Summoner Wars gameplay detection

Description of Our Approach

Aleksander Hański, Jakub Radziejewski

Informations from milestone 1

Summoner Wars - two player game, where you summon creatures (cards) on the game board, cast spell cards and try to defeat the opponent's leader - summoner.

As we chose a really complex game we decided to fit it to the requirements of the project (having enough number of detected objects (3 plus board) and events(7)), and focus on the most crucial parts of the game and focus on corrected detection of it. Below is the logic implemented in our videos and how we adapted to the task requirements using computer vision.

Description of the data set

The dataset consists of nine MP4 videos showing a simplified for our task constraints version of Summoner Wars on a different difficulty level: easy, medium and hard.

Dataset and our output detection video results are available at:

https://drive.google.com/drive/folders/1lridoDl2Of4L5lEuyWcF35dPPpg4n7ap

Key Characteristics

Game Board: Consists of two boards: outer and inner. We focused on detecting the inner one, as this is the actual game place. The inner size grid consists of 48 cells - 6 rows x 8 columns. The board is divided into two team territories (Team A left, Team B right), and the place where the card appears first implies the team this card is playing for. Board parameters serve as the reference measurement for detectors which have adaptive parameters based on the actual grid size in every frame.

Elements of the game

Playing Cards: Portrait-oriented rectangular cards, for simplicity we assumed that all cards have the same strength (2 tokens need to be put on it to lose the battle).

Battle Tokens: Small circular red tokens that appear above and below active battle zones as hit indicators. Two tokens are enough to make the card lose the battle.

Dice: Decides on the number of tokens put on the card during the battle, if the result of the throw is greater or equal to 3, a token can be put on the card. Otherwise, no token is awarded.

Detected events

Setup: During setup each team is required to place two cards on their side of the board. After that movements and dice throws can be made. An appropriate message is written out on the screen.

Card movement: A tracker is assigned to each card, and we track how the card is moving on the board. An appropriate message is written out on the screen.

Battle detection: Battle is detected when two cards of opposing teams are facing each other, being in the adjacent grid cells. An appropriate message about which card is fighting against which is written out on the screen.

Dice detection: The player who is attacking, throws dice first. The number of dice depends on the number on the attacking card, which we do not detect. The dice and its result is detected, a throw of more than one dice is allowed, then more tokens can be placed on the board, and also an instant win is possible, when two or more dice have scored higher than 2 each. An appropriate message about the number of dice and scores is written out on the screen.

Token detection: Token is detected and tracked after being placed in an active battle place, it is only detected for the period of the battle being active, after that even when it is still in the same place, it will be no longer detected. An appropriate message about the number of hits and which card got hit is written out on the screen.

Battle end detection: End of the battle is detected when any of the cards leave the battle bounding box, it is taken down from the board. The winning card remains in the same place. An appropriate message about who lost is written out on the screen, and the battle bounding box is deleted.

End of the game detection: Game is ended when one team is left with no cards on the board. An appropriate message is displayed.

Description of the used techniques

Board Detection

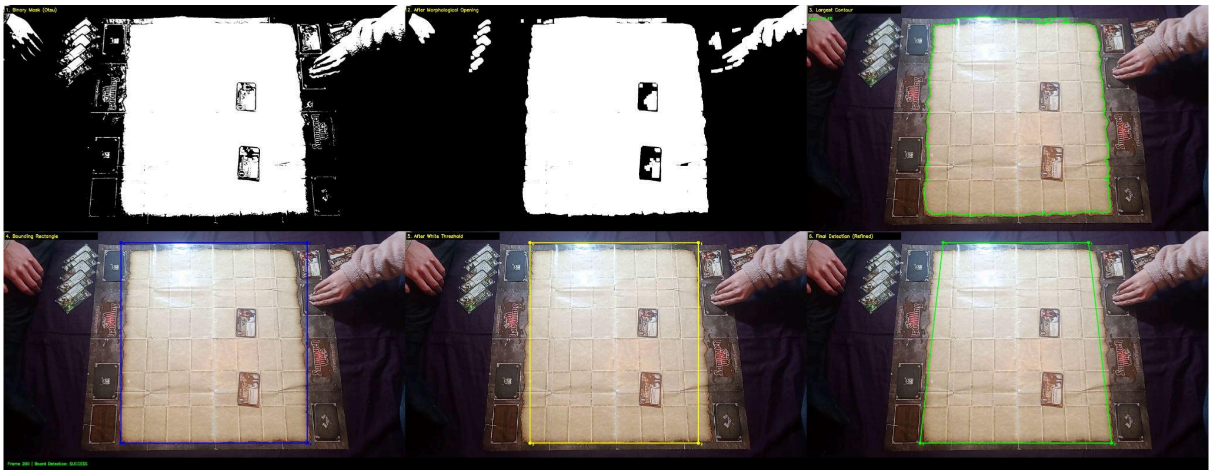
We first apply gaussian blur, then apply Otsu threshold and multiply the result by 110% to detect less noise.
Then we perform an opening to get rid of more noise. In the cleaned mask we find contours, and select the largest detected contour.
Then we find a rectangle bounding this region.

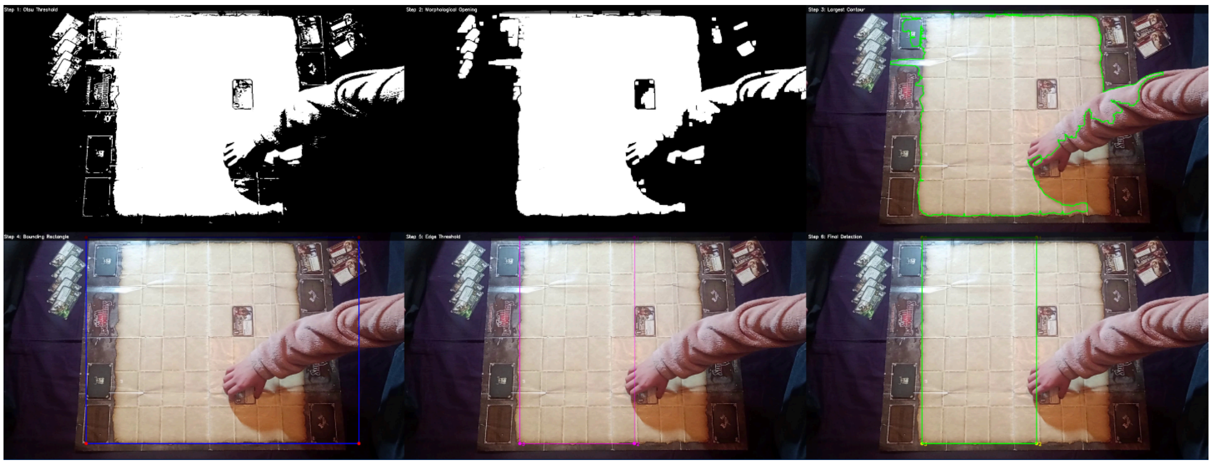
Bottom and top edges of the resulting rectangle are almost approximately correct, but left and right edges are too wide, as elements of the outer board are also detected in the largest contour.
So we scan inward from left and right edges in 5-pixel steps, and analyze areas spanned by dx and rectangle y coordinates, finding where white pixel density reaches 80%. Then we stop and return corrected x coordinates.

But the result is still rectangle, we want to also take into consideration situations where the board is shown from some angle.
So we adjust both edges simultaneously by moving left corners outward/inward and right corners inward/outward, continuing until the percentage of white pixels stops increasing.

We calculate distance between next detected corners and mean of 10 last detected corners. To calculate distance, we first compute the average movement of 4 corners, and then we compute the sum of differences between average and movement of each corner - when detected board moves because of camera movement, every corner should move in the same direction, and sum of differences from the mean should be small, but when change in detection is because of hand entering board, two corners move and two stay in place, so this difference should be bigger in this situation.

If distance is bigger than a certain threshold, we consider detection as unreliable (due to hand interfering with board). In this case we return the mean of recent detections rather than raw frame-by-frame results. When detections are rejected repeatedly, the threshold increases multiplicatively (×1.1) to eventually accept larger movements, and get track of board back.


1. Binary Mask (Otsu)
2. After Morphological Opening
3. Largest Contour
4. Bounding Rectangle
5. After White Threshold
6. Final Detection (Refined)
Frame 200 | Board Detection: SUCCESS



Card Detection

To detect cards, we developed an approach combining edge detection, morphological operations, and watershed segmentation to isolate individual cards from the game board.

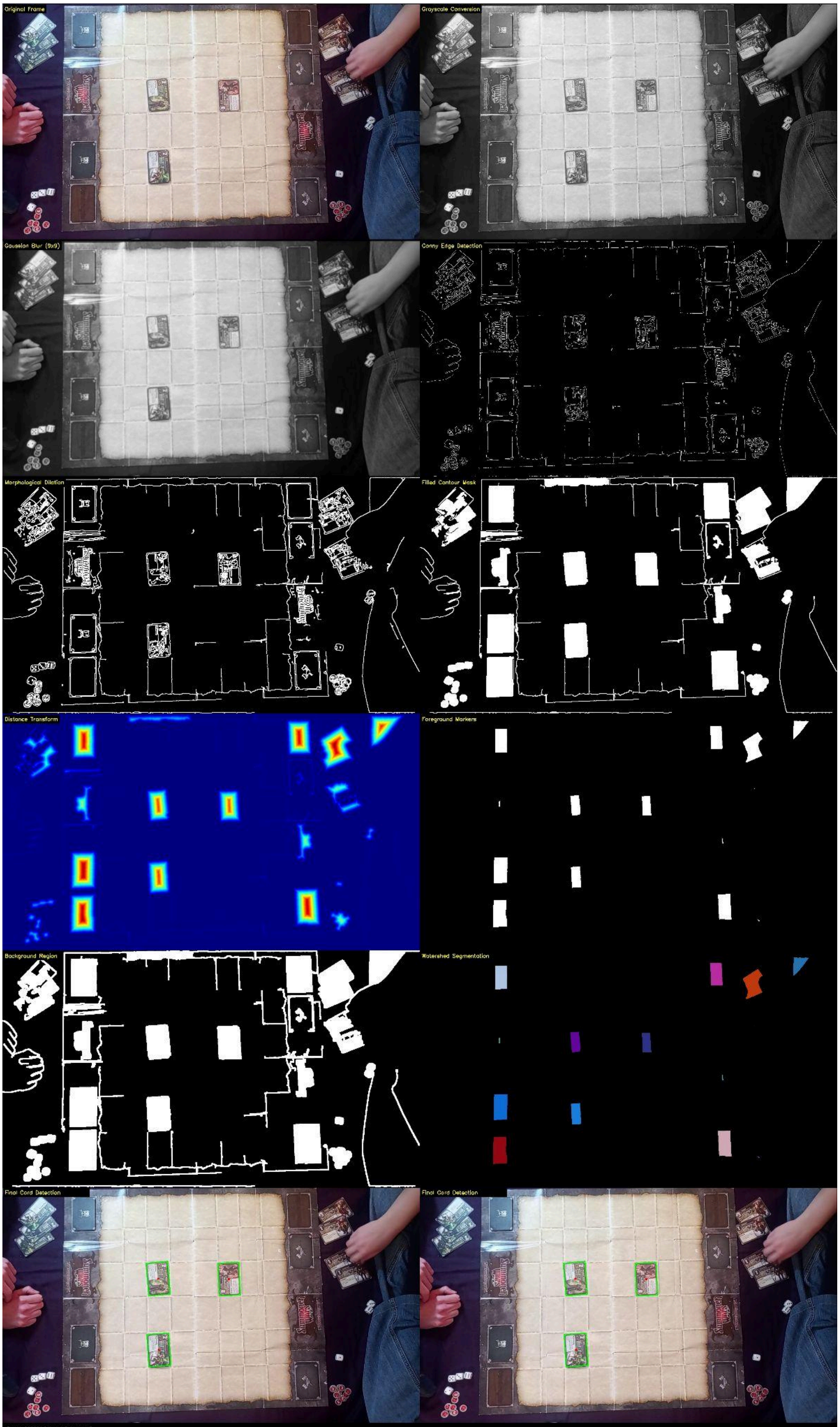
Detection Pipeline:

1. *Preprocessing*: The input frame is converted to grayscale and smoothed with a 9×9 Gaussian blur to reduce internal card details while preserving card boundaries.
2. *Edge Detection*: Canny edge detection (thresholds: 50, 150) identifies card contours, followed by dilation with a 3×3 rectangular kernel to connect fragmented edges.
3. *Foreground Extraction*: Connected components from the dilated edges form a binary mask, which undergoes distance transform. Thresholding at 0.5 × max distance identifies confident foreground regions (card centers).
4. *Watershed Segmentation*: The algorithm marks unknown regions between foreground and dilated background, then applies watershed segmentation to separate overlapping cards.
5. *Candidate Filtering*: Detected regions are filtered by area (10,000-15,000 pixels), aspect ratio, to make sure that they are rectangles (1.3-2.4, after ensuring height > width), and final filtering using ORB keypoint detection (minimum 15 keypoints) to distinguish cards from empty grid cells which just have visible boundaries.

Tracking: Detected cards are matched to existing tracks using IoU (threshold: 0.3) with KCF trackers. New detections initialize new tracks with team assignment based on board center position. Cards are only instantiated if they lie within the board boundaries and don't overlap existing tracks.

If only the detector sees a card, the system treats it as a new object, assigns it a new ID, and starts a new tracker. If only the tracker sees it, the system keeps the card alive using the tracker’s prediction so the card doesn’t disappear due to a temporary detection failure. If neither can see it, the tracker waits for 15 frames which can happen when the card is fully covered by the hand and then removed and the card is considered gone.

Figure shows the intermediate steps of card detection, visualizing the progression from raw input through edge detection, morphological operations, distance transform, and final segmentation.


Original Frame
Grayscale Conversion
Gaussian Blur (9x9)
Canny Edge Detection
Morphological Dilation
Filled Contour Mask
Distance Transform
Foreground Markers
Background Region
Watershed Segmentation
Final Card Detection
Final Card Detection

Battle Detection

Battles are detected geometrically when opposing team cards within close proximity (when they are on the neighbouring cells on the grid). In practice, we detect the battle when two cards (of opposing teams) bounding boxes' expanded by a 10% buffer of card size intersect. Each battle maintains an initial center position and a dynamic bounding box encompassing both participants.

Battle Resolution: The system tracks which card leaves the battle zone first by monitoring card centers against the original battle bounding box. When a card exits or disappears, the remaining card is declared the winner.

Dice Detection

We make use of detected board (1), we expand it and binarize (2).

We make the first binary mask (3) using the Otsu threshold multiplied so that we detect more white pixels. Then we perform closing on this mask (4).

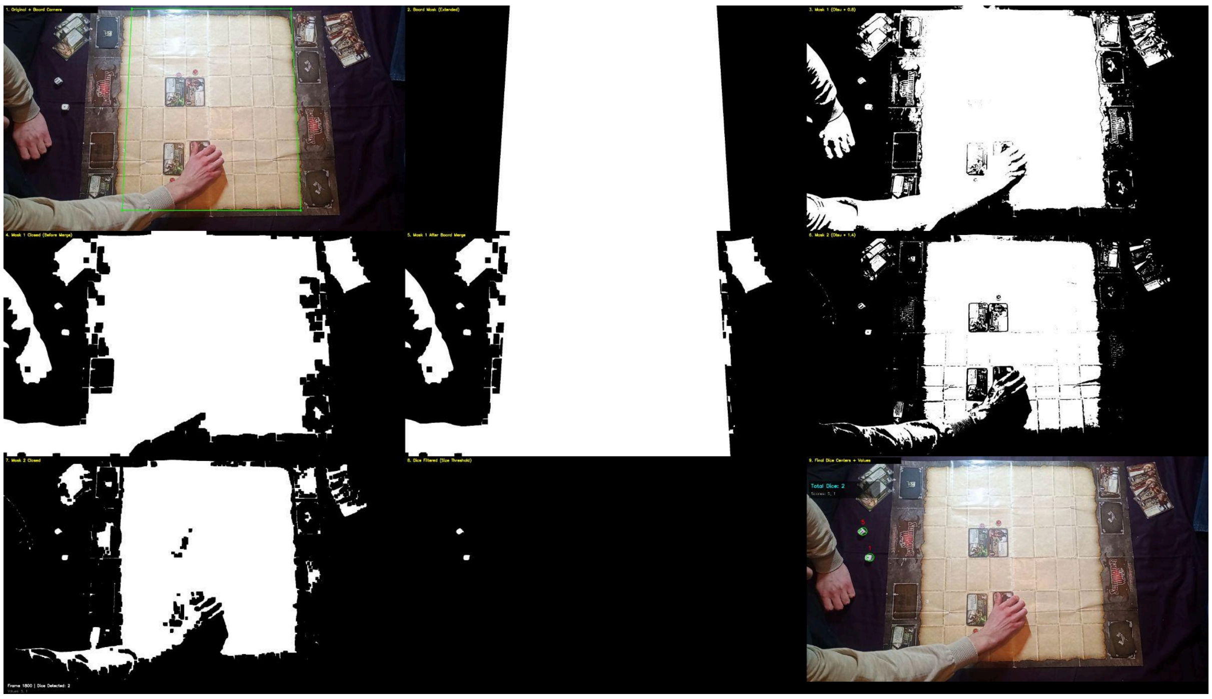
Next, we perform OR operation on the first mask and expanded detected board (5).

We make an image descriptor - for each connected component of result of (5), we calculate the size of it. Then to each pixel of this connected component, we write the size of the connected component (so black pixel will get value 0 in our descriptor).

We make a second binary mask, detecting less white, also multiplying Otsu result (6), and we perform closing on it (7).

Then we apply a descriptor to result, we leave as white only those pixels, where values of the descriptor do not exceed a certain threshold. This way we are left only with pixels that are very bright (as they were white in the second mask), and are not a part of a bigger bright object (8).

Each left element is detected as a die (9).


1. Original + Board Corners
2. Board Mask (Extended)
3. Mask 1 (Otsu + 0.8)
4. Mask 1 Closed (Before Merge)
5. Mask 1 After Board Merge
6. Mask 2 (Otsu + 1.4)
7. Mask 2 Closed
8. Dice Filtered (Size Threshold)
9. Final Dice Centers + Values
Total Dice: 2
Scores: 5, 1
5
1
Frame 1800 | Dice Detected: 2

Then we consider region in certain radius from center of detected die, we compute black regions in binary image in this region - this is our score on a die (-1 for background)

We keep track of dice detected in previous frames, and show only those that were detected in several consecutive frames in similar places. The score shown on video is a rounded average of the few last detected scores.

Token Detection

Token detection targets small circular red objects appearing above or below the bounding boxes of active battles, indicating hits during the battle.

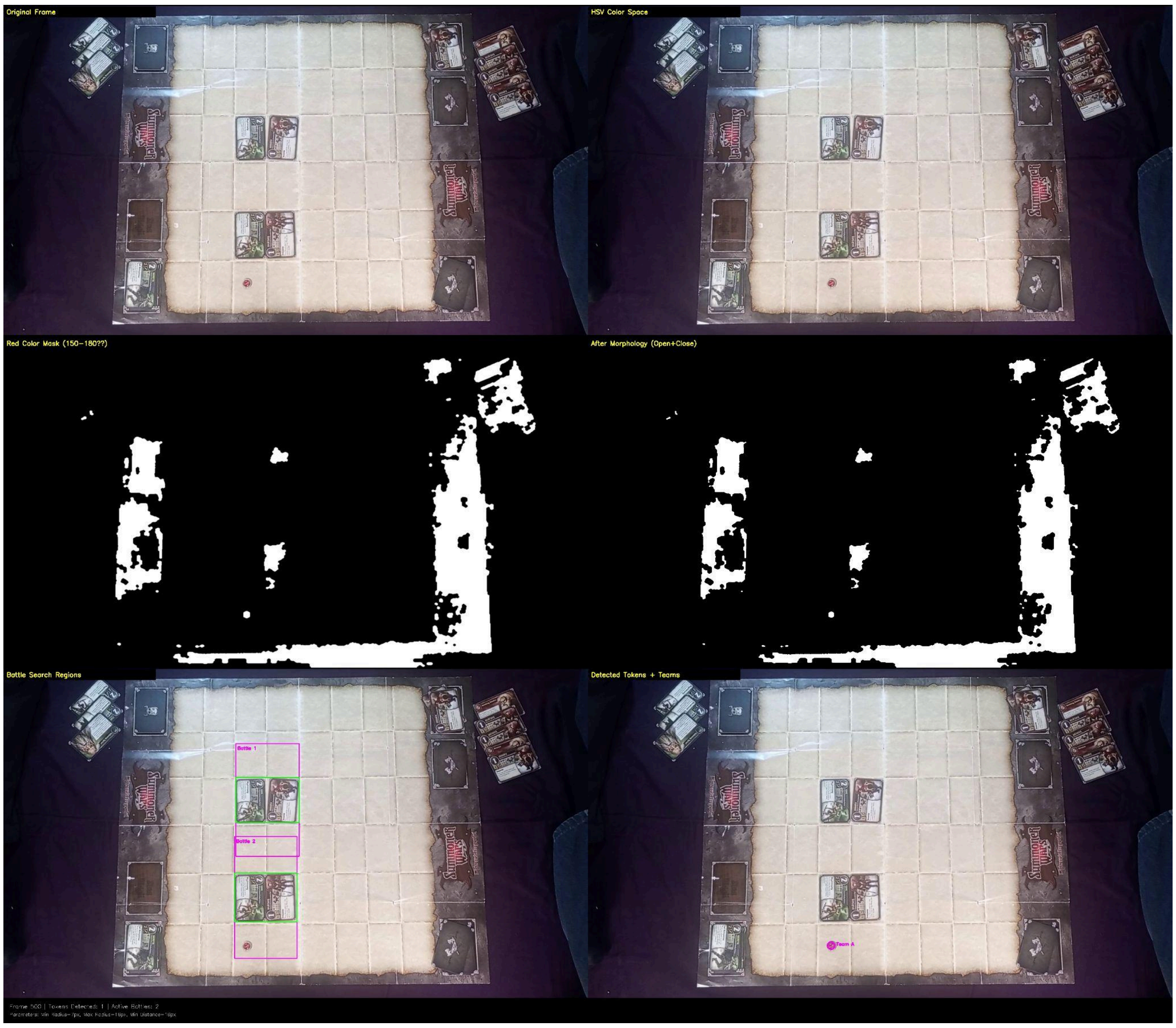
Detection Pipeline:

1. *Color Segmentation*: HSV color space enables red detection using a single range (150-180° hue, 30-255 saturation, 30-255 value).
2. *Morphological Operations*: Opening (5×5 elliptical kernel, 1 iteration) removes noise, followed by closing (7×7 elliptical kernel, 2 iterations) to fill gaps, and dilation (5×5 elliptical kernel, 1 iteration) to enhance token regions.
3. *Gaussian Blur*: A 3×3 blur smooths the mask before circle detection.
4. *Hough Circle Detection*: Applied within battle-specific search regions (extending 75% of battle height above and below, excluding the battle box interior), with radius constraints scaled to board width (0.9-2.0% ratio). Minimum inter-circle distance (2% of board width) prevents duplicate detections.

5. *Team Assignment*: Tokens are assigned to teams based on their horizontal position relative to the battle center (left = Team A, right = Team B).

Tracking and Confirmation: Detected tokens are immediately tracked using KCF trackers with a matching distance threshold of 20 pixels. A token becomes "confirmed" after stable tracking for 15 consecutive frames, at which point it increments the hit count for the corresponding card. This confirmation threshold prevents false detections when putting the token on the board - without it a lot of false detections were made, as they were indicating the trajectory of the object when putting. Tokens lost for more than 10 frames are removed from tracking. When battles end, all associated token tracking data is cleaned up, so even when the token is still on the board (player does not need to remove it), the token will no longer be detected.

Figure illustrates the token detection pipeline, showing HSV conversion, dual-range red masking, morphological operations, blur application, search region definition (above and below battles), and final circle detection with team classification.


Original Frame
HSV Color Space
Red Color Mask (150-180??)
After Morphology (Open+Close)
Battle Search Regions
Battle 1
Battle 2
Detected Tokens + Teams
Team A
Frame 500 | Tokens Detected: 1 | Active Battles: 2
Parameters: Min Radius=7px, Max Radius=16px, Min Distance=16px

System Integration

The complete detection system is run through the main processing loop in run.py, which integrates all mentioned detection modules into a pipeline:

1. **Board Detection:** The board detector calibrates on the first frame to establish the playing field boundaries and center line (for team assignment). It continuously tracks board corners throughout the video using smoothing and validation to maintain stable boundaries.
2. **Sequential Detection:** For each frame, the system performs detections in a specific order:
    - Board corner detection and validation
    - Card detection and tracking (filtered by board boundaries)
    - Battle detection from opposing card proximities
    - Token detection within active battle regions
    - Dice detection within board boundaries
3. **Event Tracking**: A dedicated *GameEventTracker* monitors state changes across frames, tracking card appearances, movements, and battle outcomes. It maintains a cooldown mechanism to prevent duplicate event reporting for cards that recently lost battles.
4. **Visualization Pipeline:** Multiple visualization functions render the detected elements with color-coded overlays, drawing boards, cards, battles, tokens, dice, scoreboards, event logs, and processing statistics for each output frame.

The effectiveness for each dataset

The results for our video dataset were mostly correct, failing on minor things rather than important events for game logic.

The full performance can be seen on the uploaded videos using following Google Drive link, where folder output contains detected gameplay using our architecture:

Analysis and conclusions of the obtained results

Our approach managed to correctly detect the most crucial events for the gameplay. It means that we correctly track cards, score and tokens played.

Some minor issues which should be improved are the dice result which sometimes is not totally accurate (six is often misled with five), but it does not influence the result from that, as in the Summoner Wars the threshold is used for determining whether we attack with a token or not, so either 5 or 6 leads to the same result - token awarded. Sometimes we also incorrectly detect as dice things not being dice, but only when we lose track of board.

Another thing is board and card detection when hands enter a board, namely we sometimes lose track for a few frames of board or cards, however we implemented several fixing ways, so the previous states are remembered, and short hand contact does not influence the changing in card ID, or breaking the board grid.

Overall, the detection process worked successfully especially on easy and medium datasets, the biggest difficulty was hard dataset, where shaking, and irregular camera movements made it much more difficult to detect everything spotlessly.