# Project 2 Report: Game events and status recognition

Authors:

Maria Musiał 156062

Martyna Stasiak 156071

Chosen Game: Super Farmer

Link to the google drive containing original recordings and our outputs:
https://drive.google.com/drive/folders/1MIxwzXAEbxG8m_RI3XE0KX67lqT8TK-d?usp=sharing

## Introduction

The goal of this project was to develop a system capable of recognising the state of gameplay and events from video clips of the board game of choice. We have picked the game Superfarmer.

The primary objective was to detect and track key in-game events, such as dice rolls, animal trading, predator attacks, buying a dog and winning the game. By analysing the video data, the system tracks the movement of game's dice, while also identifying and displaying relevant events, such as the acquisition of new animals or the occurrence of predator attacks. This approach allows for real-time gameplay analysis and event detection, providing a more interactive and automated experience of the Superfarmer game.

## Game's description:

Superfarmer is a strategic board game that supports from 1 to 6 players who manage their own farms. The aim is to collect all of the animals (rabbit, sheep, pig, cow and horse) drawn on the board. Players roll a die to obtain the animal but they might also roll out a predator (fox or a wolf) that eats the player's animals. They may also obtain an animal b ytrading the ons that they already have.

# In-game events

1. Rolling the dice: The player rolls both dice to determine which animals they take or if predators attack.
2. Trading animal tokens: Player can trade animals with the bank for protective dogs and other animals.
3. Predator attack: If a wolf is rolled, the player loses all animals except small dogs unless they have a big protective dog.
4. Winning the Game: Player wins the game if the whole board- their farm is filled with the animals.
5. Getting a protective dog: Player may trade their animals for either small or big dog, ensuring protection from the predators.

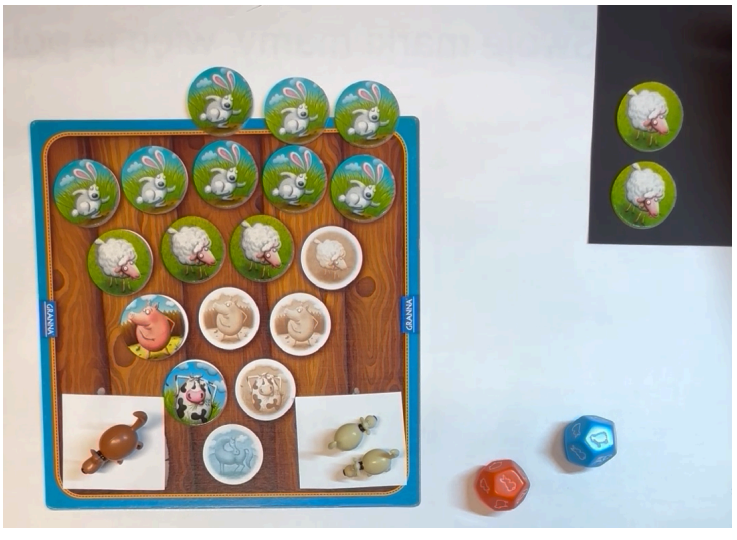
# In-game items:

1. Player's farm board
2. Animal tokens (rabbit, sheep, pig, cow, horse)
3. Two 12-sided dice with symbols representing the animals (both the farm ones and the predators)
4. Protective dogs

# Dataset Description

The dataset for this project consists of 9 video clips captured from our real gameplay of *Superfarmer*. The clips are divided into three difficulty levels:

- **Easy**: This group consists of three clips with the best conditions. The gameplay elements are visible, with minimal hand obstructions or shadows. The camera is stable and provides a clear view of the game space from above, without shaking.



- **Medium**: These three clips have more difficult conditions. In them the light varies between the frames, the light reflections are present as well as the magnified shadows, created by turning light off partially.


(shadows and light reflection on sheep)

- **Difficult**: The three clips in this category present the most complex challenges. They include camera shaking or shaking the board and varied lighting conditions as in the Medium dataset. Also, the camera in some of the videos is angled, making it more difficult to track the positions and movements of game pieces.


Each video clip has at least one event, whereas in each dataset all five events are present.

# Techniques

1. *Board Detection*

   1.1 Preprocessing: The input frame is converted to grayscale and smoothed using Gaussian blur to reduce texture noise.

   1.2 Edge Detection: Canny edge detection highlights strong edges, followed by morphological closing to close gaps in contours.

   1.3 Contour Detection and Filtering: Contours are identified and approximated as polygons using cv2.findContours, cv2.approxPolyDP with epsilon thanks to which we can detect the board at an angle. Polygons with four vertices are considered as potential boards.

   1.4 Validation: The detected polygons are reordered and validated based on aspect ratio, area, and bounding box size to ensure they match the board's expected dimensions. If a valid board is found, its coordinates and dimensions are returned. Otherwise, default values indicate no board detected.

2. *Empty Animal Spaces (used for determining Final State)*

   2.1 Converting frame to gray, then putting it through Gaussian blur (this help tremendously with texture on the board)

   2.2 Detecting Circles: Using HoughCircles with parameters of size determined from detected board and param1, param2 that catch the mostly white spaces

   2.3 If the number of empty circles is 0 for some time, it triggers “Game won” event

3. *Animal Token Detection*

   3.1 Converting frame to gray, then putting it through Gaussian blur (this help tremendously with texture on the board)

   3.2 Detecting Circles: Using HoughCircles with parameters of size determined from detected board and param1, param2 that detects all animals in the frame

   3.3 Stabilizing Circles: To improve detection reliability, the code stabilizes the detected circles by comparing them to circles detected in previous frames. If the new circle is within some threshold from the previous, it isn’t taken into consideration.

3.4 Extracting and Matching Regions: For each detected circle, the corresponding region of the frame is extracted. This region is then compared to a set of predefined templates for animals using template matching to identify which animal it represents. We resize the region to match the template and take the match with the highest value. This method requires the tokens to be put all in the same angle. Which is our key assumption.

3.5 Excluding the empty circles from consideration. If the circle detected is close to the center of any of the empty circles, it's skipped.

3.6 Avoiding False Positives: The code excludes regions that overlap with previously detected empty circles to avoid false positives and ensures that detected circles match the expected size and location criteria.

3.7 Updating state of the game by counting the animals chosen.

4. *Trading*

4.1 Extending the Board Area: An extended rectangular region around the detected board is defined to monitor for animal tokens that move outside the board. This region includes a buffer zone to capture potential trading events.

4.2 Monitoring Token Movement: For each detected animal token, its position is checked relative to the extended board area. If the token moves outside this area, it is flagged as a potential trading candidate.

4.3 Tracking Out-of-Board Tokens: A tracking mechanism records the positions of tokens outside the board. If a token remains outside the board for a certain number of frames, it is confirmed as a trading event.

5. *Detection of getting a dog*

5.1 Defining Regions of Interest (ROIs): Two specific regions in the corners of the board are defined to monitor the dogs: the right ROI for small dogs and the left ROI for large dogs. These regions are positioned relative to the board dimensions.

5.2 Detecting Circles: The Hough Circle Transform is applied within each ROI to identify circular regions that potentially represent dog figurines.

5.3 Tracking Detection History: A stack tracks if there were any circles (potential dogs) detected in a number of frames.

5.4 Identifying Dog Acquisition: The detection history is evaluated, and if the ratio of positive detections exceeds a threshold, the corresponding dog token is marked as acquired. This triggers event of getting a dog

6. *Wolf detection*

   6.1 Detection from animals: When detecting the state of animals, we keep track of those across frames using stack. We analyze the stack. If the number of animals is 0, and there were animals in the previous frames, it tells us that there was a wolf attack

   6.2 Detection from a dog: When detecting a dog, we keep a track of big dog across frames. If the stack is 0, while in previous frames had the dog, this tells us the dog was eaten by wolf

   Both those detections have some tolerance in analysing the stacks to ensure the least possible false positives.

7. *Dice tracking*

   7.1 Excluding the board from consideration by masking it. Board contains a wood texture that is very similar to the orange dice.

   7.2 Hardcoding color of a dice with some tolerance. Converting frame to HSV color. Getting a mask for orange and blue using inRange(). In the mask we use findContours to get dices.

   7.3 Thresholds for contours: Discarding contours that are too big/too small determined by area. We also detect if the detected dice is within some distance to the detected animal, as they have similar color in some light. Last threshold is determining if the contour is square-ish.

   7.4 Tracker KCF is initialized for each new dice. We track time of initialization, to detect Dice Rolled event after 2s.

# Effectiveness on datasets

We were able to detect all of the events on all of the datasets, but as the level of difficulty increased we have encountered more problems, mostly with the false positives regarding the detection of the animal tokens.

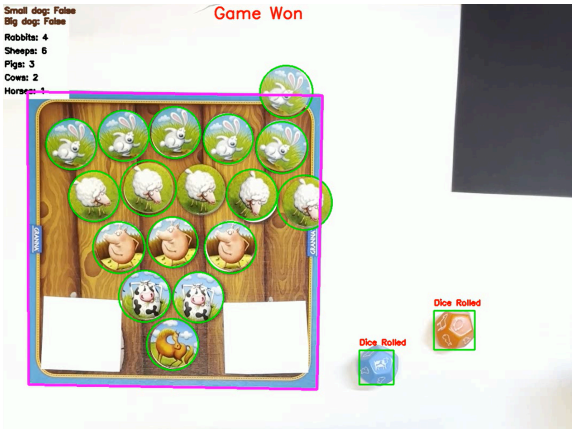
- *Easy Dataset*

  In this dataset we had almost perfect conditions with good light, the only obstructions were our hands that moved the animal tokes on the board creating some shadows.

  - oE1 detected events:

        - trading,
        - dice roll,
        - predator attack - wolf was rolled twice and first ate the big dog and then cleared the board.
    - oE2 detected events:
        - dice roll,
        - trading,
        - getting a dog,
        - predator attack - wolf was rolled and cleared the board leaving only small dog
    - oE3 detected events:
        - dice roll,
        - trading,
        - getting a dog,
        - finishing the game (winning)

In these datasets we had some problems with false positives regarding the detection of animal tokens on board; sometimes, for a short period of time, the empty space was considered as an animal. Also when the pig was outside the board it was rarely detected not only as an animal but also as a dice because of its colour.

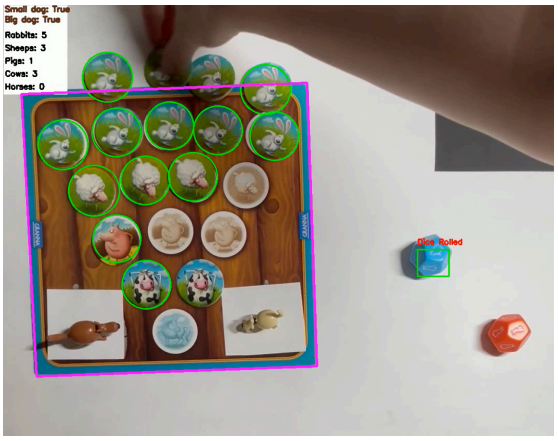

- *Medium Dataset*

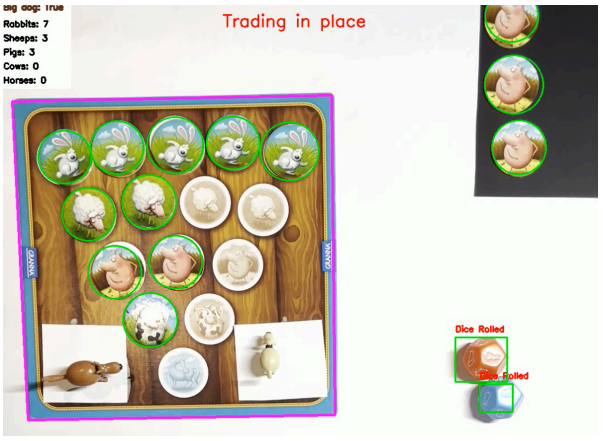
  In this dataset, we had much worse light by which there were more shadows and in some moments the light was also flickering and changing the tones- from white light to warm, and changing its intensivity. Also in the last video we have flickering light reflections done by moving the flashlight above the playing area.

    - oM1 detected events:
        - trading,
        - getting a dog,
        - dice roll,

        - finishing the game (winning)
    - oM2 detected events:
        - dice roll,
        - trading
    - oM3 detected events:
        - dice roll,
        - trading,
        - getting a dog,
        - finishing the game (winning)

In this dataset we had the same problems as in the previous one, but also sometimes because of the changing light intensity and reflections the dice were not detected properly. Additionally because of the shadows the empty spaces were sometimes detected and counted as animals.



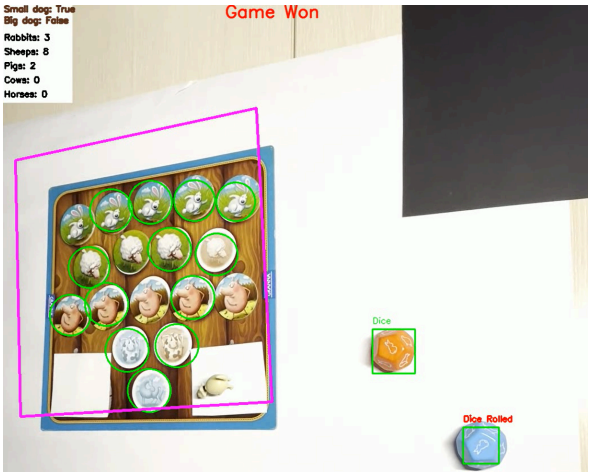




- *Hard Dataset*

    In this dataset, we had same light conditions and shadows as in the mediun dataset, but also now we had moving board and the camera was at an angle.

    - oH1 detected events:
        - dice roll,
        - predator attack - wolf was rolled and ate the big dog,
        - trading
    - oH2 detected events:
        - trading,
        - dice roll,
        - getting a dog,
        - finishing the game (winning)
    - oH3 correctly detected events:
        - dice roll,
        - trading

In this dataset we had the same problems as in the previous ones, but additionally due to the playing area movement there was a problem regarding the detection of dogs which indicated being also attacked by a wolf. Another problem occured when we chose too big angle to record the game (oH3) and because of that the empty animal spaces were being identified as the animals so the event of winning the game was being shown all the time.


# Analysis and conclusions of the results

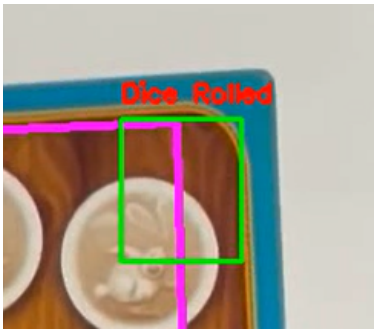
Our system successfully detects the game events and animals across all datasets, however the accuracy of the detection drops as the difficulty of the datasets increases, due to the changing lighting and the movement.

In the Easy Dataset, the detection performance was highly accurate due to the good lighting and minimal obstructions. The events were identified correctly most of the time, the only problem was with the occurrences of falsely detected animals on the board since the places on board for them look highly similar. Also, the pig token when placed outside of the board was sometimes not only detected as an animal but also as a dice since it has a similar colour as the dice. The protection that discarded dices where circles are should have been also done the other way around.

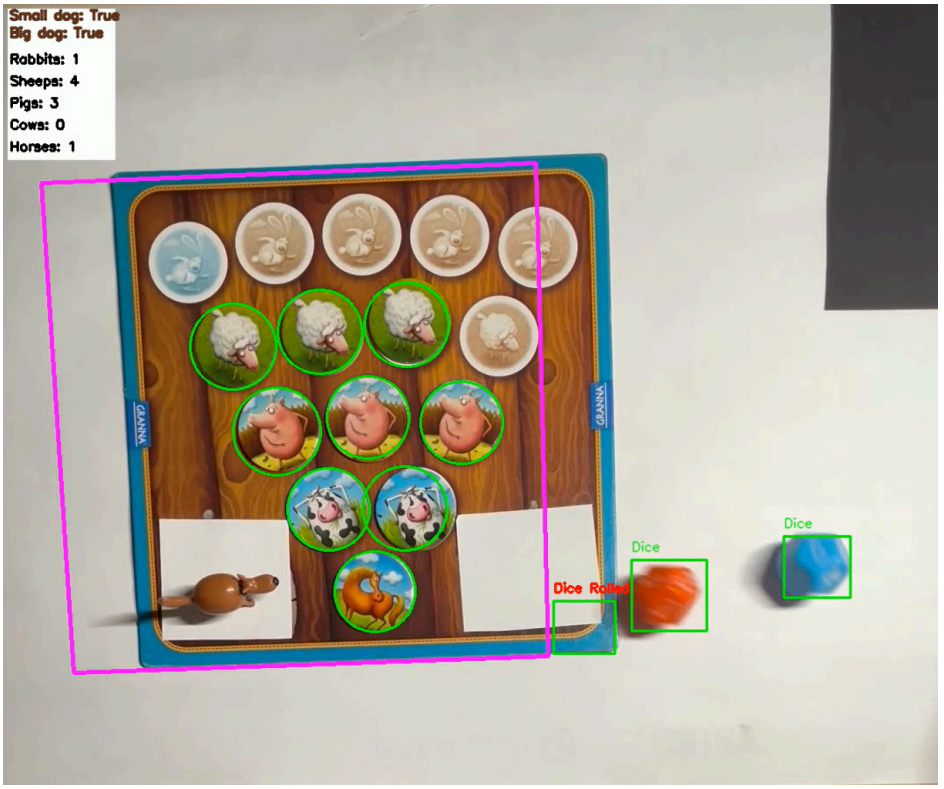

In the Medium Dataset the performance and detection accuracy did not drop significantly, but due to the lighting changes caused by the shadows the frequency of falsely detected animals has risen. Additionally the reflecting light caused the dice to not be always detected, as it was out of HSV determined bounds.

In the Hard Dataset due to the moving camera, the board was determined at the wrong position. Almost all detection is determined in regard to board position except dice detection, so functions had problems. To go over it:

- Dogs were falsely detected, as corners of the board were not actualized.
    - This led also to false wolf detecting (big dog disappearance).
- Dice was detected inside the board (the wood texture caught as orange dice).
- Empty spaces were not detected correctly, as the camera saw too little white, too much brown in this setting. This leads sometimes to false winning of animals being detected on empty spaces.

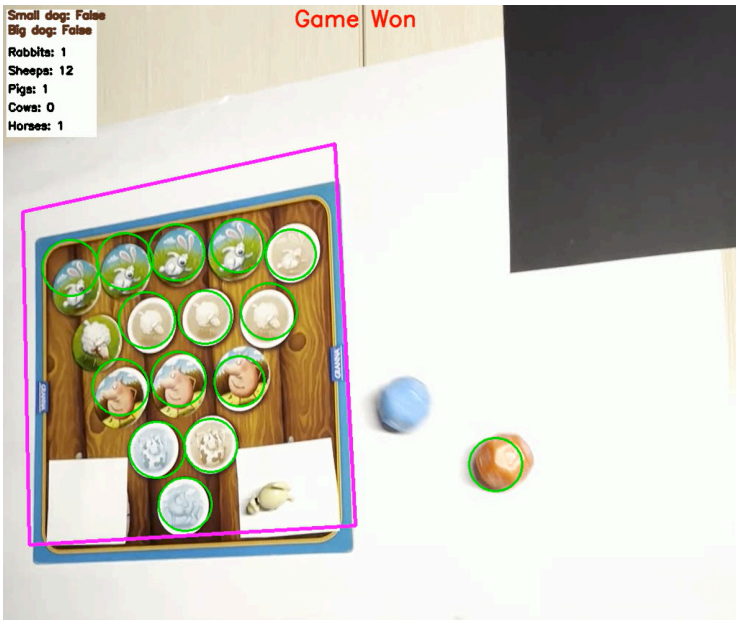



Dice detected inside a board




Small dog detected (state true)


Wolf attract (big dog not determined well)




Game won (empty spaces not detected well)

Trading worked well in those conditions, as in no setting we moved board in frame so much that it went over the board+threshold defined.


Trading working well

This all was a big problem we tried addressing with updating the board position. Unfortunately, we couldn't come up with a solution for it. Tried methods:

- Detecting corners to track left bottom corner didn't work, as our board has not enough sharp edges
- keypoint extraction didn't work, as dog or animals on board changed the keypoints too much in a gameplay
- Recalculating by the polygons didn't work either, as our gameplay has putting animals on a border. This messes up the shape too much. In fact, a shadow of a dog's tail was a problem in a medium dataset even (fortunately we were able to parametrize detection enough for it to not be a problem)

We also tried to do SWIFT/ORB/pattern matching for detecting wolf on the dice. We couldn't parametrize it well enough, or our template was not right. It was a bit hard especially due to the dice being 12-sided and the images on it not symmetric.

What can also be done is some kind of dealing with angles, distortions. But that would require at least the board being visible at all times and detectable. This would deal with animal sizes being smaller/bigger depending on fragment of frame (as can be seen in the picture above, with trading)

To sum up, our algorithm works very well for an easy dataset, the mediumdataset has some problems due to lighting, therefore colors not matching.. Hard dataset is the most challenging, and lacks board position updating the most. We can not think of any solution that would work better than what we have with the board position being so challenging to update.

Literature:

- OpenCV documentation
- StackOverflow
- ChatGPT (for faster development, not getting logic)