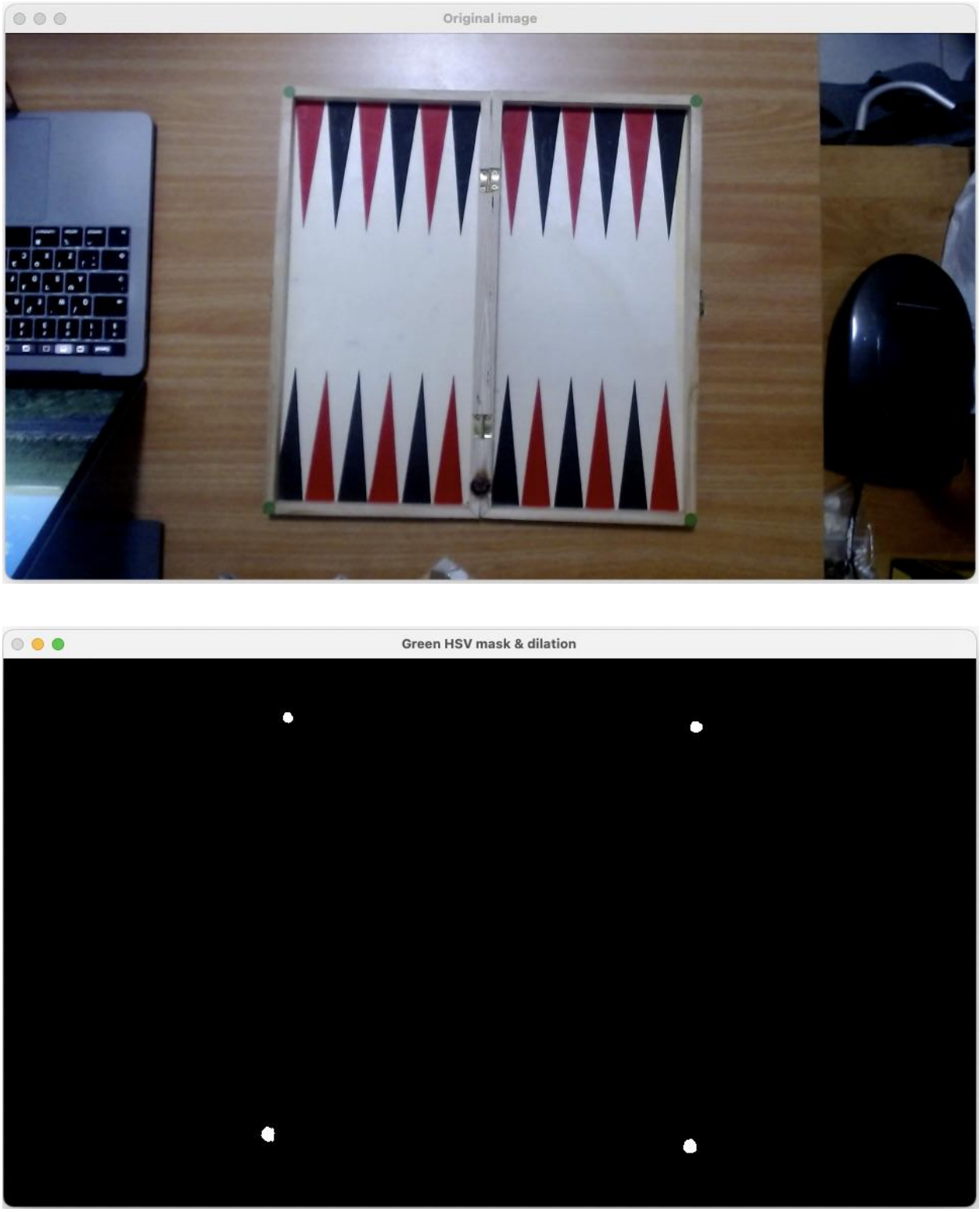


Figure 6: Original board image along with 4 small green stamps, one at each corner of the board (top); The same image after applying green HSV filter, erosion and dilation, for highlighting the circles.

This setup was just enough for getting the board detection right. In order to reduce noise and highlight the green circles, we used erosion and dilation. Then, the Hough circle transform was used to locate the circles. However, as we've noticed issues in different light conditions with objects near the board (adding some "false-positive" circles to the image occasionally), and in order to increase the robustness, we added a very simple geometric check. Specifically, we used the fact that the board is a square, determining the distance ratios expected between the four points, with some factor to enable reasonable rotation of the camera. This increased robustness was added at the expense of limiting the camera angle, however we figured out that the more frequent use case is when the board is moved / rotated sideways on the table, which is completely unaffected by this detection method, while camera angle is rather stable and in the vicinity of 90° as it has to capture the entire board, which is quite large. This additional check can be enabled/disabled easily.

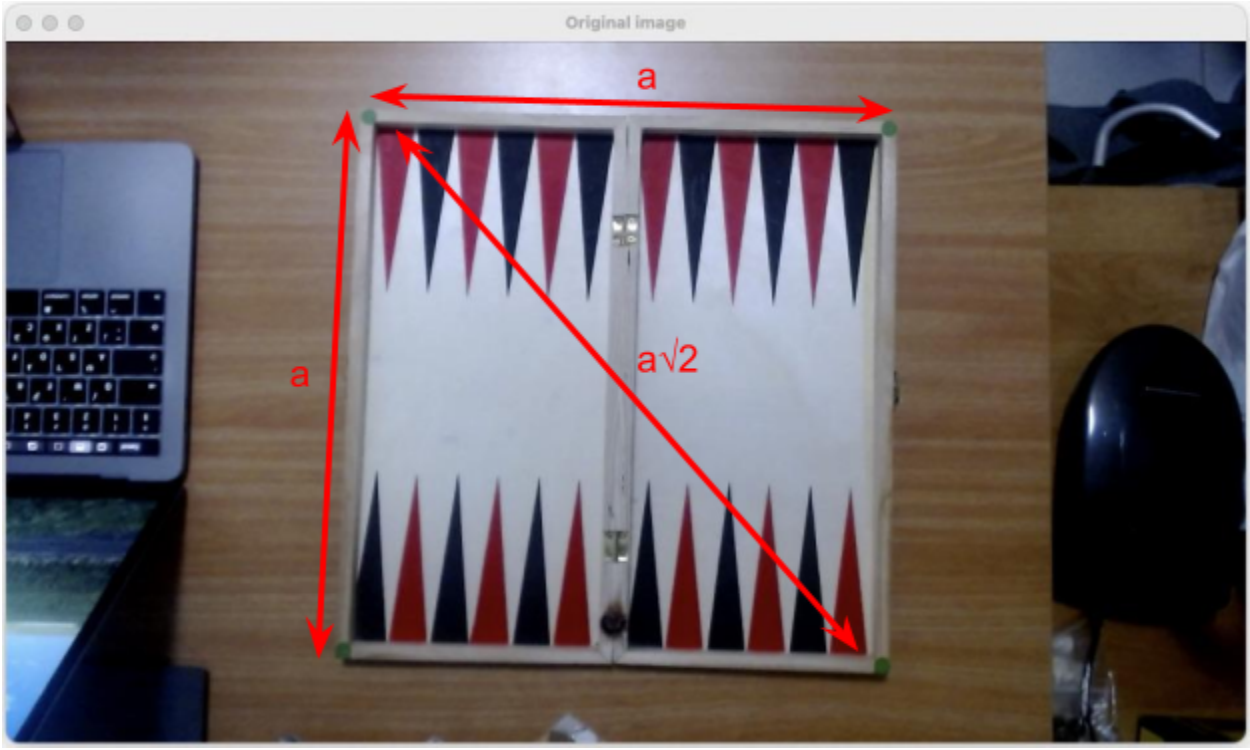


Figure 7a: The expected length between points according to the square shape of the board.
Figure 7b: The detected board after homography.

After detecting the board vertices, homography was performed in order to ensure that we have an identically looking image at each detection, as if the camera was aligned directly over the board. In order to perform it, we had to locate where each of the board vertices is compared to the image vertices (e.g. top-left, bottom-right). This is done automatically and then homography transformation is applied, from each of the 4 green points to a 400 x 400 pixels square (Figure 7b).

Board Calibration

As mentioned above, calibration takes place once, just before the beginning of the game, and requires an empty board. The goal of this process is to find all relevant areas in the board where checkers can “legally” be placed, define these areas numerically on the detected board image and tag them uniquely. We named these *checkers containers*. Overall, we had to tag 25 checkers containers, that include 24 triangles (12 red, 12 black) and the bar, which is the long rectangle separating both parts of the board. (checkers removed by the opponent are being placed in the bar).

Calibration was performed on a tightly cropped board image (figure 7b). The calibration process consists of two steps, and is performed automatically with no user intervention.

Red Triangles Detection

Assuming tight cropping and alignment of the board image, we figured out that detecting all 12 red triangles would be (i) most accurate, due to the distinct shape and color, and (ii) their contours are enough for detecting all other checkers containers in the board.

Thus, taking the aligned and cropped board image as an input, this method applies Gaussian blur for noise removal, and then applies HSV filter for the red color. Then, all contours in the image are extracted and processed according to the following:

1. Each contour is approximated to a polygon.
2. FIltering only contours that have exactly 3 edges.
3. We observed that triangles are isosceles triangles with ratio 4.5:1 to the base, and this is verified geometrically on all triangles filtered at step #2. Note that the camera angle doesn’t affect this check due to the homography performed during board detection.

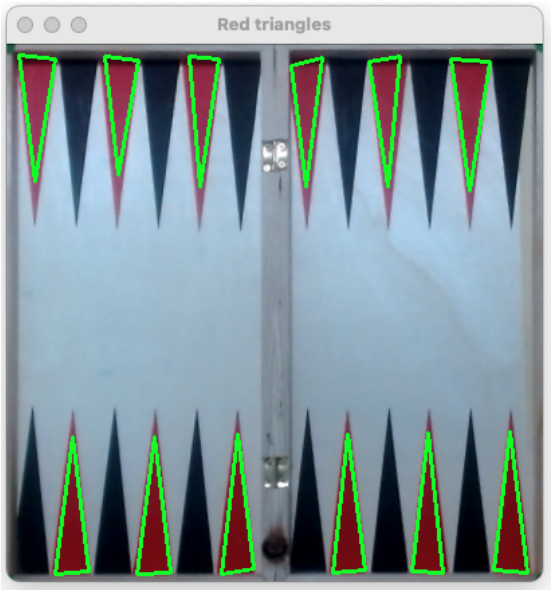


Figure 8: Red triangle contours as detected after following the process mentioned above on the cropped board image.

Creating checkers containers

This stage is performed analytically, without the need of additional images. It gets as input solely the red triangles contours obtained previously, and includes the following steps:

1. Create a bounding box around each of the triangle contours. The bounding boxes are more useful than triangles, as in practice the circular checkers override the triangles and are bound by a “virtual” rectangle. Of course, this is also true for the bar which is a rectangle.
2. Per step 1, we have 12 bounding boxes, 6 in each side of the board. Now we sort them top to bottom, and then sort the two top-and-bottom lists left to right.
3. Create a list with 25 containers, and assign each of the existing rectangles to their corresponding numbers in the Backgammon board (1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23).
4. All of the remaining containers were created based on the dimensions of the containers created at step 2. For example, black triangle #2 is between red triangles #1 and #3, and its dimensions are computed according to the width and height of their bounding boxes. Same goes for the rest of the black triangles and the bar (detailed description can be found in the code (`bg_board_cv.py`, function `get_checkers_containers`).

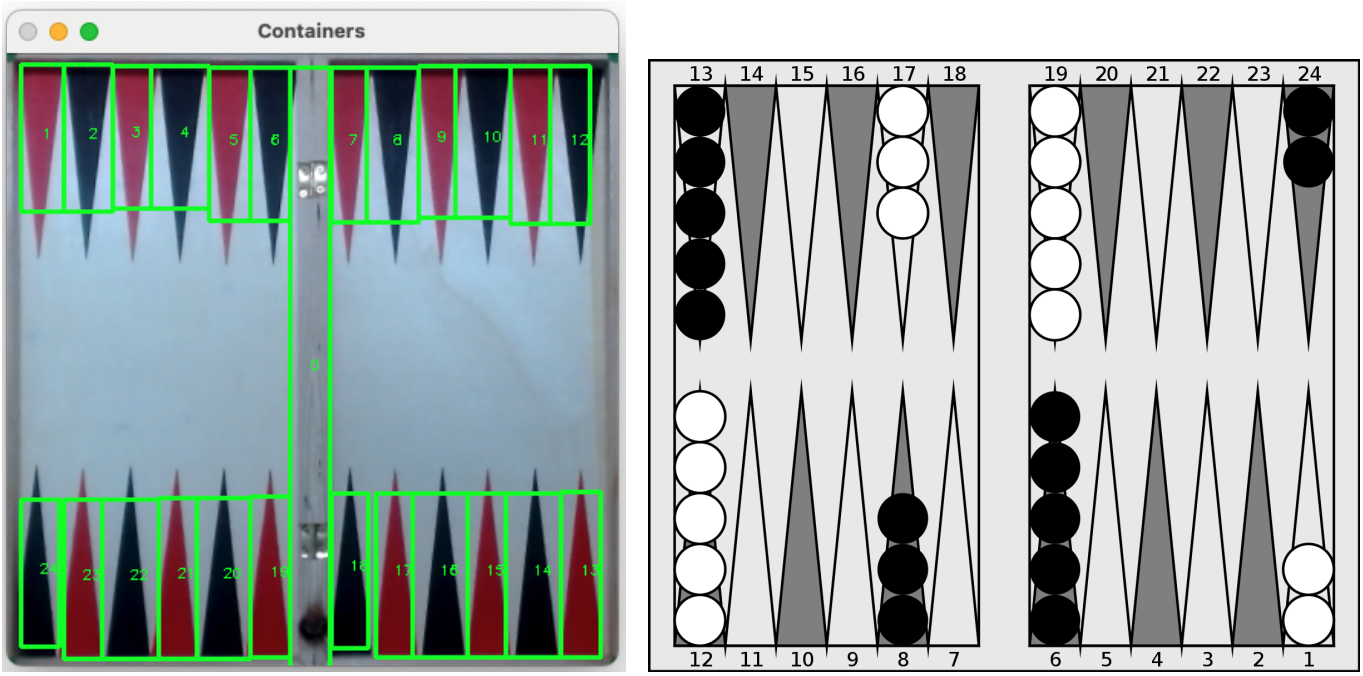


Figure 9: The cropped board image with the checkers containers’ bounding boxes and their numbers (left) and the Backgammon board numbering[3] (right). Note that the left image is flipped, as the image is taken from the computer point of view, which opposes the user (i.e. the user is in front of triangles 1-12).

[3] Image taken from https://en.wikipedia.org/wiki/Backgammon_notation

During the Game

Following the calibration step performed before the game begins, all that needs to be done during the game is to detect the location of various checkers on the board, and assign each to one of the checkers containers identified previously.
However, as noted above, this part also had a challenge, where the black checkers could not be distinguished when placed on a black triangle. Thus, we had to paint the black checkers with a distinctive color (a silver marker was chosen).

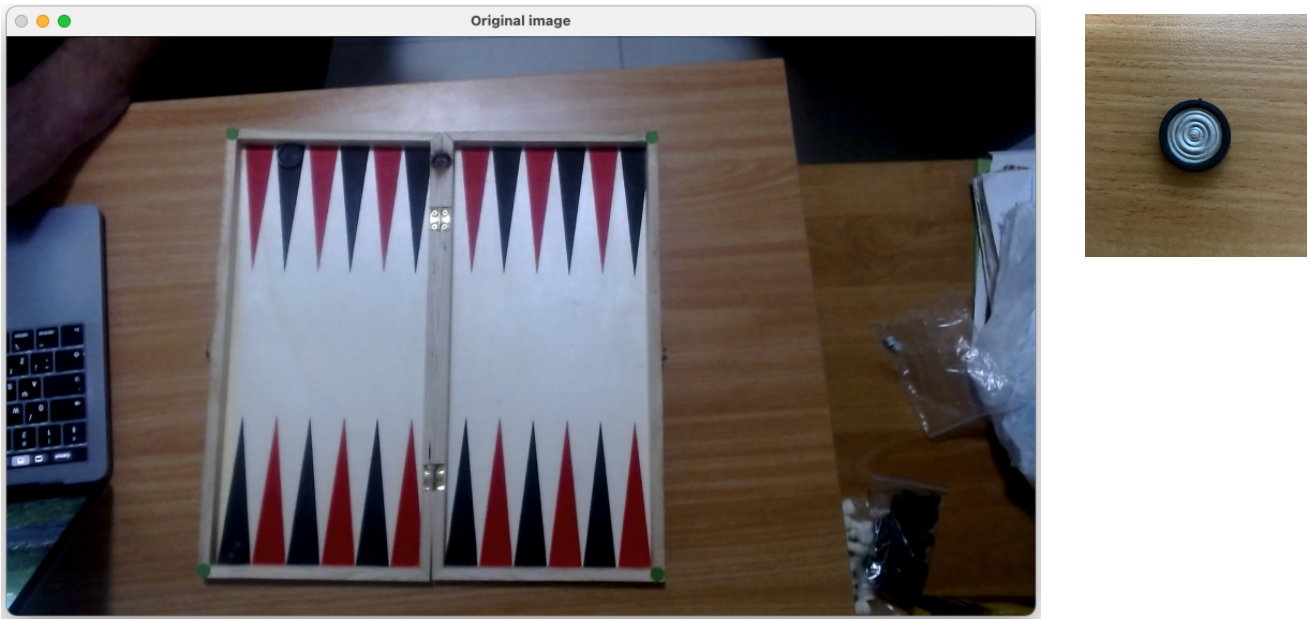


Figure 10: The left image was taken using the webcam, and the board has two black pieces on it. Are you able to locate them? Probably not. Obviously, also the computer can't... On the right, a black checker after recoloring.

Whenever the physical board status is inquired by the game management module, it calls the `get_current_board_status` method, which performs the following:

1. Board detection using the same method mentioned above.
2. Using two HSV filters, one for capturing the white checkers and one for capturing the black (now silver) checkers.
3. Performing erosion and dilation for cleaning the noise and highlighting the circles in each image.
4. Use Hough circles transform on each of the two images.
5. Perform steps 2-4 on multiple images (around 11), and choose the two images with the highest number of votes in terms of checkers counts (black and white). For example, if 10 images were taken, 6 of them had 14 white checkers, 7 of them had 12 black checkers, then these corresponding images are chosen respectively for the white and

black checkers. This mechanism was added for additional robustness as per issues of Hough transform not performing well on all pictures.

6. Associate the detected checkers to their checkers containers according to the container that includes their center. Return error to the game management if found checkers in an illegal location (i.e. outside all of the containers).

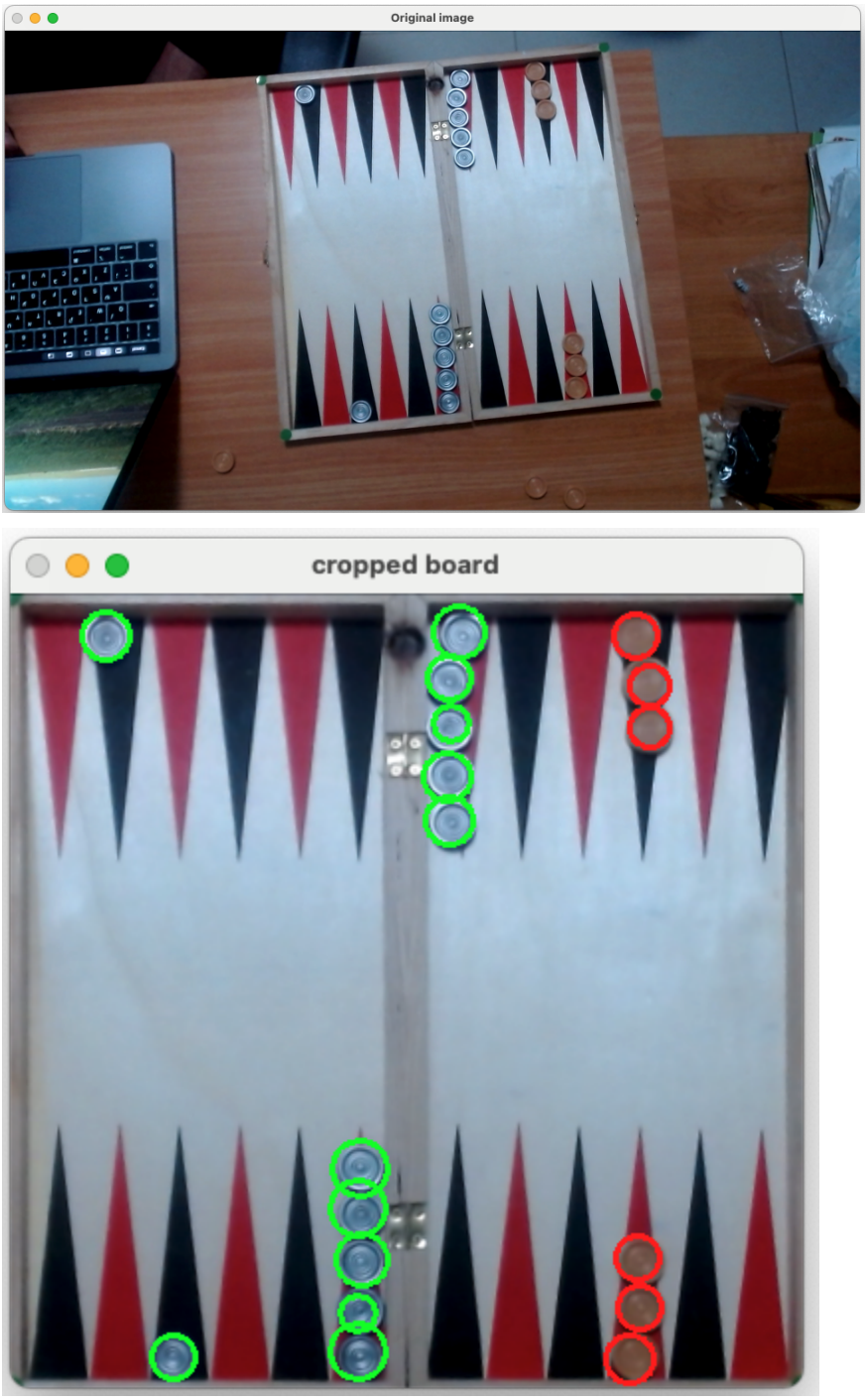


Figure 11: The original image (top) and the cropped board along with the detected checkers. White checkers denoted in red, black denoted in green (bottom).

Experiments and Setup

Hardware and physical setup

The game board is placed on a table, opposing the computer screen and the camera. Distance of approximately 40 - 80 cm from the camera to the board was tested successfully.

Laptop: MacBook Pro 2.3 GHz Quad-Core i5
Webcam: Logitech HD PRO Webcam C920

Backgammon board: 33.5 x 33.5 cm square, divided by a bar in the middle.
The board consists of 24 triangles, 12 of which are red and 12 are black.
15 white checkers, caramel coloured.
15 black checkers, black coloured (later painted silver).

Software

Operating system: macOS Big Sur 11.2.2
Python version: 3.8.5
OpenCV version: 4.5.1

Experiments Summary

Multiple experiments were conducted with the system. It was found to function pretty well, robust to board angle changes and to minor camera angle changes. Generally, lighting conditions changes were harder to deal with, sometimes requiring altering some of the HSV filters. The system performed better with a mix of indoor lighting and sunlight from the window, and thus most of the tests were conducted in such an environment. Under these environment conditions, we were able to play against the computer on the physical board, as designed.
In terms of performance, the experience was really smooth, with sub-second performance for capturing the game status, virtually nonexistent from the user point of view.

Conclusions

First, even though it was hard and even frustrating at times when encountering challenging problems, we really enjoyed working on this project. More than anything, we experienced first hand the great challenge of computer vision, learning to appreciate our eyes as a magnificent wonder, while realizing how far computers are from humans in this area. Learning deeply how hard it is to develop robust computer vision applications, we now look at such software in a different way. So, to conclude, it was a great learning experience for us, and a good closure to the course as we used many of the building blocks and tools studied during the lectures.

While we are happy with our deliverables, we have a wishlist of upgrades for the future…

1. Improve granularity of board detection, so boards in different colors (as seen at figure 1) could be detected with a similar calibration process.
2. Improve the system robustness to deal better with lighting conditions.
3. Remove the need for visual cues (green stickers) from the board detection.
4. We believe that using a deep neural network trained properly may be able to address both #1 and #2 and #3.
5. Detecting automatically when the player did the pc turn/his turn.
6. Change the UI color of triangles automatically so they adapt to the color of the board (can probably be done by taking an image of the board during calibration and using it on the UI).
7. Add game series option (tournament), i.e. need a series of games in order to win.

References

1. Ohad Ben-Shahar, Introduction to Computational and Biological Vision lecture notes (2020-2021).
2. Openai-gym
3. gym-backgammon
4. Td-gammon -ai
5. Temporal_difference_learning
6. Backgammon -- Wikipedia, https://en.wikipedia.org/wiki/Backgammon
7. https://cvexplained.wordpress.com/2020/04/28/color-detection-hsv/
8. https://docs.opencv.org/4.5.1/d9/dab/tutorial_homography.html
9. https://learnopencv.com/image-alignment-feature-based-using-opencv-c-python/
10. https://en.wikipedia.org/wiki/Image_moment for calculating moments of red triangles
11. https://docs.opencv.org/master/dd/d49/tutorial_py_contour_features.html