

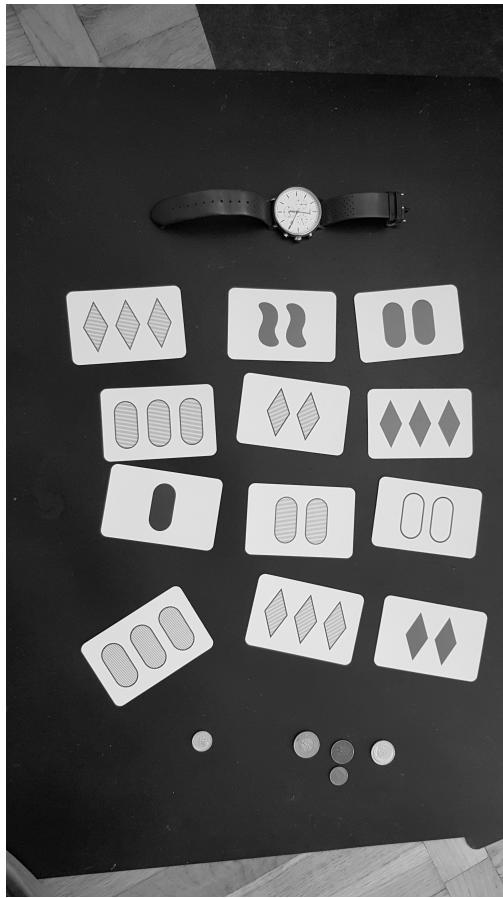
Sprawozdanie z projektu: Card Detector

Celem projektu było wykrycie i podpisanie ze zdjęcia kart, które przedstawiają figury o różnym kształcie, kolorach, liczności i wypełnieniu na białym tle. Projekt obejmował zidentyfikowanie kart ze zdjęć należących do trzech folderów różniących się poziomem trudnościami (easy, medium, hard).

Na zdjęciach "easy" karty znajawały się na jednolitej powierzchni bez żadnych dystraktorów. Na zdjęciach "medium" tło posiadało niejednolitą powierzchnię np. panele podłogowe oraz mogły się tam znaleźć dystraktory w postaci np. zegarka czy pudełka po kartach. Na zdjęciach "hard" tło bardzo przeszkadza w wykryciu kart, ponieważ jest białe albo posiada białe elementy, zlewające się z tłem kart.

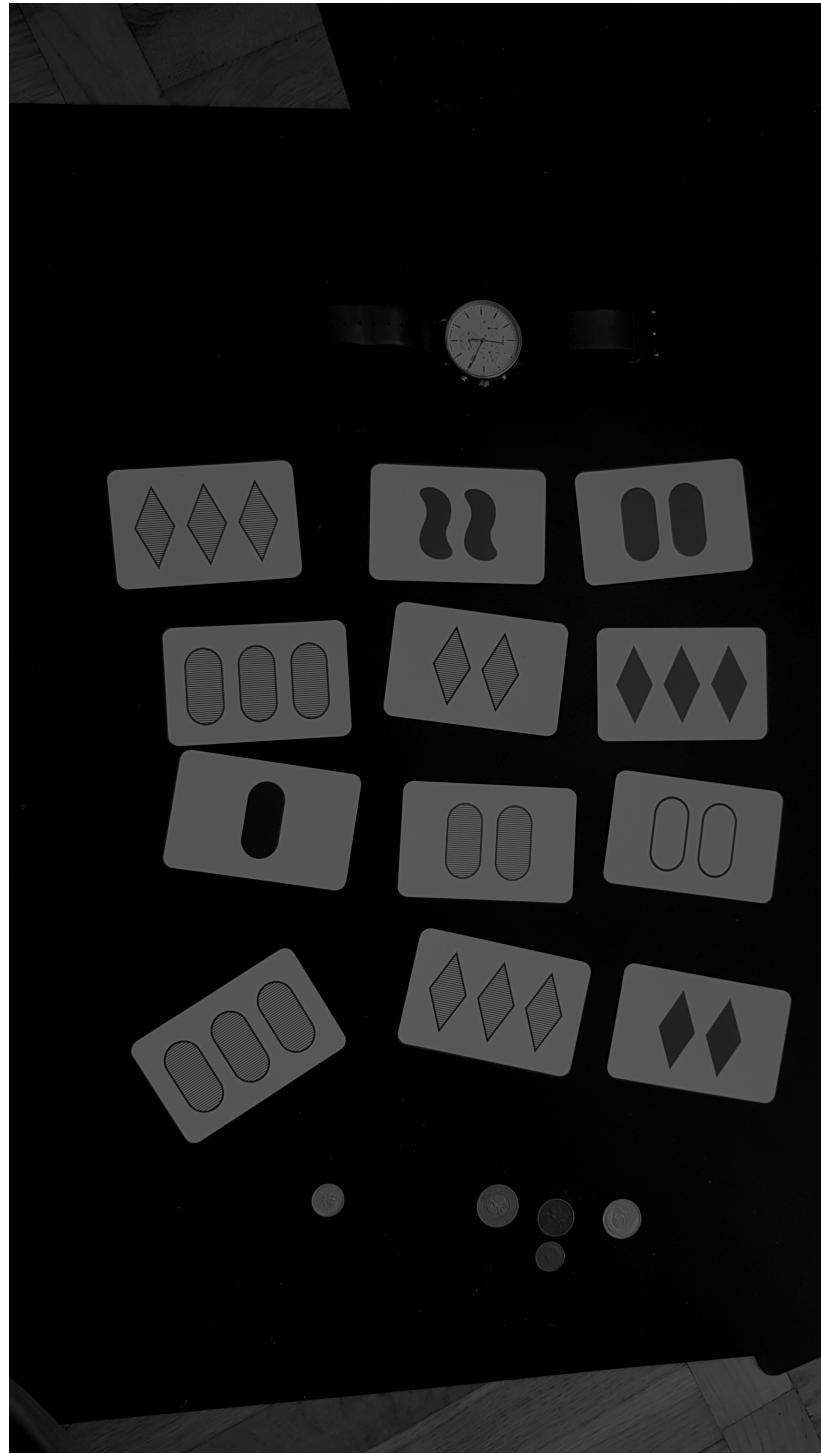
Projekt został napisany w języku Python z wykorzystaniem biblioteki OpenCV oraz Numpy. Oto kolejne etapy przetwarzania wybranego zdjęcia:

1. Po wczytaniu zdjęcia do programu z reprezentacji kolorów BGR jest ono przekonwertowane na reprezentację odcieni szarości.



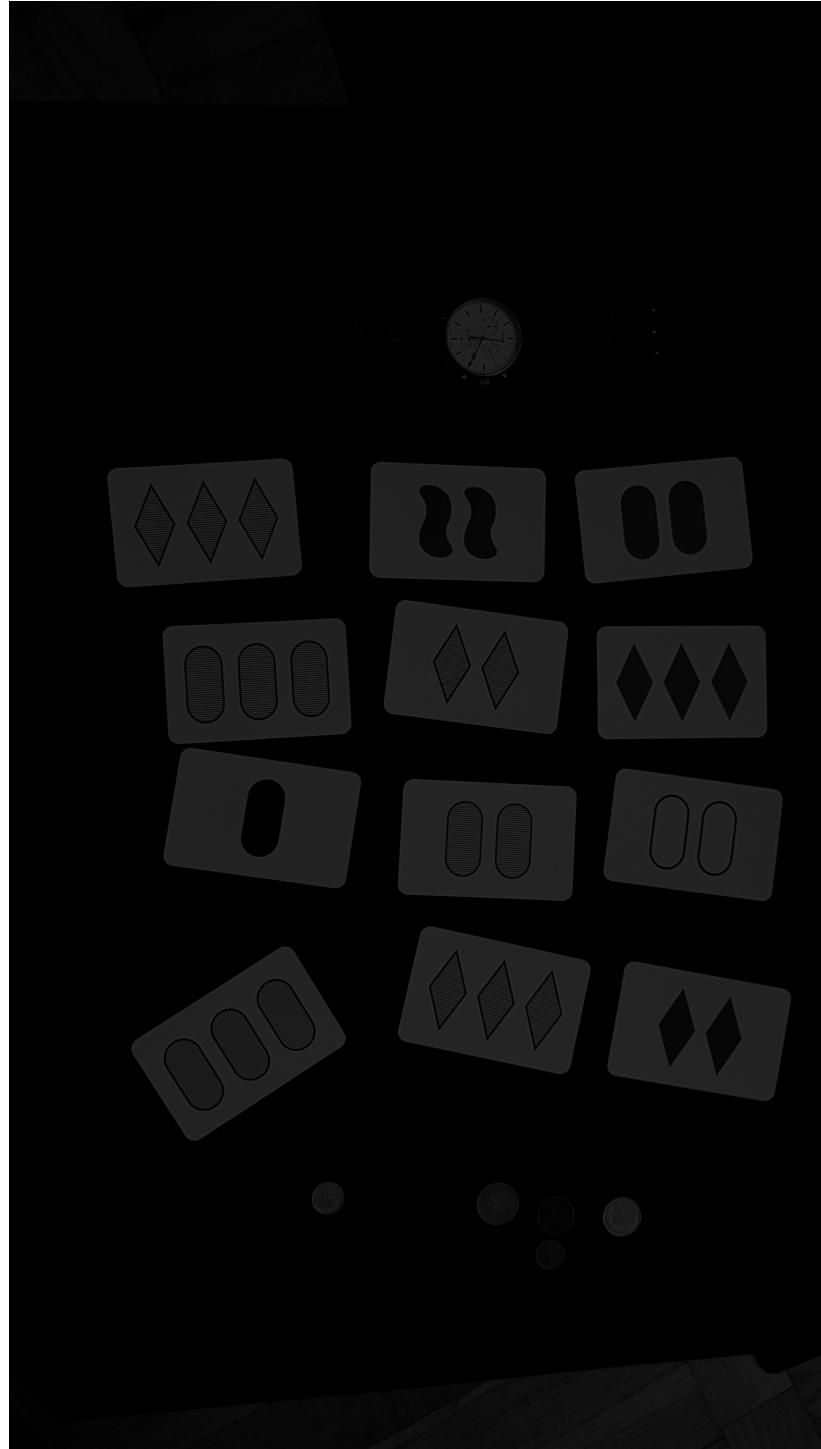
Rysunek 1: Reprezentacja odcieni szarości.

2. Następnie został zastosowany bardzo wysoki kontrast do zdjęcia, aby pogłębić różnice między białym kolorem kart, a ciemnym tłem.



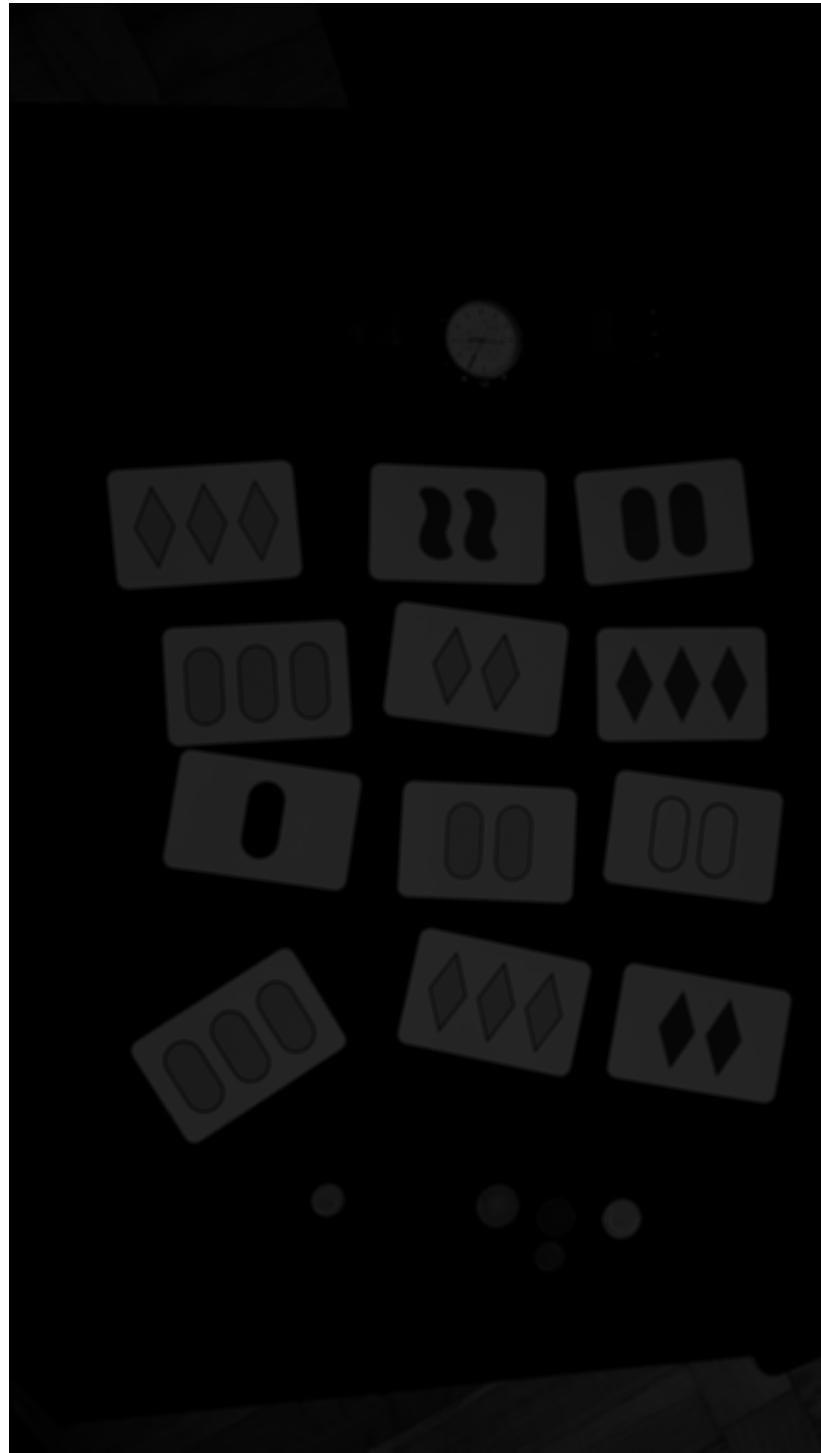
Rysunek 2: Wysoki kontrast.

3. Zastosowanie niskiej gammy, aby tło stało się jeszcze ciemniejsze.



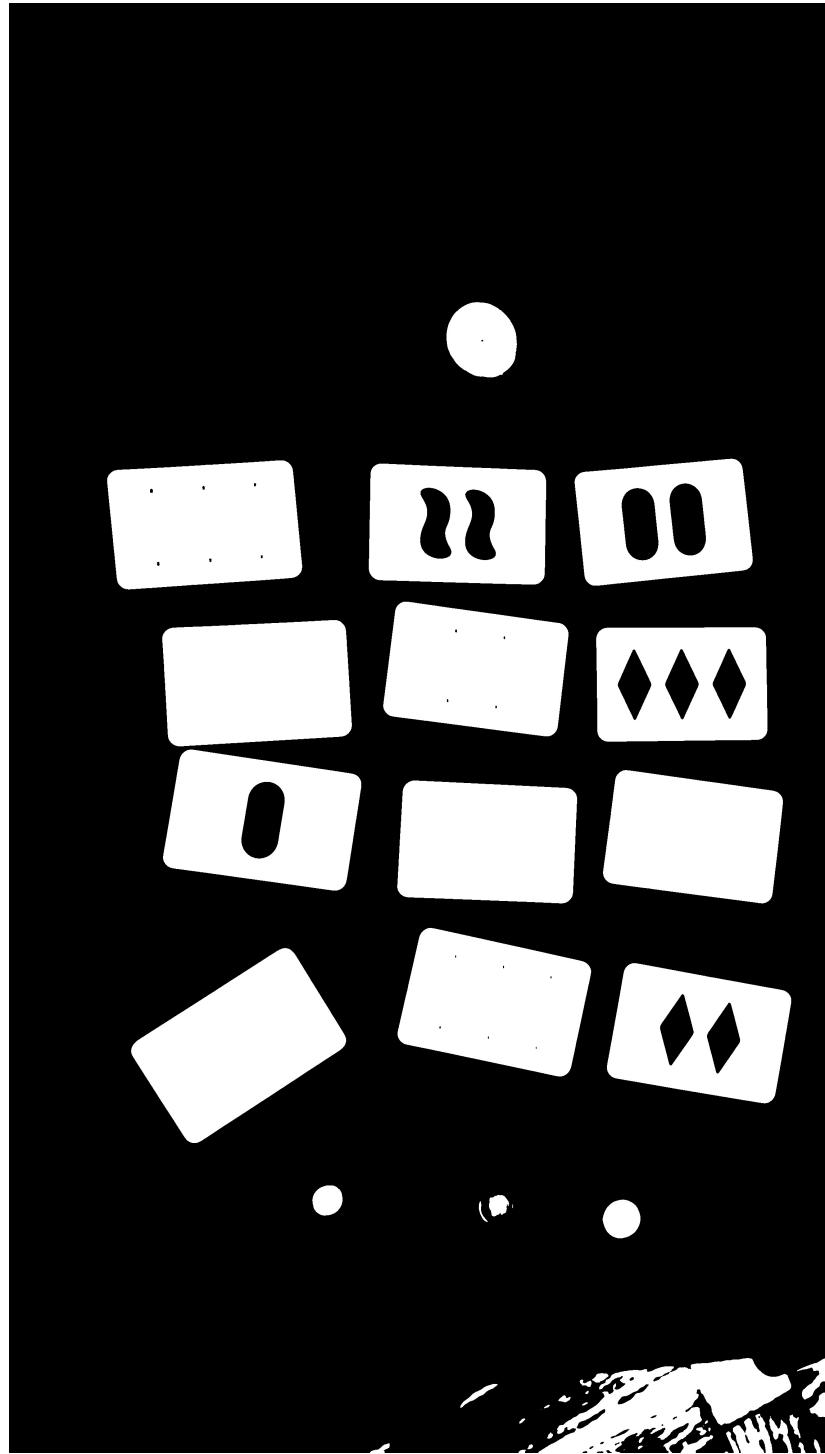
Rysunek 3: Gamma.

4. Aby wymazać niepotrzebne szczegóły zastosowany został blur.



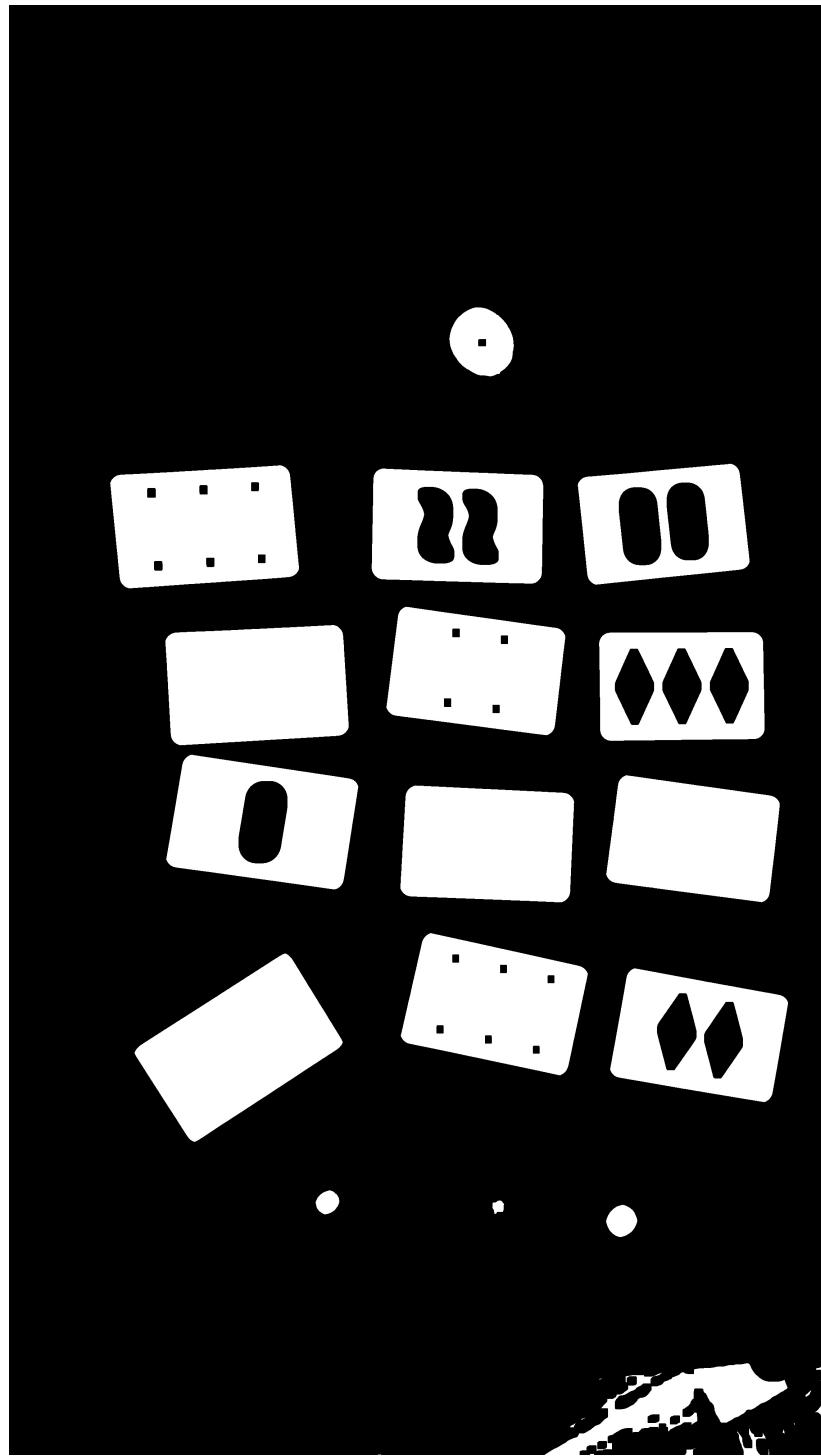
Rysunek 4: Blur.

5. Po tak przygotowanym zdjęciu można było zastosować threshold'a z Gaussowskim blurem.



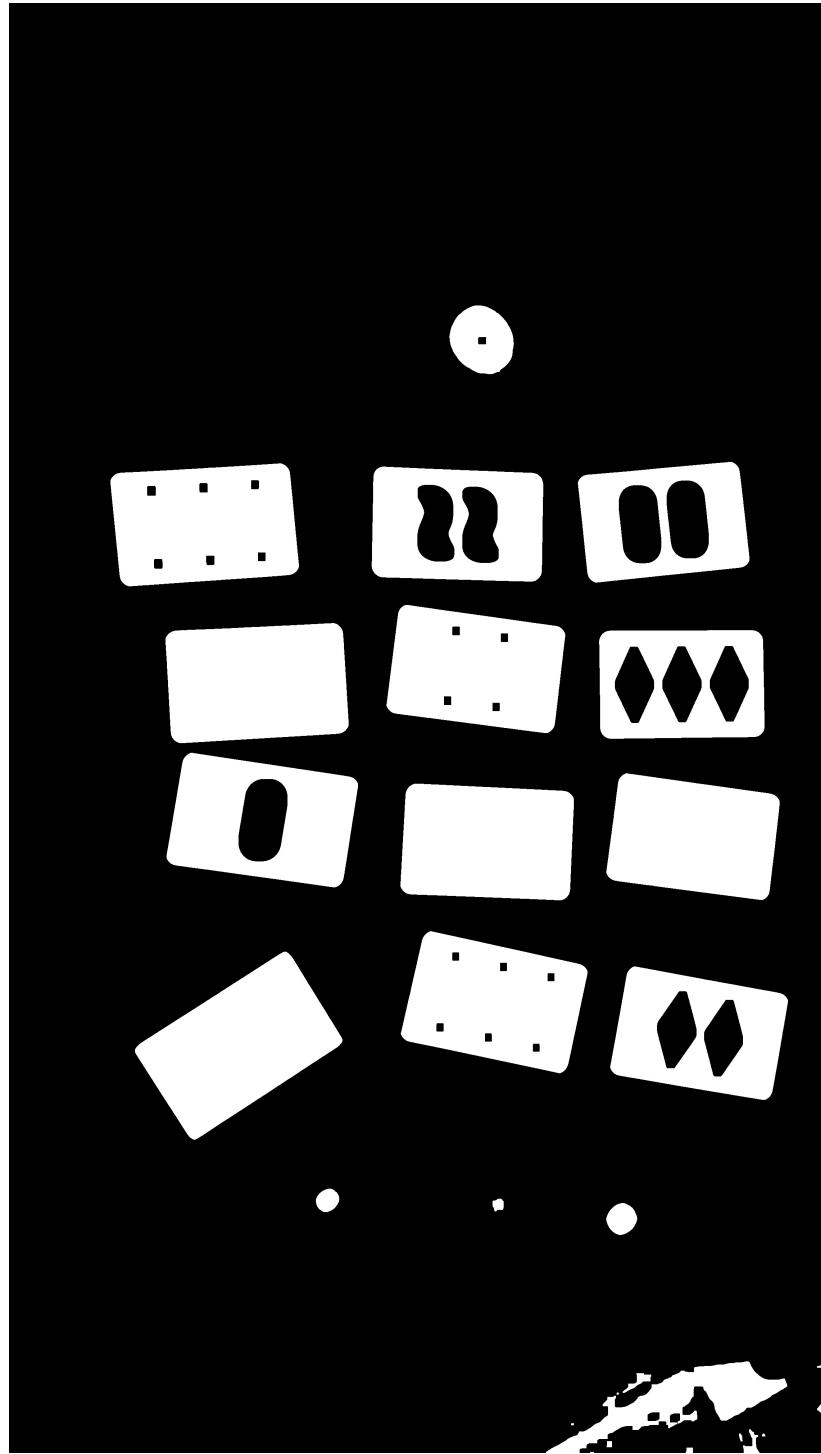
Rysunek 5: Threshold.

6. Aby pozbyć się niedoskonałości tła oraz okolicy kart wykorzystana została poczwórna erozja.



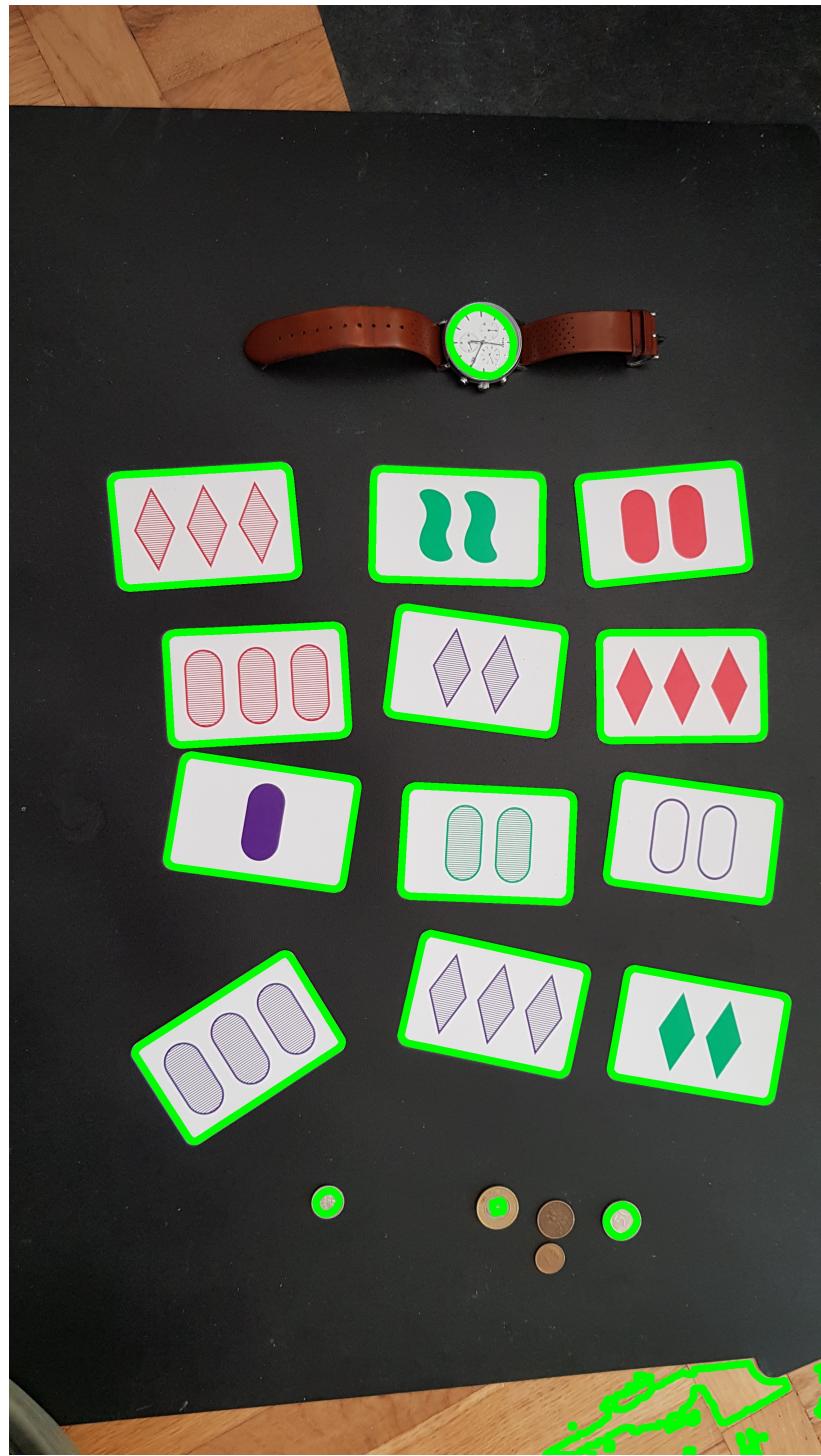
Rysunek 6: Erozja.

7. Następnie wykorzystana została operacja zamknięcia, aby poprawić model kart oraz ponownie pozbyć się niedoskonałości tła.



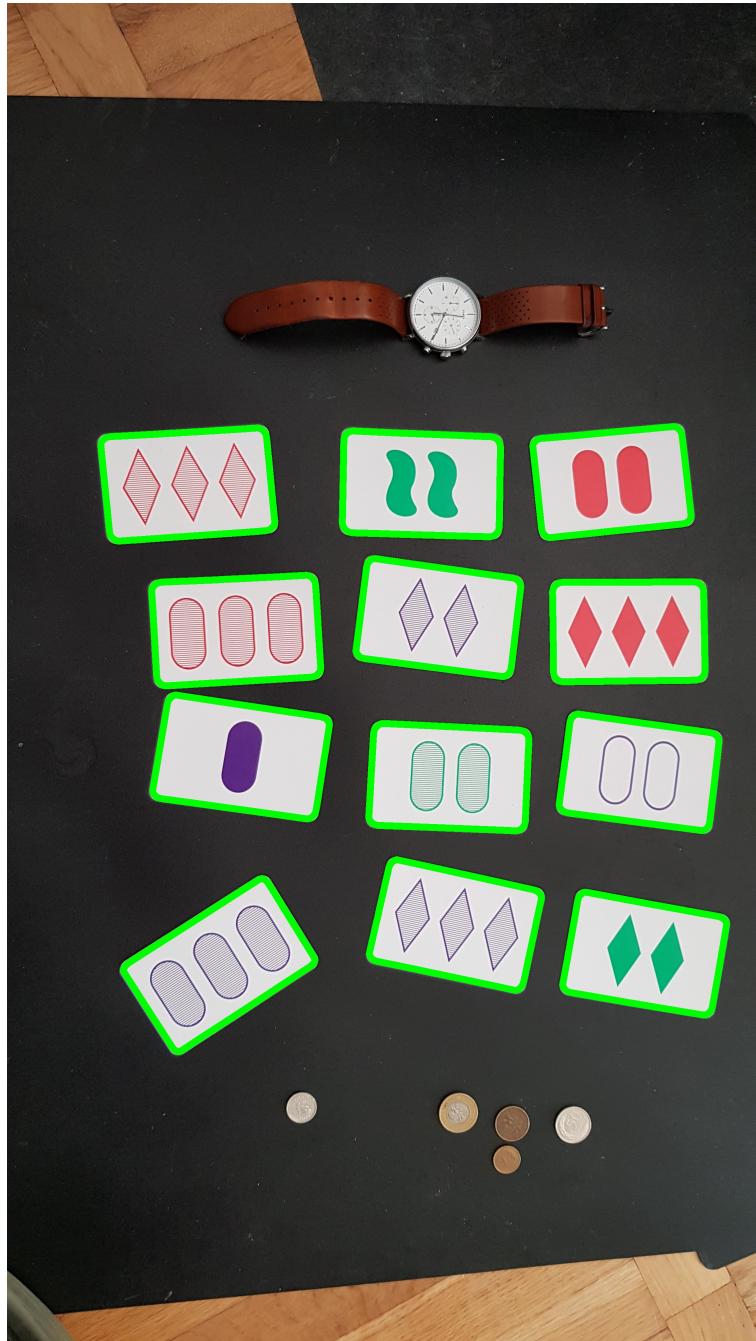
Rysunek 7: Operacja otwarcia.

8. Dopiero po takim przygotowaniu zdjęcia można było wykryć kontury za pomocą gotowej funkcji w bibliotece OpenCv - .findContours



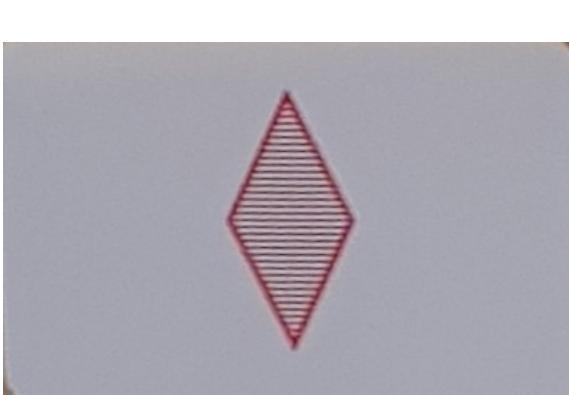
Rysunek 8: Kontury.

9. Pierwszą operacją na konturach było pozbycie się wartości odstających, czyli konturów o zbyt małym oraz zbyt dużym polu powierzchni oraz pozbyciu się konturów o liczbie kątów różnej niż 4.

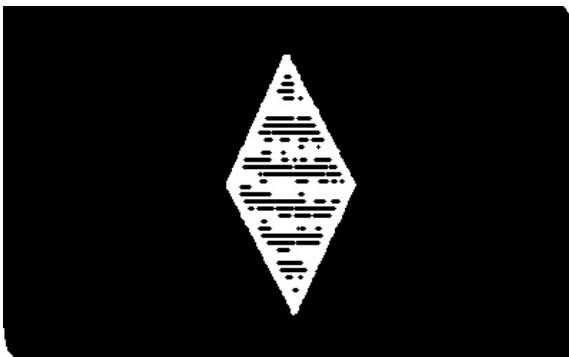


Rysunek 9: Wyselekcjonowane kontury.

10. Następnie z wyselekcjonowanych konturów został wybrany najbardziej typowy kontur za pomocą gotowej funkcji `numpy.median()`. Wybrany kontur posłużył jako wzorzec dla pozostałych konturów i za pomocą funkcji `.matchShape()`, która zwraca wartość równą 0 dla identycznych elementów, z początkowo wybranego zestawu konturów (bez wstępnej selekcji) wyłonione zostały kontury najbardziej podobne do wzorca. To pozwoliło na wybranie tych kart, które według wcześniejszych obliczeń nie były kartami, ponieważ ich kontur nie wyglądał jak prostokąt (np. przez niekorzystne oświetlenie).
11. Kolejnym etapem było wycięcie poszczególnych kart ze zdjęcia oraz przekazanie ich do funkcji weryfikującej kształt, wypełnienie, kolor i liczbę symboli.

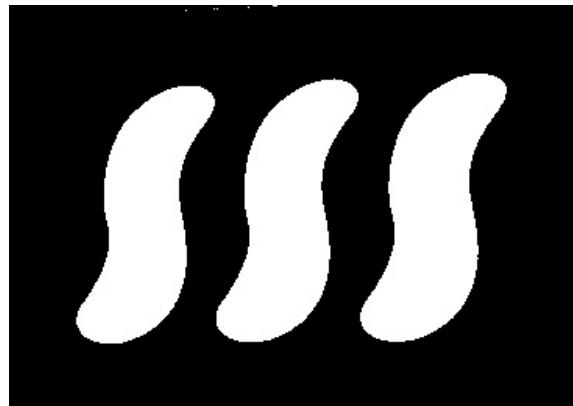
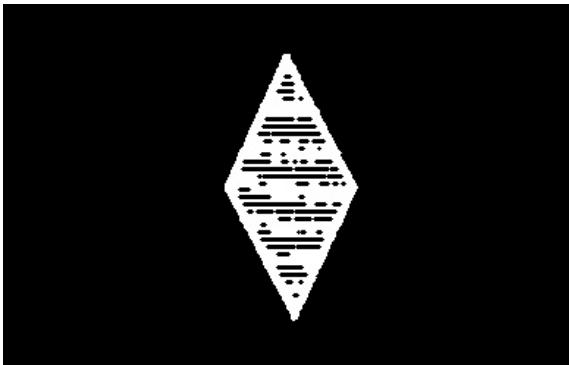


12. Obraz zostaje poddany przetworzeniu za pomocą kontrastu, gammy, progowania, dy-latacji oraz erozji, aby otrzymać zdjęcie gotowe do dalszego przetwarzania.

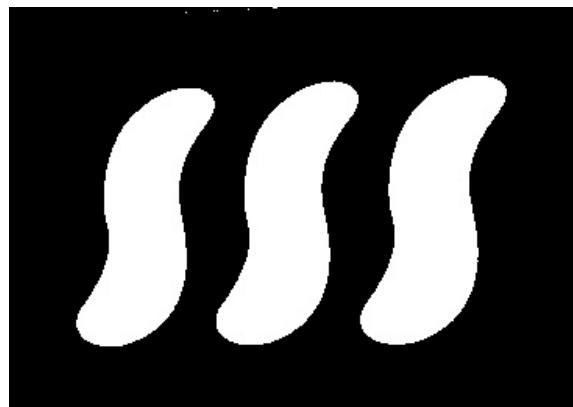
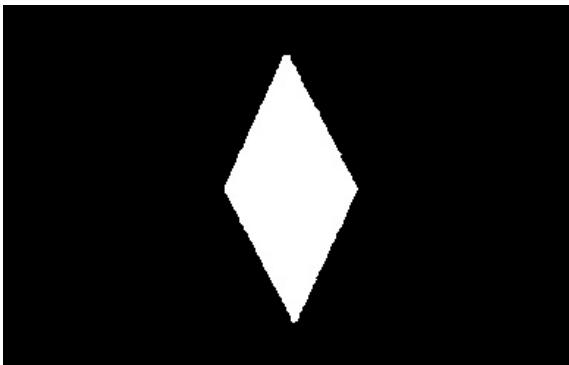


13. W przypadku gdy zdjęcie wyciętej karty zawiera fragmenty podłoża po swojej prawej i lewej stronie, spowodowane tym, że karta fotografowana pod kątem ma kształt równoległoboku, konieczne jest zastosowanie funkcji, która wygasza widoczne tło.

Funkcja ta iteruje po wierszach zdjęcia i wygasza wszystkie białe pixele od jego początku, dopóki nie napotka czarnego pixela. Po wykonaniu takiego wygaszenia lewej strony obrazka, funkcja wykonuje taką samą operację, czytając wiersze od końca, powodując wygaszenie pixeli po jego prawej stronie.



14. Aby wyznaczyć liczbę kształtów zawartych na przetwarzanej karcie program przetwarza co 10 wiersz obrazu i zlicza liczbę wystąpień ciągów zapalonych pixeli dla tego wiersza. Zliczana jest liczba wystąpień jednego, dwóch oraz trzech takich ciągów na wiersz. Liczba kształtów na danej karcie wybierana jest poprzez porównanie zliczonych wystąpień i wybranie najczęściej pojawiającej się liczby ciągów.
15. Program wykorzystuje funkcję `floodFill()` wbudowaną w bibliotekę cv2, która wypełnia kształty na karcie w celu ujednolicenia dalszego przetwarzania.

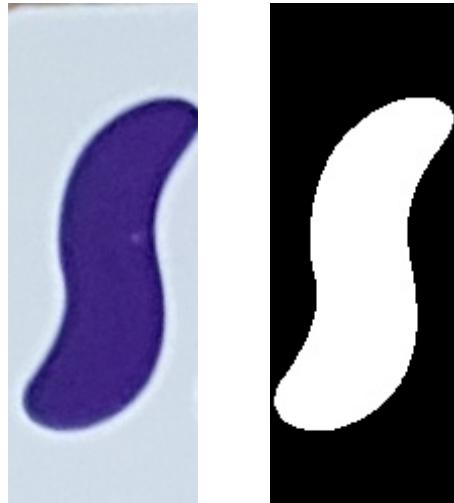
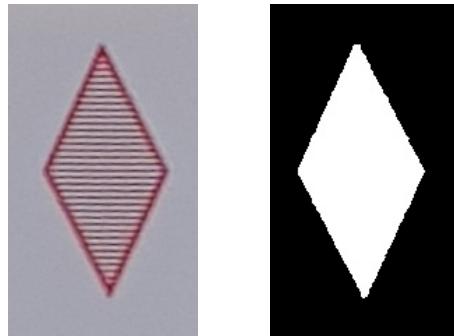


16. Wyznaczanie cechy wypełnienia karty odbywa się poprzez porównanie ilości pixeli przed i po wywołaniu funkcji `floodFill()` na obrazie.

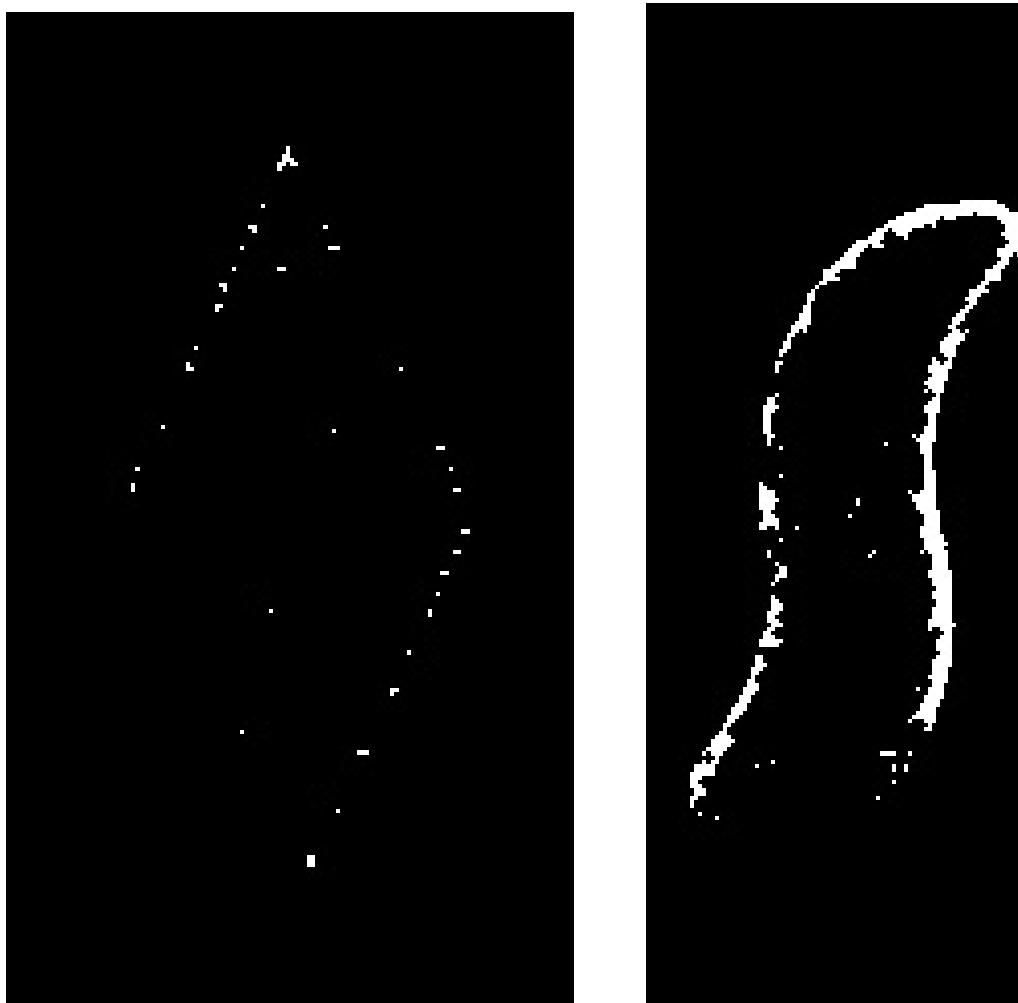
```
pixelePrzed / pixelePo = procentWypelnienia
procentWypelnienia >= 0.95: "full"
0.95 > procentWypelnienia >= 0.3: "striped"
0.3 > procentWypelnienia: "empty"
```

17. Aby wyznaczyć pozostałe cechy karty, konieczne jest dokonanie wycięcia pojedynczego kształtu z obrazu – zarówno obrazu oryginalnego, oraz tego, który został poddany progowaniu.

W celu wycięcia pojedynczego kształtu przetwarzany jest środkowy wiersz obrazu w celu odnalezienia pierwszego ciągu białych pixeli, aby wyznaczyć szerokość kształtu. Następnie kształt wycinany jest na odnalezionej szerokości z pewnym dodatkowym marginesem dla bezpieczeństwa.

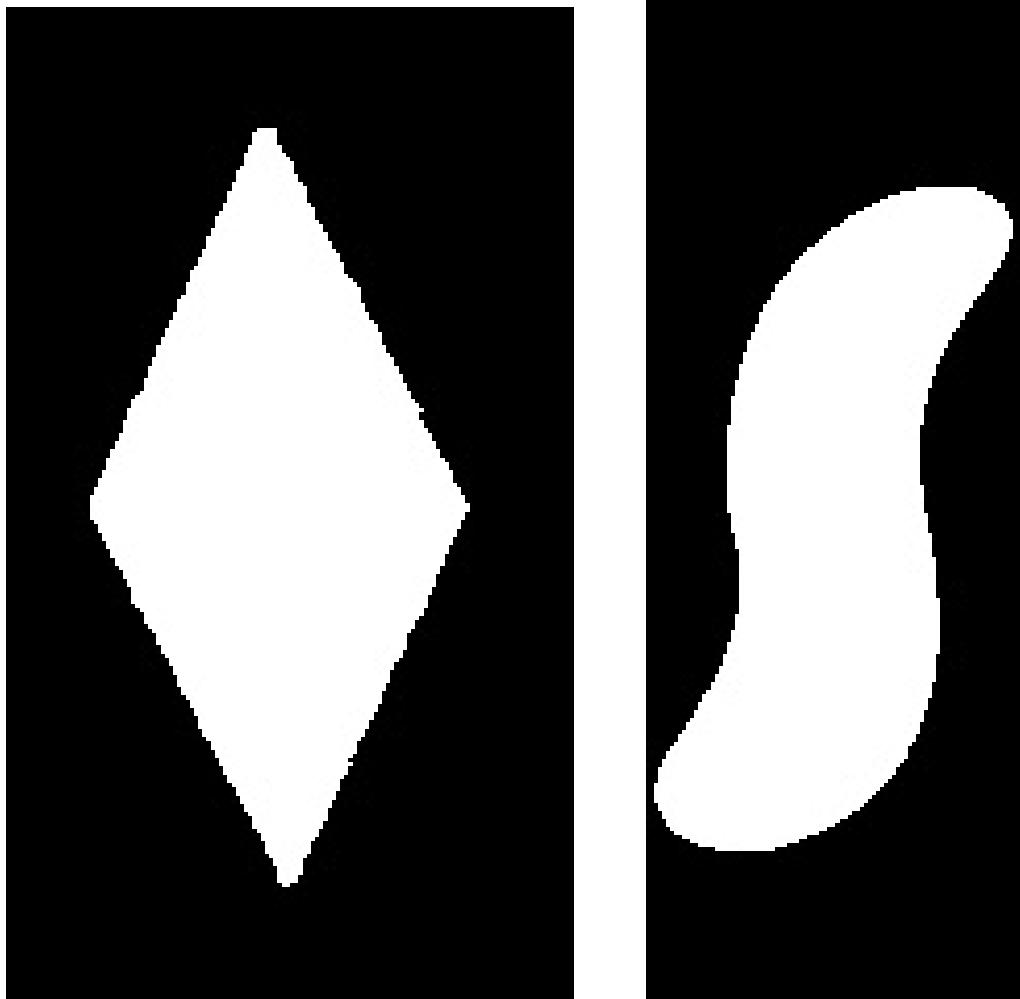


18. Wyznaczanie cechy koloru karty odbywa się poprzez nałożenie progowania na fragment oryginalnego obrazu z pojedynczym kształtem, zmieniając wysokość progu tak, aby otrzymać 30 najciemniejszych pixeli. Następnie program iteruje po wszystkich pikselach obrazu. Jeżeli odnajdzie zapalony pixel, to pobiera wartości kolorów RGB pixela oryginalnego zdjęcia, znajdującego się na tej samej pozycji. Następnie wyznaczana jest mediana wartości każdego z kolorów.



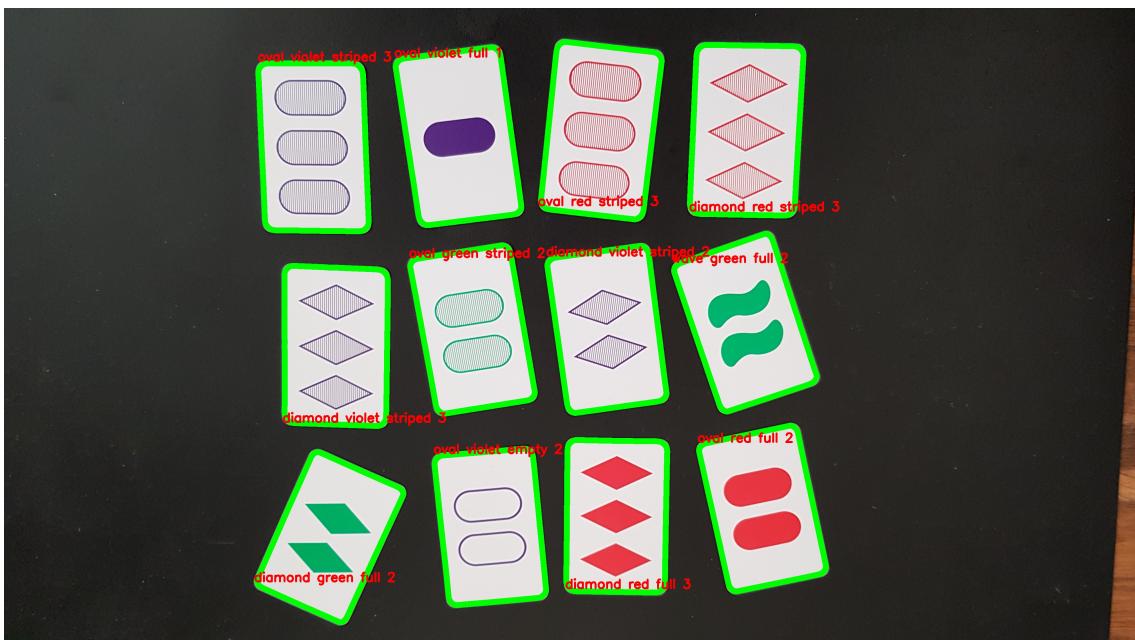
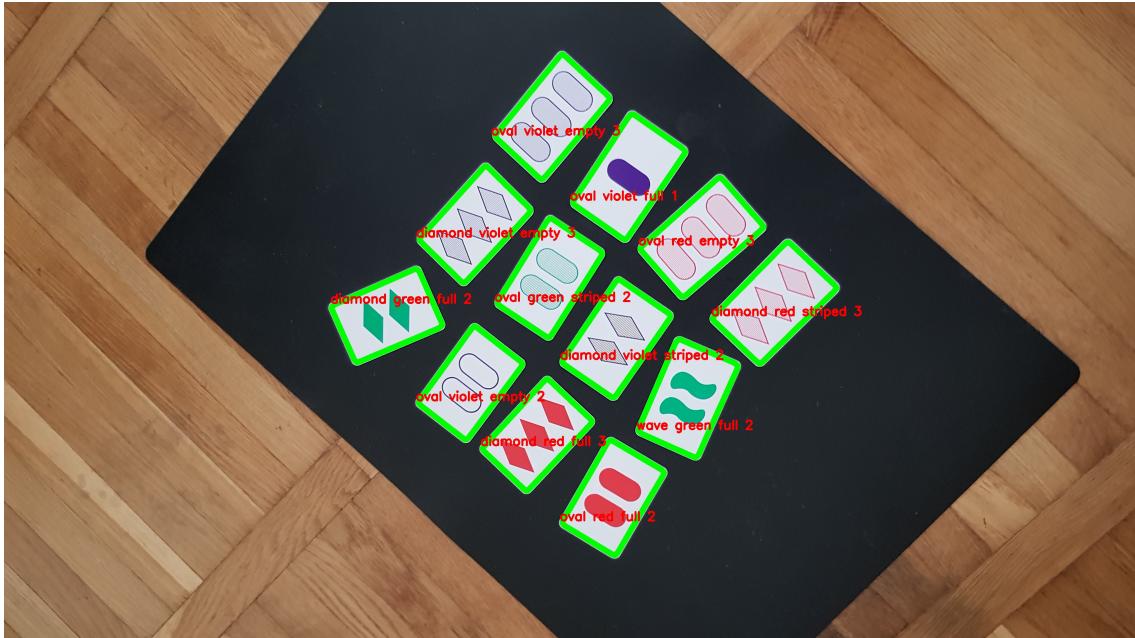
-
19. Wyznaczanie cechy kształtu odbywa się poprzez wywołanie funkcji approxPolyDP() wbudowanej w bibliotekę cv2. Z jej pomocą wyznaczany jest wielokąt rysowany na krawędziach kształtu przetwarzanego na obrazie.

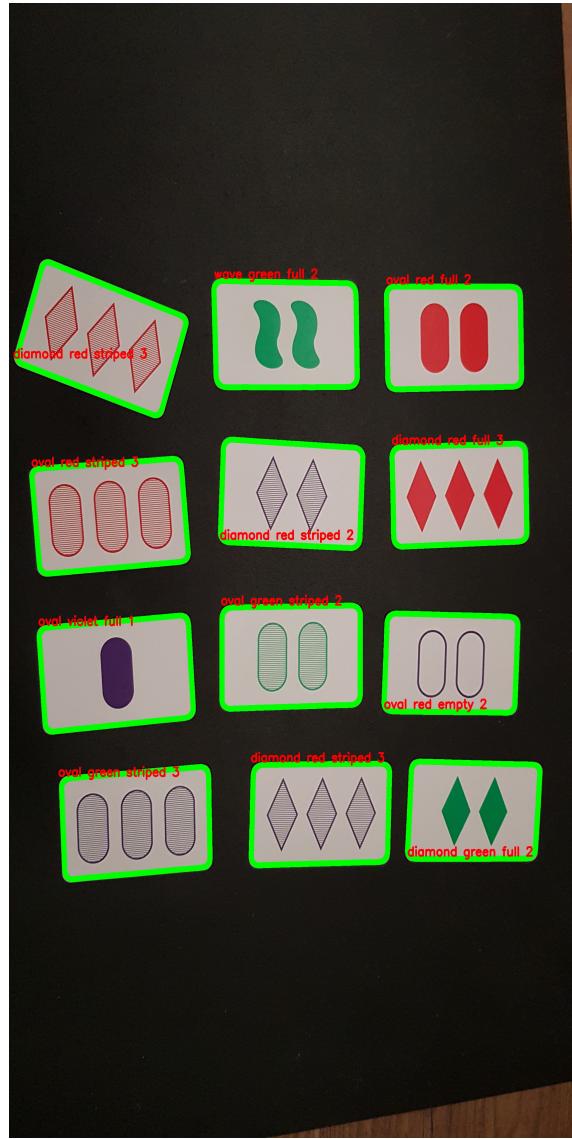
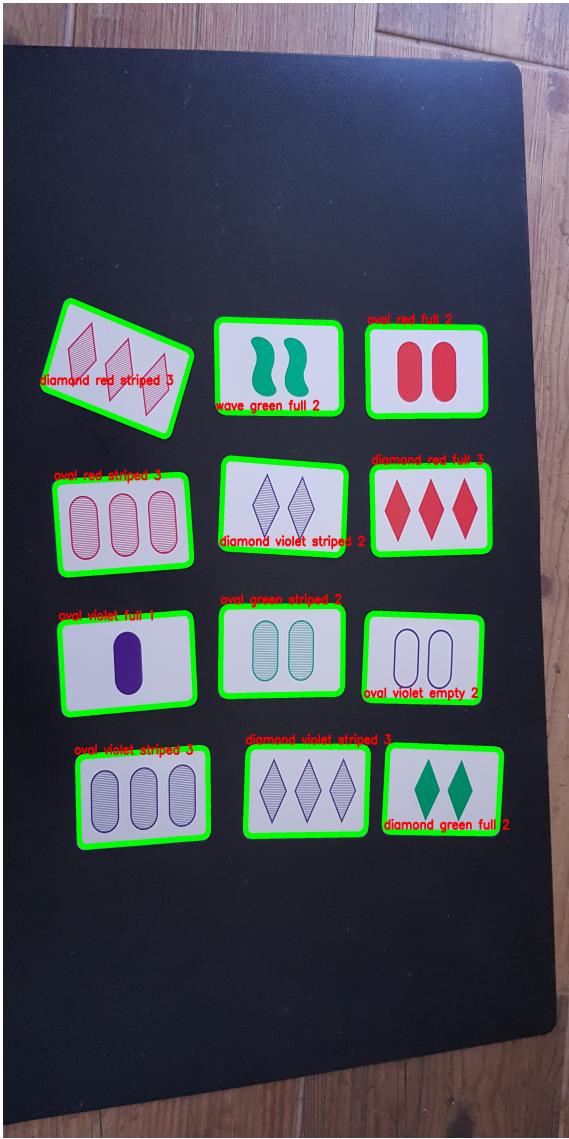
k – liczba krawędzi wielokąta wykrytego przez cv2
k >= 18: "wave"
17 > k >= 11: "oval"
11 > k: "diamond"

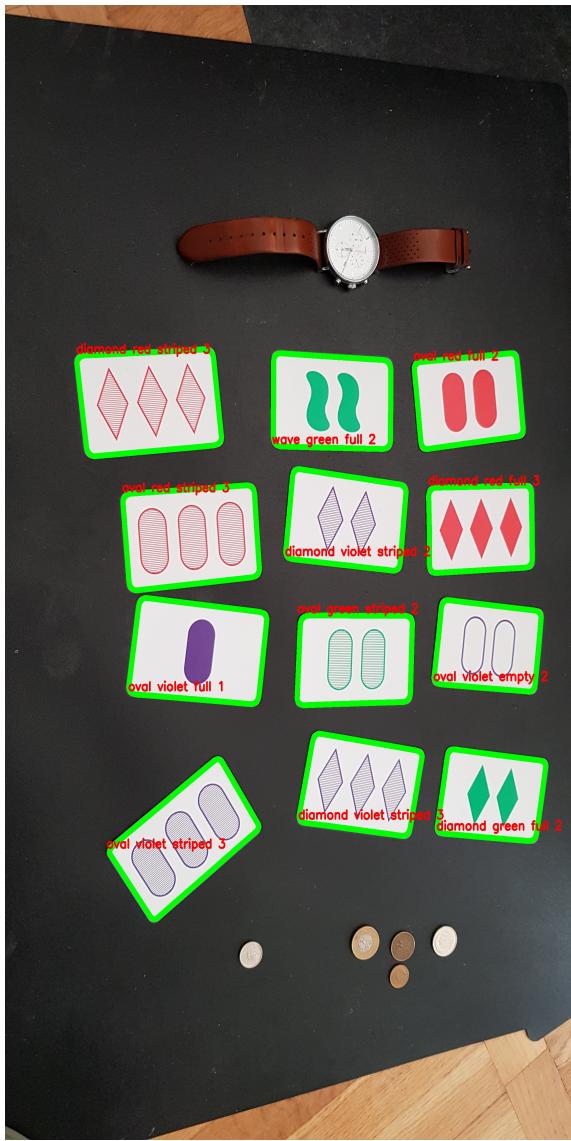


Oto efekty działania programu dla każdego z zestawu:

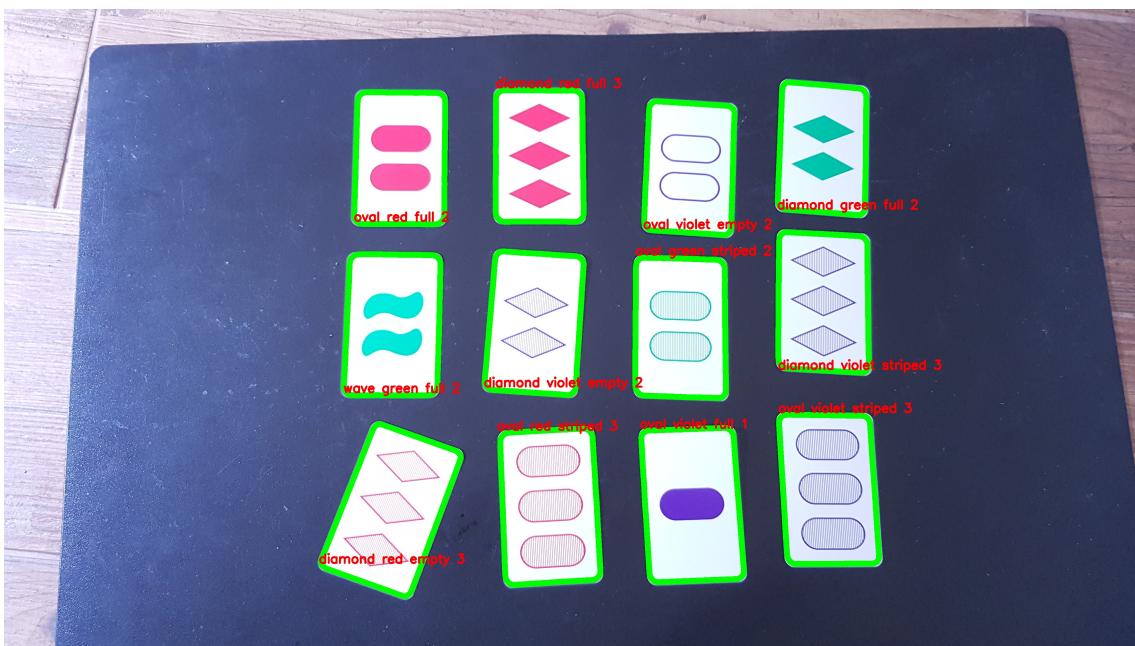
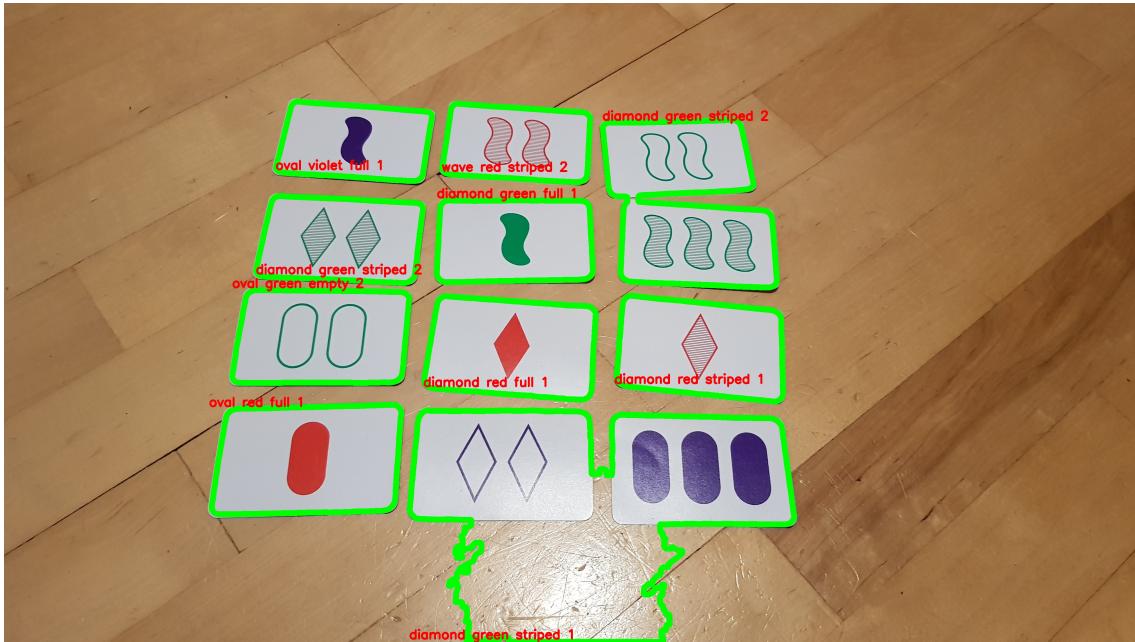
1. Zestaw łatwy:

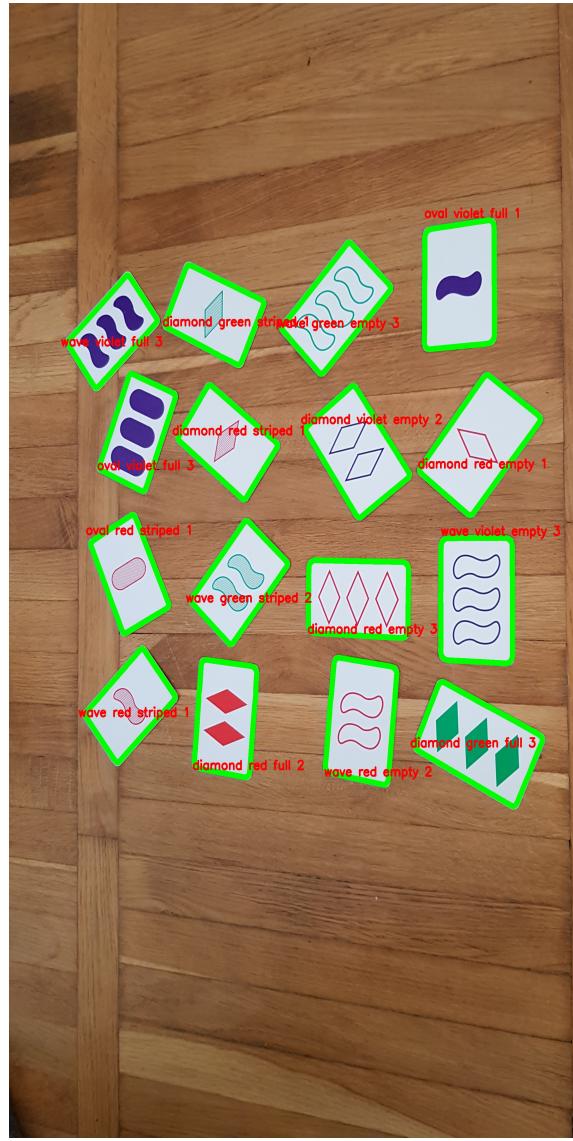
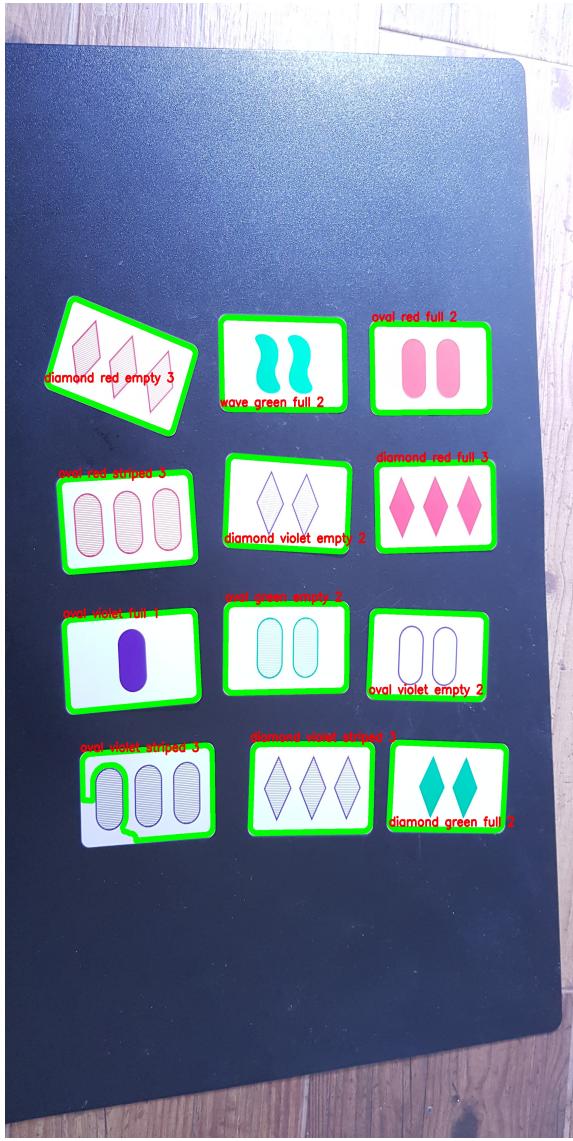


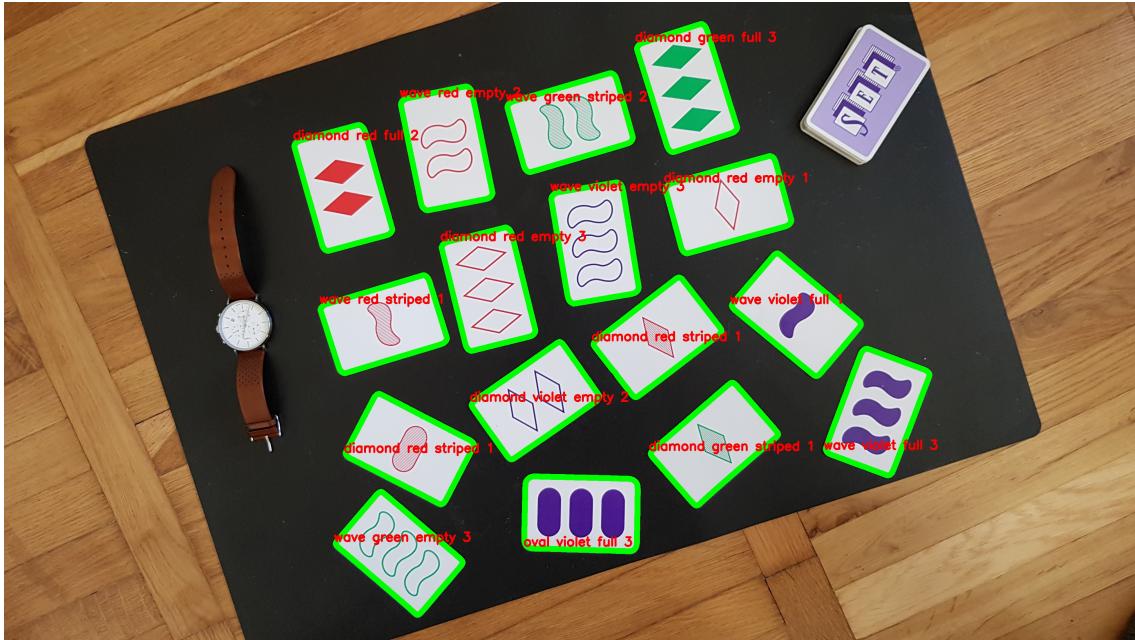




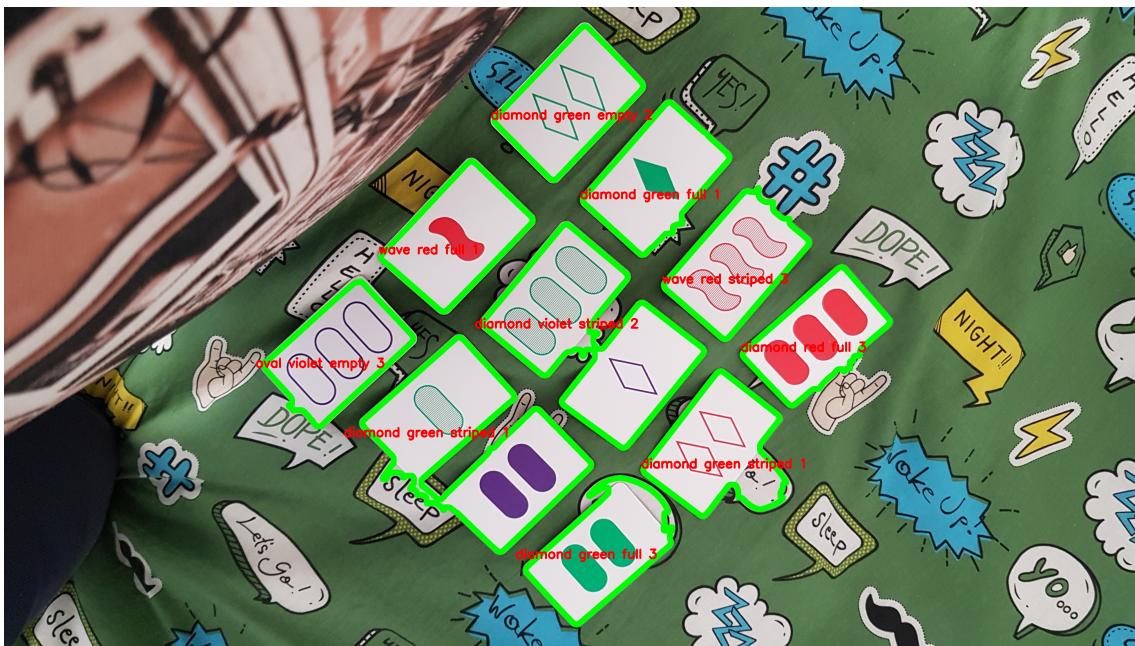
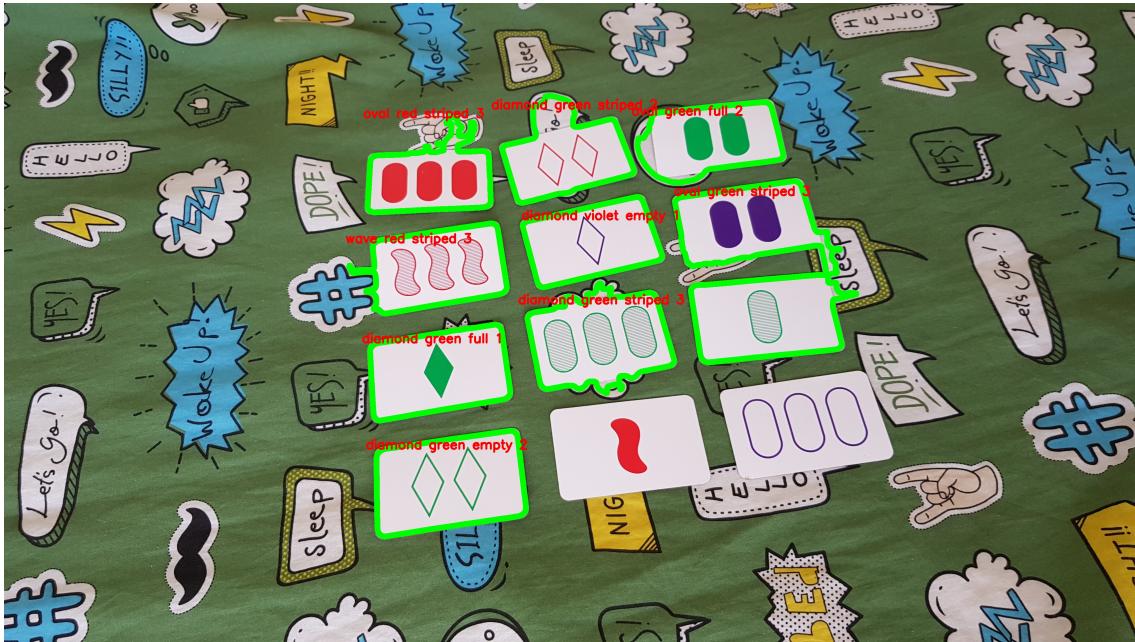
2. Zestaw średni:

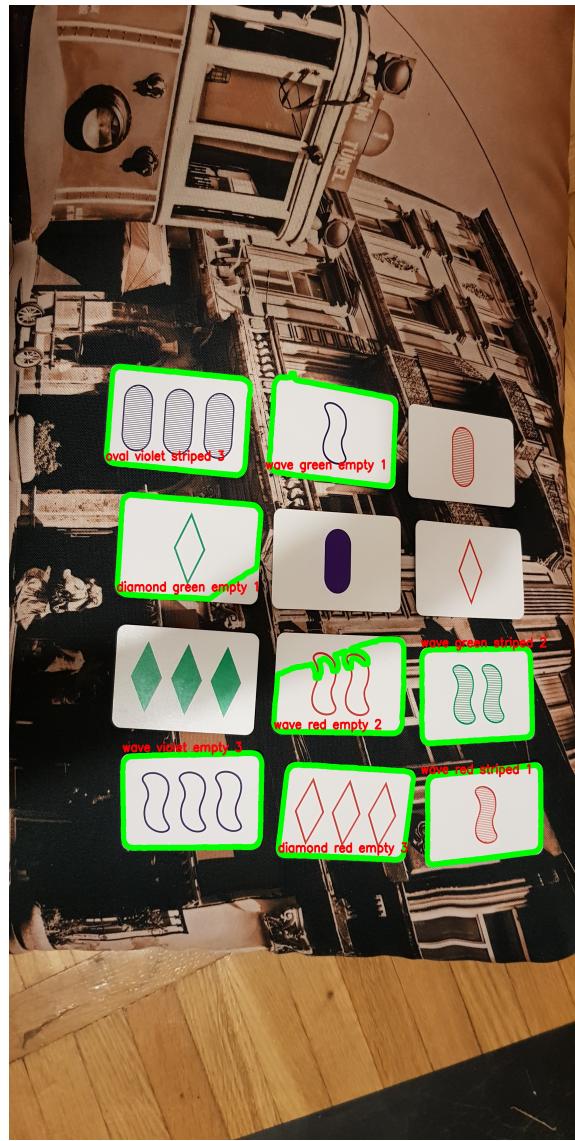
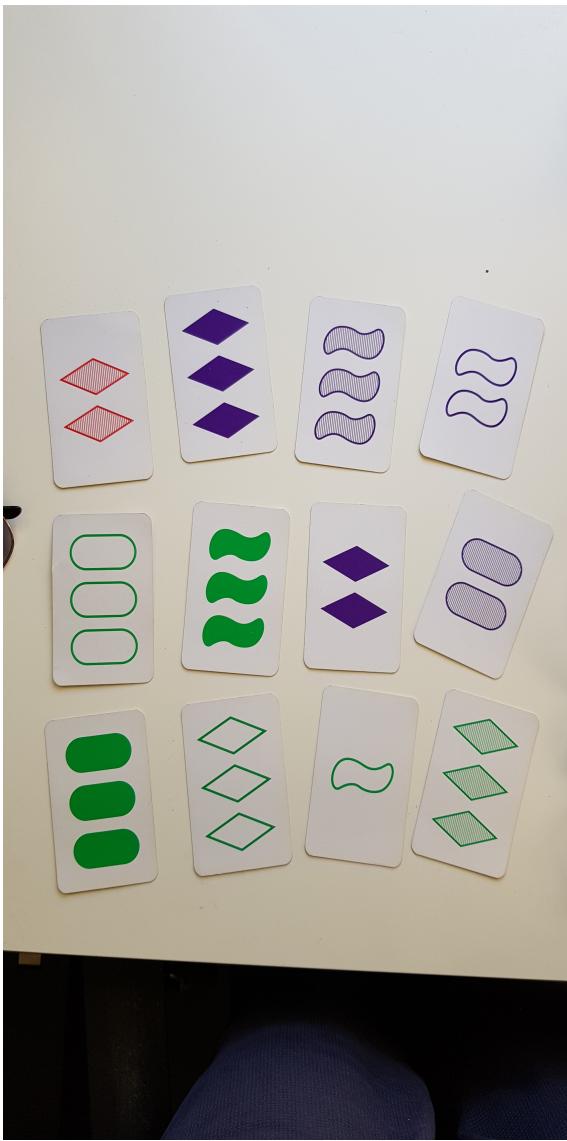


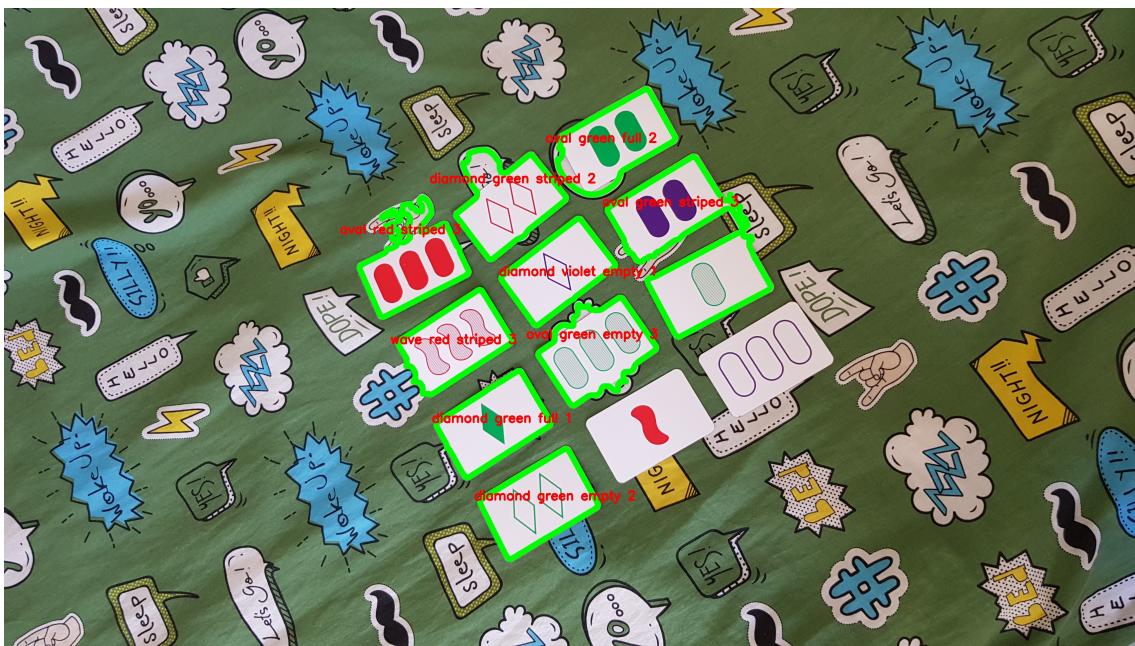
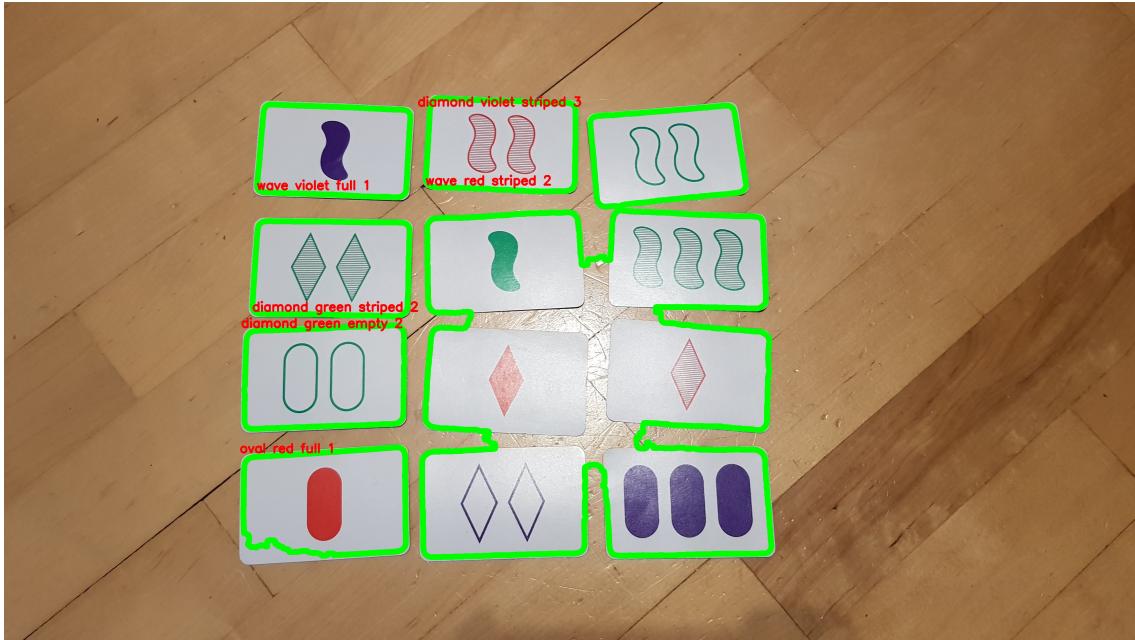




3. Zestaw trudny:







W zestawie łatwym każda karta zostaje prawidłowo rozpoznana, spowodowane jest to tym, że leżą one na ciemnym tle, którego łatwo się pozbyć na etapie przetwarzania za pomocą chociażby wysokiego kontrastu. Dystraktory nie mają wpływu na działanie programu, ponieważ nie są one czworokątami (np. zegarek) oraz mają zbyt małe pole powierzchni.

Zestaw średni również nie spowodował większych problemów z rozpoznaniem zdjęć. Mimo tego, że zdjęcia zostały wykonane pod większym kątem program potrafił rozpoznać typ karty. Największym problemem, co widać na pierwszym zdjęciu, jest wysokie oświetlenie skupione wokół jednego punktu, co powoduje, że znajdują się tam bardzo jasne piksele i program nie potrafi odseparować podłożu od równie białej karty, przez co kontur kart łączy się z kawałkiem tła. Taki przypadek nie został dobrze zidentyfikowany.

W zestawie trudnym na żadnym zdjęciu nie zostały rozpoznane karty w 100%. Na zielonym tle znajdowały się białe wzory, których kolor był bardzo niekorzystny, ponieważ podczas przetwarzania zdjęcia program nie był w stanie rozróżnić czy jest to karta czy wzór. Po przerzutzeniu kontur karty był połączony z konturem wzoru. Na zdjęciach z białym tłem kontury kart wcale nie zostały wykryte, ponieważ ciężko było zwiększyć kontrast między kartami, a podłożem. Na pozostałych zdjęciach problemem również się okazało wysokie natężenie światła skupione w jednym punkcie oraz cień rzucony na karty.

Program doskonale sobie radzi ze zdjęciami znajdującymi się na ciemnym tle oraz warto zauważać, że żadne dystraktory np. w postaci zegarka czy odwróconych kart (zdjęcie nr. 5 zestawu średniego) nie mają wpływu na działanie programu.