# Project 2 – Curling Game
# Cojocariu Sebastian - 407 IA

## Implementation details

There is a file containing multiple reusable functions called *utils.py* and three entry points for the first three tasks: *task_1.py, task_2.py, task_3.py.*

## utils.py:

- *BOUNDARIES*: HSV boundaries to detect various portion of the map: the ice, yellow_stone, red_stone, gray color, the bands between two ice surface, the scoring table, all kinds of red, all kinds of yellow, the inner circle (button) and the outer circle (the house).
- *find_hsv*: Helper function used to find HSV boundaries for each color of interest.
- *get_color_masks*: Computes masks from an image based on HSV boundaries for multiple colors. Several mechanisms to remove noise are used: erosion, dilation and other functions such as get_score_table or remove_small_areas.
- *find_circle_overlapping:* Computes how many points (1 or 255) from the mask overlapped with a given circle.
- *find_rectangle_overlapping:* Computes how many points (1 or 255) from the mask overlapped with a given rectangle.
- *get_hough_circles:* Use HoughCircle to find multiple circles. Then, an heuristics based on the percentage of red/yellow points inside the circles is further used to remove FPs.
- *get_histogram:* Computes a histogram based on the possible colors from a matrix/image.
- *get_score_table:* Computes a mask for the scoring table (we needed it because there were red/yellow stones there that could be taken as FPs by the algorithm using HoughCircles). We will match on the contour color (green-ish) using a closing operation and then find the largest component (we assume that if such scoring table exists, it will be the largest).
- *remove_small_areas*: Used to cover the small holes in the image(noise). The idea is to use a given threshold for the accepted areas and filter based on that.
- *get_map:* Used to get only the center map, without the scoring table (some photos contain 3 ice surfaces, so we want our algorithm to be limited only to the relevant ice-surface and remove the non-relevant ones). The idea is to combine the masks from the scoring_table and bands, to invert the last matrix and to compute the largest component. This should be always the center map.
- *show_images:* Helper that shows multiple images in a grid format (specified by nrows and ncols). It supports both black and white / BGR.

## task_1.py

- *task1:* function that receives the path to an image:
    - reads the image.
    - applies *get_map* on it (to get only the relevant ice-surface and remove the scoring table)
    - applies *get_hough_circles* to detect the relevant circles (using a filtering mechanism).
    - finally, it saves the result in the specified file.

## task_2.py

- *find_map_details*: Computes the center of the button (inner-circle) + the radius to the house. It combines the masks related to the button and the house. To compute the center of the button, we first find the mean coordinates of all red points (bear in mind that at this point erosion and remove_small_areas were already applied, so small holes should have been removes). Then, we only keep the red points that are inside circle with the center calcualted before and radius=200 (hyperparameter). We then find the margin red points and compute all the distances between each 2 pairs. We then choose the largest 100 and compute their mean on each coordinate. This would be saved as the center of the button. Then, to find the radius to the house, we calculate the distance from all the blue (the color of the outer circle) points, remove the outliers and compute the mean of the largest 100 remaining. This would be the desired radius.
- *algorithm:* the logic behind the curling game. It groups all the circles corresponding to the stones. Then, it filters them, keeping only the ones that touch the house. Then, these stones are sorted based on their relative distance to the center. The score will be equal to how many consecutive stones of the same color as the closest one there are (starting from the closest one).
- *task2:* it takes a video, choose the last frame and does the following:
    - applies *get_map* on the last frame (to get only the relevant ice-surface and remove the scoring table).
    - calls *find_map_details* to find the center of the button and the radius to the house.
    - applies *get_hough_circles* to detect the circles that correspond to the stones.
    - calls algorithm to compute the final score.
    - finally, it saves the result in the specified file.

## task_3.py

- *reduce_bounding_box:* It reduces the bounding box calculated by the CSRT tracker. It tries to shrink the rectangle using red_stone/yellow_stone masks. Before reducing the rectangle, we remove small areas of color from the mask (to remove noise). Additionally, it expands the calculated rectangle within an epsilon_width/epsilon_height range (to account for the gray contour for instance).
- *task3:* it takes a video, instantiates a CSRT tracker on the initial rectangle received and for each frame it does the following:
    - uses *CSRT tracker* to compute the current bounding box.
    - applies *reduce_bounding_box* to further reduce the bounding box.

- finally, it saves the result in the specified file.