

POLITECHNIKA POZNAŃSKA

WYDZIAŁ INFORMATYKI i TELEKOMUNIKACJI

Informatyka

Juliusz Horowski

nr albumu: 136247
adres e-mail: juliusz.horowski@student.put.poznan.pl

Michał Kalinowski

nr albumu: 136252
adres e-mail: michal.kalinowski@student.put.poznan.pl

Tomasz Kiljańczyk

nr albumu: 136257
adres e-mail: tomasz.kiljanczyk@student.put.poznan.pl

Szymon Wasilewski

nr albumu: 136323
adres e-mail: szymon.p.wasilewski@student.put.poznan.pl

Dokumentacja projektu

PokeVisor

Spis treści

1. Opis i uzasadnienie wyboru tematu	3
2. Podział prac	4
3. Wybrane technologie	4
3.1 Technologie	4
4. Architektura rozwiązania	5
5. Interesujące problemy i rozwiązania	6
5.1 Detekcja żetonów	6
5.1.1 Detekcja kół	6
5.1.2 Rozpoznanie wartości	7
5.2 Detekcja kart	9
5.2.1 Wykrywanie kandydatów	9
5.2.2 Przetwarzanie wycinka	12
5.2.3 Rozpoznanie karty	14
5.3 Podział stołu i wyświetlanie informacji	16
5.3.1 Podział na graczy	17
5.3.2 Wydzielenie części wspólnej	18
5.3.3 Nakładanie informacji na obraz	19
6. Instrukcja użytkowania aplikacji	21
6.1 Generator klasyfikatora	21
6.2 Analizator nagrąń	23
6.3 Analizator obrazów	25

1. Opis i uzasadnienie wyboru tematu

PokeVisor to aplikacja, która powstała w celu wspomagania rozgrywek pokerowych. Aplikacja, z wykorzystaniem nagrań (z możliwością łatwego rozszerzenia do korzystania z kamery), analizuje ułożenie elementów (kart i żetonów) na stole do gry i w czasie rzeczywistym przekształca zgromadzone dane na format tabelaryczny. Umożliwia to szybkie i klarowne przedstawienie danych o rozgrywce użytkownikowi.

Dla jednego stołu aplikacja jest w stanie obsłużyć do 8 graczy. W obrębie tej gałęzi ma za zadanie: rejestrować i nadzorować rozgrywkę, jak również prezentować informacje użytkownikom.

W przyszłości PokeVisor za pomocą odpowiednio skalibrowanej i pozycjonowanej kamery będzie mógł rejestrować obraz znad stołu. Umożliwi to zbieranie w czasie rzeczywistym informacji o stanie rozgrywek.

Użytkownicy mają dostęp do podglądu informacji o rozgrywce. Aplikacja przedstawia najbardziej optymalny i uniwersalny zakres informacji, tj. wyłożone na stół karty graczy, karty wspólne, kombinacje kart graczy (hand) oraz sumę wartości wyłożonych przez nich żetonów.

2. Podział prac

Tomasz Kiljańczyk Szymon Wasilewski	Back-end: Analiza obrazu z różnych źródeł, rozpoznawanie obiektów, przekształcanie danych z obrazu na informacje o rozgrywce, synchronizacja danych w czasie rzeczywistym (w przypadku nagrani)
Juliusz Horowski Michał Kalinowski	Front-end: Stworzenie graficznego interfejsu, reprezentacja danych w postaci zrozumiałej i wygodnej dla użytkownika, umożliwienie synchronizacji danych w czasie rzeczywistym

3. Wybrane technologie

3.1 Technologie

1. Python 3 - mało wydajny i prosty w obsłudze język programowania. Język ten posiada wiele bibliotek, dlatego został wybrany w implementacji PokeVisor.
2. TkInter - biblioteka wykorzystywana do tworzenia "brzydkich" interfejsów. Użyta została do zaimplementowania aplikacji okienkowych.
3. OpenCV - biblioteka związana z przetwarzaniem obrazu. Biblioteka ta umożliwia w sposób intuicyjny tworzenie, wczytywanie, manipulowanie, wyświetlanie i zapisywanie obrazów.
4. NumPy - biblioteka związana z obliczeniami matematycznymi. OpenCV wykorzystuje tą bibliotekę.
5. scikit-learn - biblioteka związana z uczeniem maszynowym. Za jej pomocą stworzono klasyfikator MLP (multilayer perceptron).

4. Architektura rozwiązania

PokeVisor składa się z szeregu skryptów implementujących logikę aplikacji oraz interfejs.

Z części opisującej logikę aplikacji można dalej wydzielić następujące grupy:

- detekcja kart,
- detekcja żetonów,
- rozpoznawanie ręki,
- nadzorowanie rozgrywki.

Na skrypty opisujące interfejs użytkownika składają się:

- generator klasyfikatora,
- analizator zdjęć,
- analizator nagrani,
- okno statusu rozgrywki.

5. Interesujące problemy i rozwiązania

5.1 Detekcja żetonów

Aby rozpoznać żetony należało pokonać dwa zasadnicze problemy:

- Jak znaleźć obiekty przypominające żetony (koła),
- Jaką wartość reprezentuje znaleziony żeton.

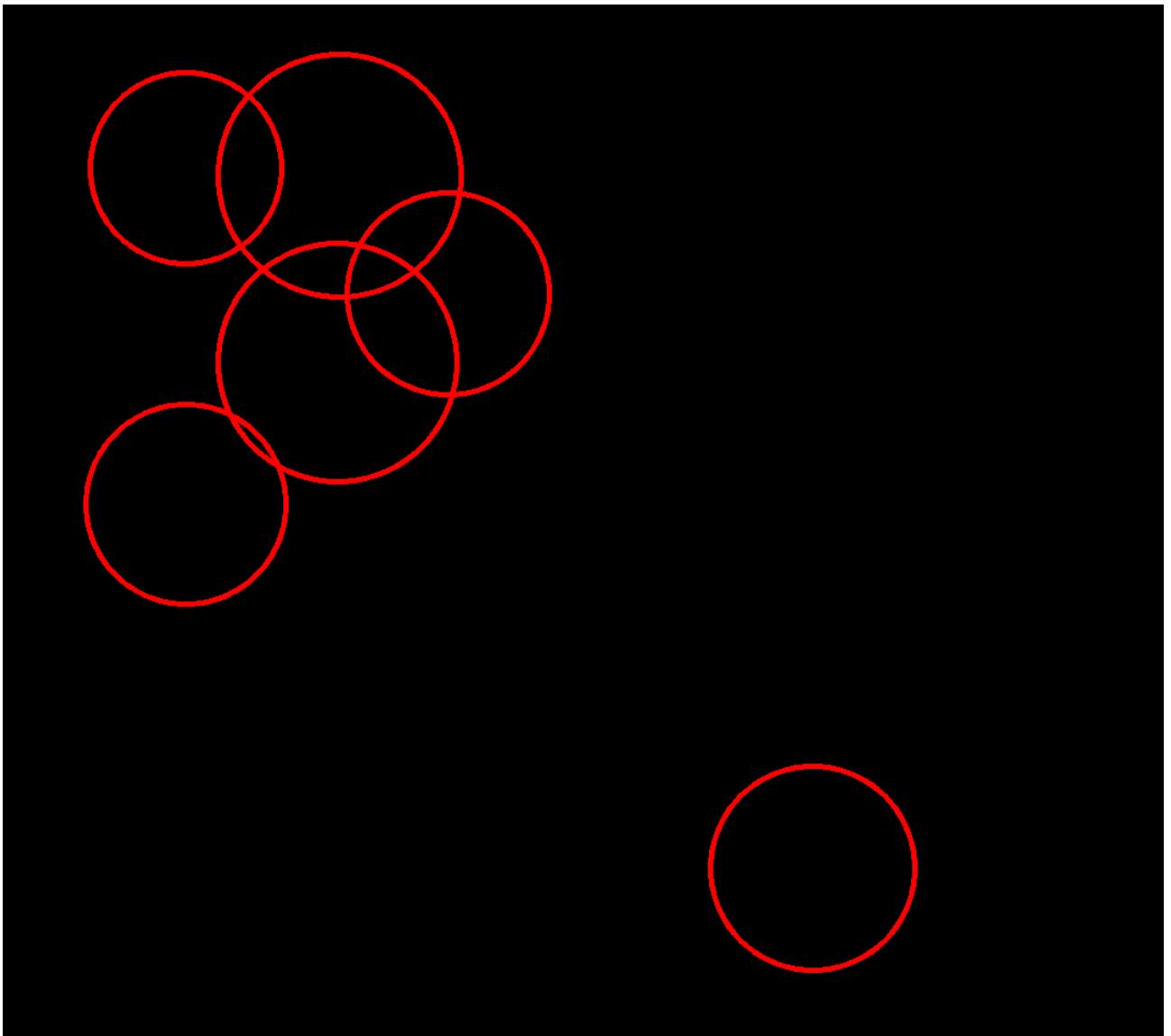
Należy zaznaczyć, że zastosowane rozwiązanie jest adaptacją detektora monet, który znajduje się na repozytorium [GitHub](#).



Grafika 01 - grafika użyta do detekcji żetonów

5.1.1 Detekcja kół

Wykrywanie kół wymaga konwersję obrazu na skalę szarości, rozmycia i zastosowania gotowej funkcji OpenCV HoughCircles, która odpowiada za wykrywanie okręgów.



Grafika 02 - wykryte okręgi

5.1.2 Rozpoznanie wartości

W trakcie rozwiązywania tego problemu wykorzystywany jest obraz w HSV. Przed próbą rozpoznania wartości żetonu, wycinany jest z obrazu kwadratowy fragment o szerokości i wysokości równej średnicy okręgu.



Grafika 03 - wycinki na bazie wykrytych okręgów

Każdy przekazywany jest do wyszkolonego wcześniej klasyfikatora, który zwraca jakiego koloru jest żeton lub informacje o nieroznalezieniu żadnego koloru.



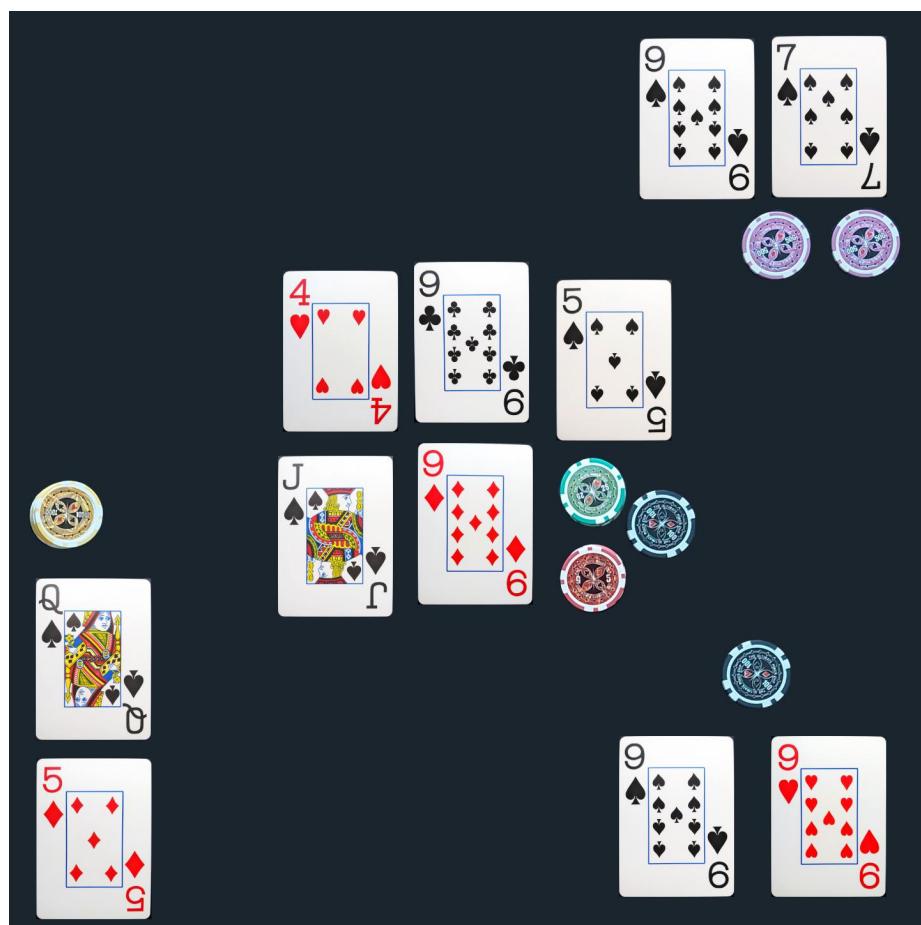
Grafika 04 - wykryte żetony.

5.2 Detekcja kart

Rozpoznawanie kart było jedną z kluczowych funkcjonalności projektu. Jako że wymagane jest przetwarzanie obrazu, to pojawiły się różne problemy:

- jak wykryć obiekt, który może być kartą,
- jak przetworzyć wycinek obrazu, aby można spróbować rozpoznać typ karty,
- rozpoznanie typu karty.

Należy ponownie zaznaczyć, że zastosowane rozwiązanie jest wzbogaconą wersją już istniejącego, które można znaleźć na repozytorium [GitHub](#).

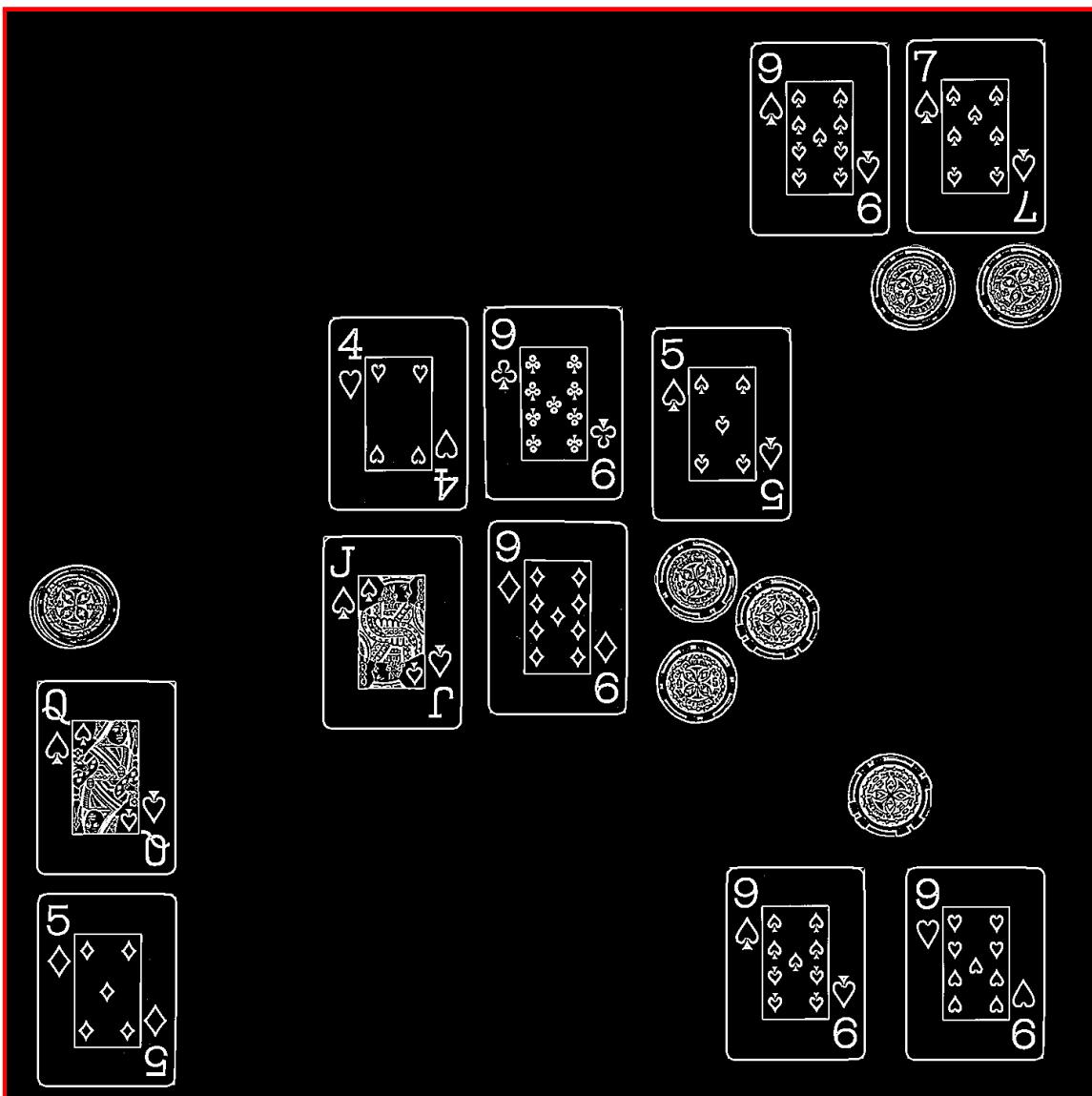


Grafika 05 - obraz użyty do detekcji kart

5.2.1 Wykrywanie kandydatów

Proces znajdowania kandydatów rozpoczyna się od konwersji obrazu na skalę szarości, następnie jego rozmycie oraz progowanie. W PokeVisor zastosowano progowanie

adaptacyjne, aby zmniejszyćczęstość występowania problemów związanich z złym oświetleniem.

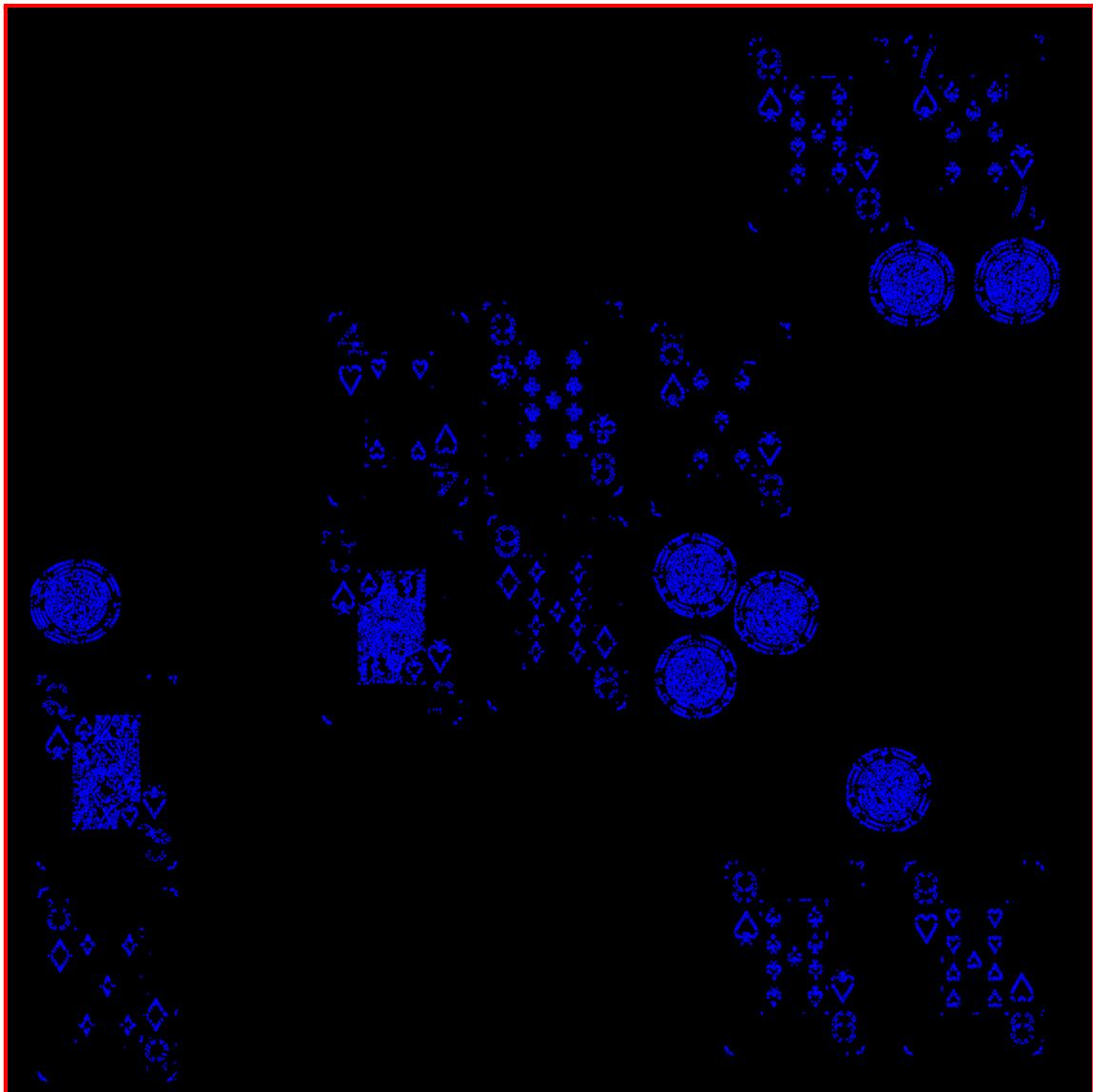


Grafika 06 - wyprocgowany obraz

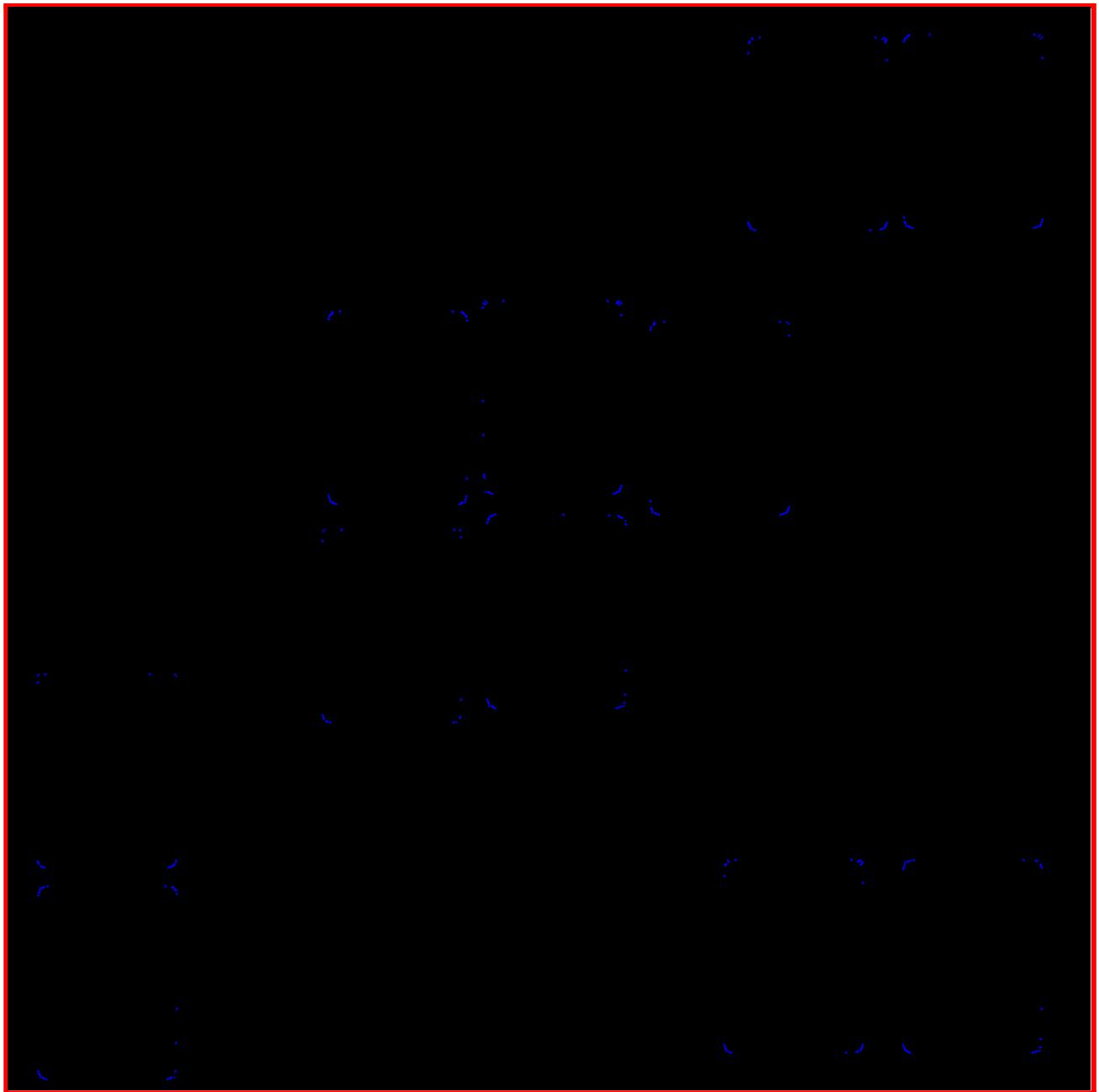
Następnie wyszukiwane są kontury na wyprocgowanym obrazie. Każdy kontur musi spełniać określone warunki:

- pole konturu mieści się w stałym przedziale,
- kontur przybliżony do wielokąta jest czworokątem.

Jeśli kontur nie spełnia warunku jest on oznaczany do usunięcia. Kontury, które spełniły powyższe warunki są sprawdzane pod kątem zawierania się jednego w drugim. Jeśli kontur zawiera się całkowicie w innym konturze, to jest on tak jak poprzednio oznaczany do usunięcia. Na tym etapie pozostają tylko kontury z największą szansą bycia kartą.



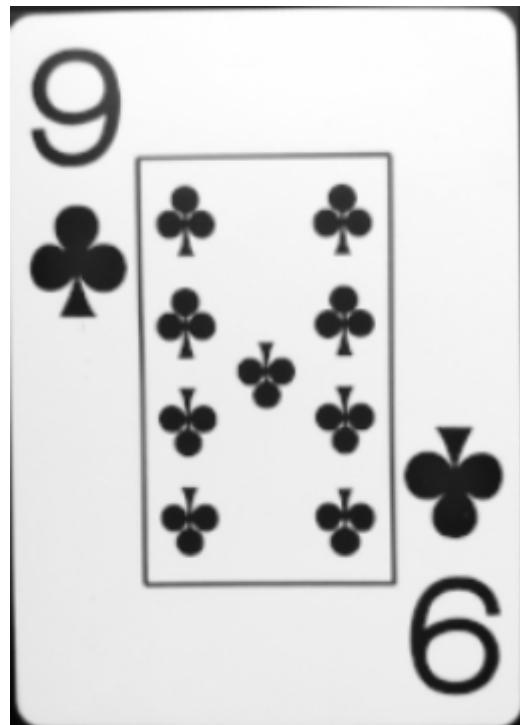
Grafika 07 - kontury przed odrzuceniem złych kandydatów



Grafika 08 - kontury odfiltrowanych kandydatów

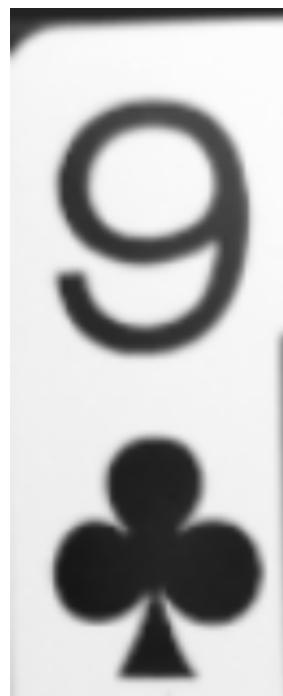
5.2.2 Przetwarzanie wycinka

Otrzymane w poprzednim kroku kontury służą do wycięcia fragmentu obrazu. Taki fragment poddawany jest szeregowi operacji, aby uzyskać pożądane informacje. Wpierw obliczany jest punkt środkowy oraz rozmiary czworoboku znajdującego się na wycinku. Dzięki tym danym możliwe jest wykonanie przekształcenia perspektywistycznego wycinka.

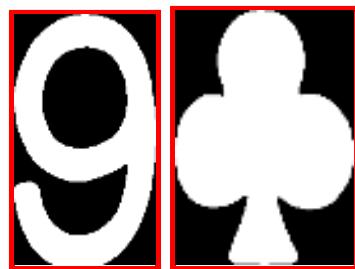


Grafika 09 - wyprostowana karta

Wyprostowany fragment obrazu jest przeskalowany do stałego rozmiaru i wycinany jest z niego lewy górny róg. Ostatecznie z fragmentu wycinane są ranga i kolor karty.



Grafika 10 - wycinek rogu karty



Grafiki 11 - wycięta i przeskalowana ranga i kolor karty

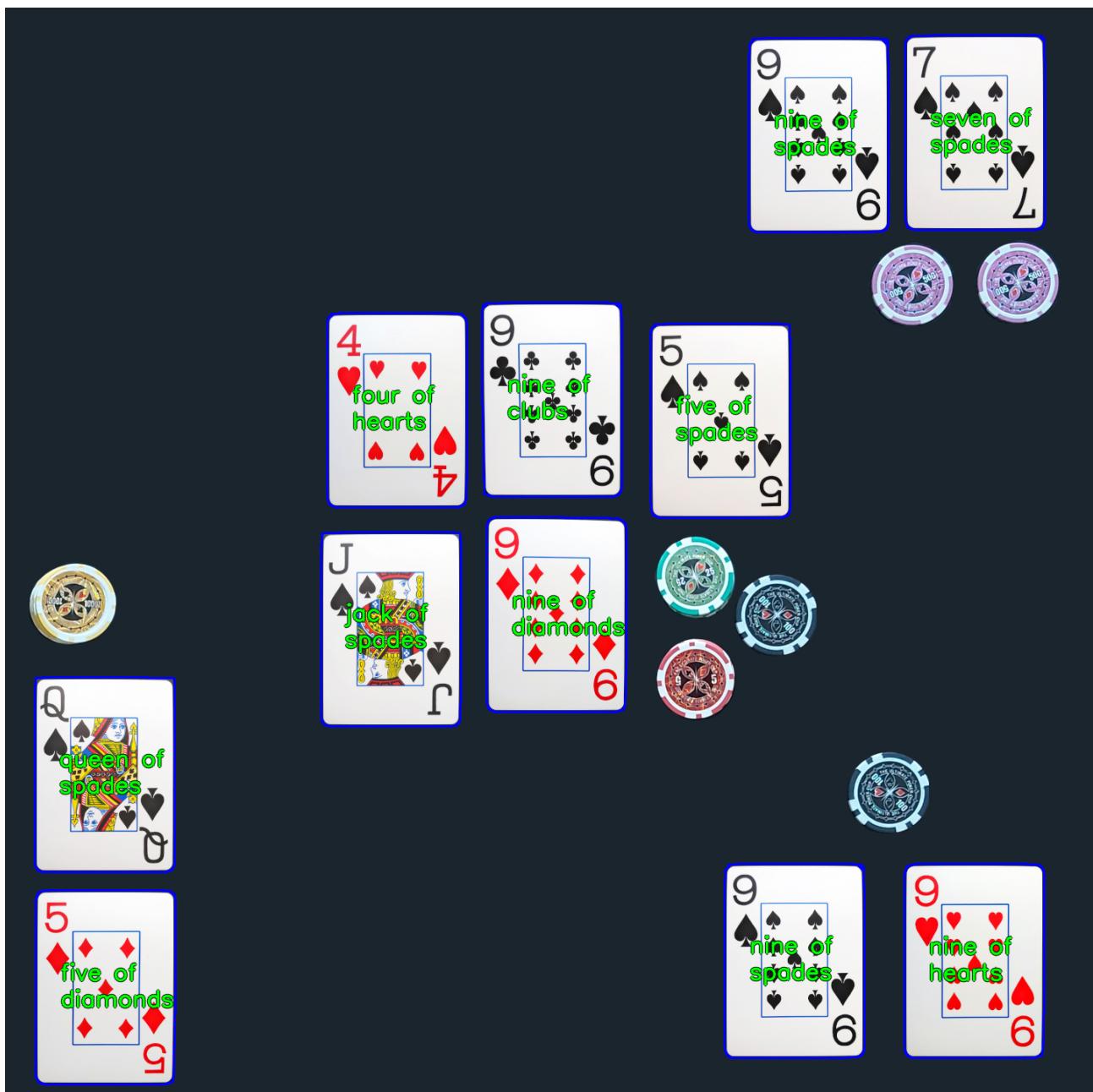
5.2.3 Rozpoznanie karty



Grafika 12 - zbiór symboli używanych do porównywania rangi i koloru karty

Rozpoznanie karty wymaga użycia symboli przedstawionych na grafice x. Wycinek z rangą porównywany jest z symbolami rangi, dla wycinku z kolorem stosowana jest analogiczna strategia. Różnica obliczana jest za pomocą funkcji OpenCV absdiff. Wybierana jest ranga/kolor karty, która ma najmniejszą różnicę z wycinkiem. Jeśli ta

różnica nie jest mniejsza od zadanej stałej, to ranga/kolor karty oznaczana jest jako nieznana. W przeciwnym przypadku karcie przypisane zostaną odpowiednie informacje.

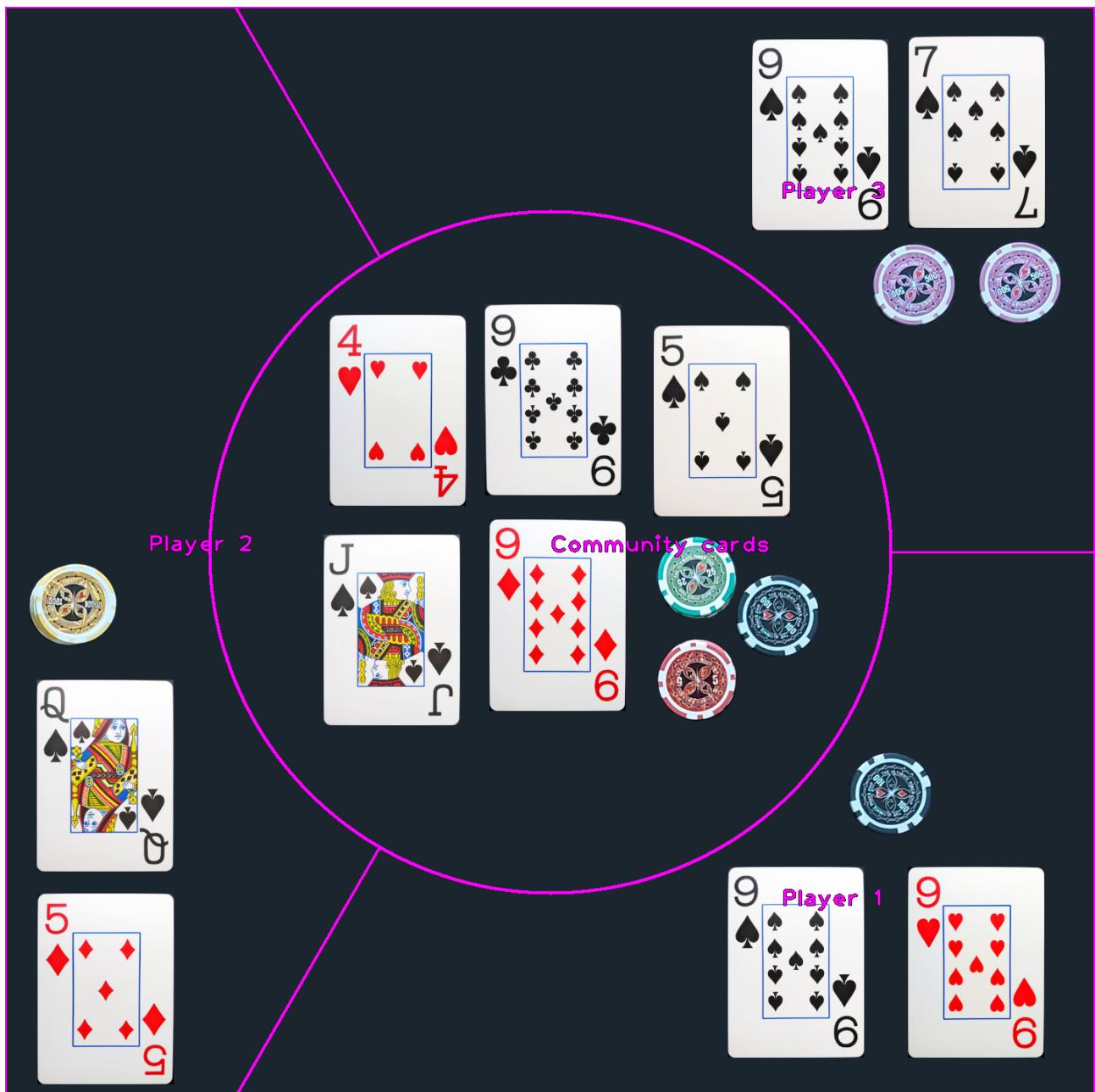


Grafika 13 - wynik detekcji kart

5.3 Podział stołu i wyświetlanie informacji

Podzielenie stołu i wyświetlanie informacji stawało przed projektem trzy wyzwanie:

- jak podzielić stół pomiędzy graczy (od 2 do 8),
- jak wydzielić część dla kart wspólnych,
- jak nałożyć informacje tak, żeby nie przeszkadzały w detekcji obiektów.



Grafika 14 - efekt końcowy podziału stołu na 3 graczy

5.3.1 Podział na graczy

Podział stołu na graczy zaimplementowano poprzez prowadzenie linii pod kątem z centrum obrazu do jego krawędzi. Dzięki wierzchołkom tych linii możliwe jest wycięcie fragmentów odpowiadających poszczególnym graczom. W rozwiążaniu występują dwa specyficzne przypadki z dedykowanym rozwiązańiem:

- dwóch graczy - punkty linii były niewystarczające, dlatego dla każdego z graczy należało wyznaczyć dwa dodatkowe punkty (wierzchołki obrazu), aby możliwe było wycięcie ich części,
- trzech graczy - wycinki gracza 1 i gracza 3 były niekompletne. Należało wyznaczyć dla nich dodatkowy punkt (wierzchołek obrazu).

Po wyznaczeniu punktów tworzona jest maska wycinka gracza z uwzględnieniem części wspólnej.



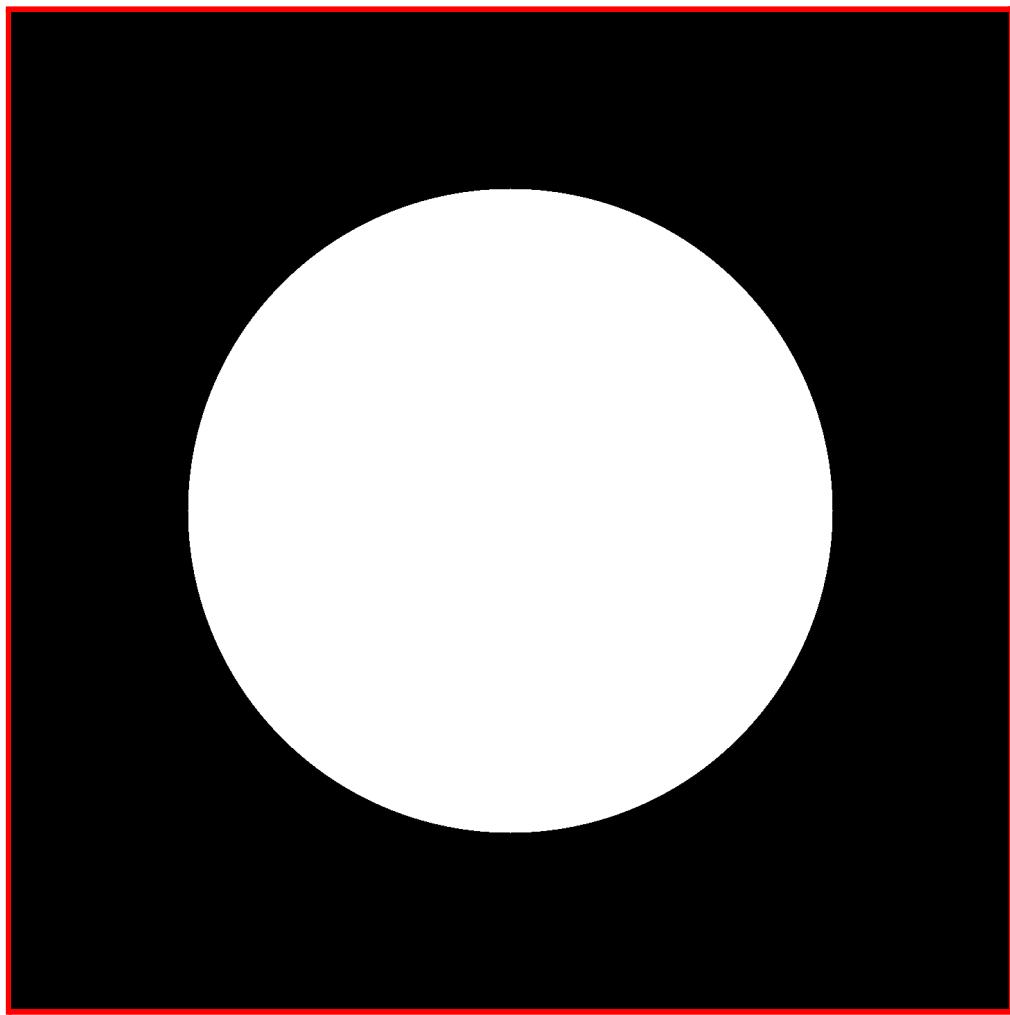
Grafika 15 - maski dla części poszczególnych graczy

5.3.2 Wydzielenie części wspólnej

Najprostszym sposobem na wydzielenie części wspólnej jest utworzenie kształtu mającego środek w centrum obrazu. W rozwiązaniu uwzględnione są dwa przypadki:

- dwóch graczy - w tym przypadku część wspólna jest prostokątem rozciągniętym na całość obrazu. Wysokość prostokąta ustawiana jest przez użytkownika jako odchyłka od centrum (n pikseli w górę i dół),
- trzech i więcej graczy - w tym przypadku część wspólna jest kołem o promieniu ustawianym przez użytkownika.

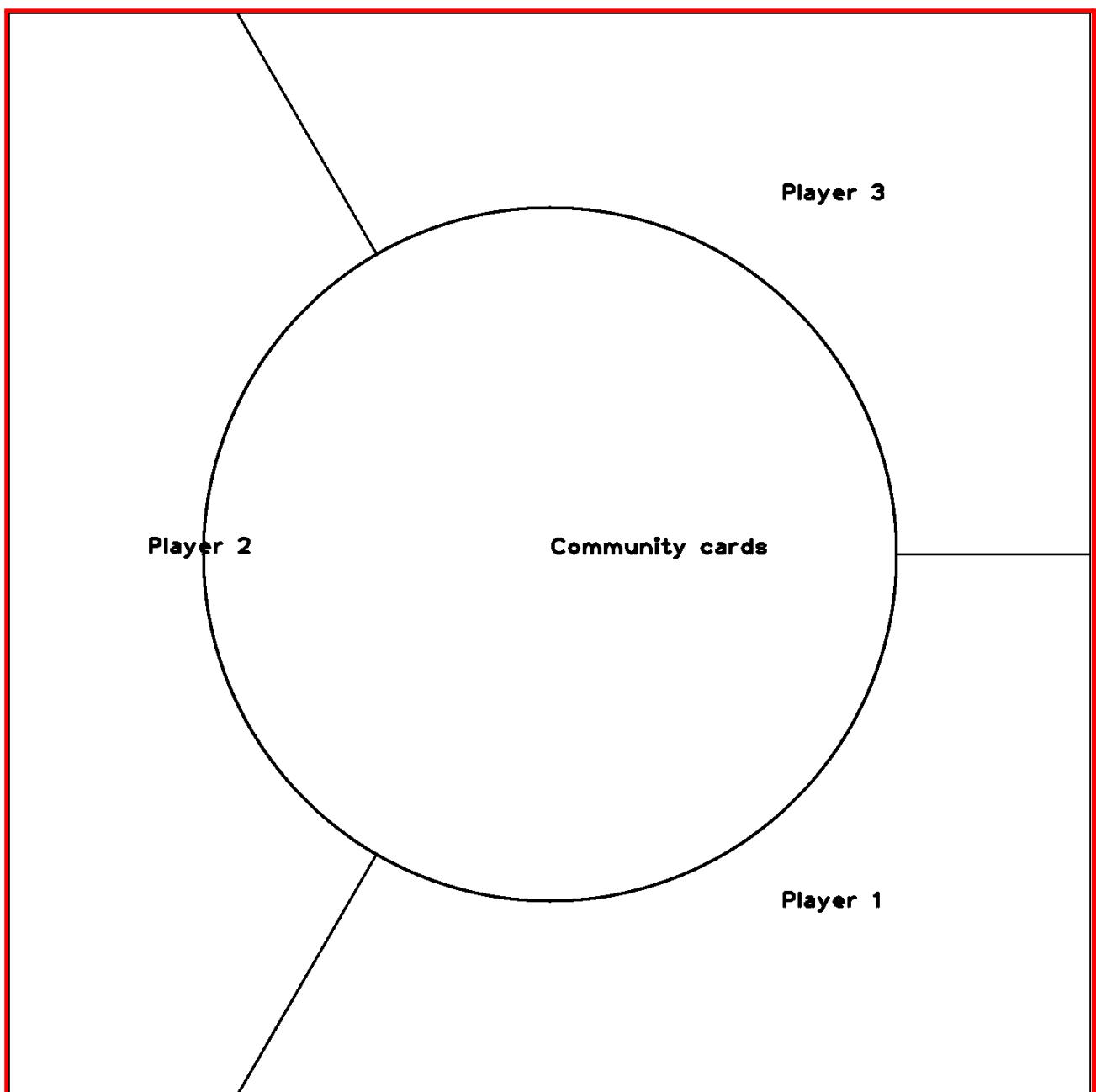
Kształt określony dla konkretnego przypadku nakładany jest na czarny obraz. Po wyprogowaniu obrazu powstaje maska.



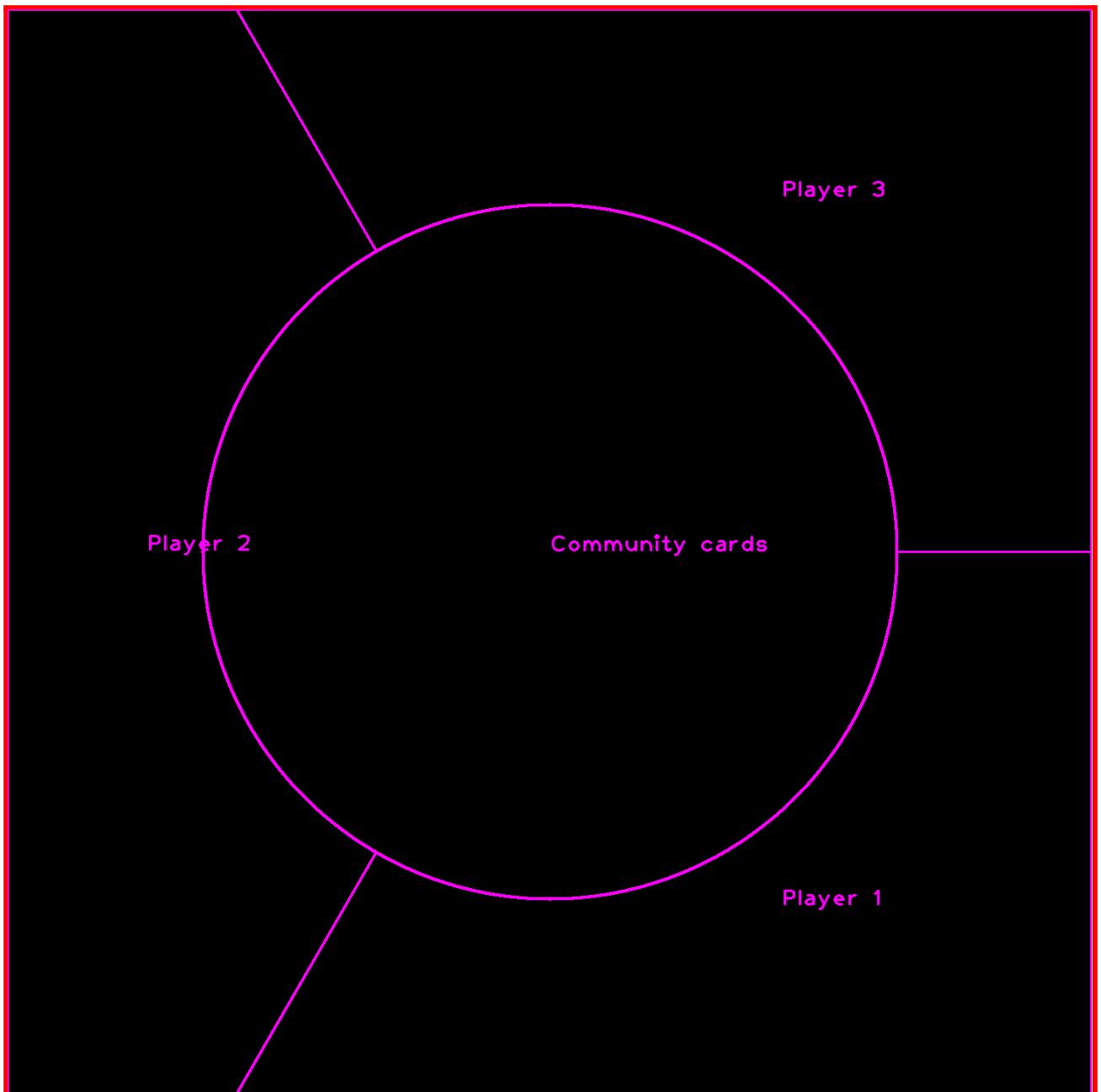
Grafika 16 - maska dla części wspólnej

5.3.3 Nakładanie informacji na obraz

Aby nakładać informacje o podziale stołu na obraz tak, żeby nie zakłócały detekcji zastosowano zdjęcie nakładkę. Aby umieścić taką nakładkę na obraz, należy wpierw użyć maski (będącej wyprobowaną nakładką) i potem dodać maskę do obrazu docelowego. To rozwiązanie umożliwiło wyłączenie opisu podziału stołu z detekcji kart i żetonów.



Grafika 17 - maska nakładki podziału stołu



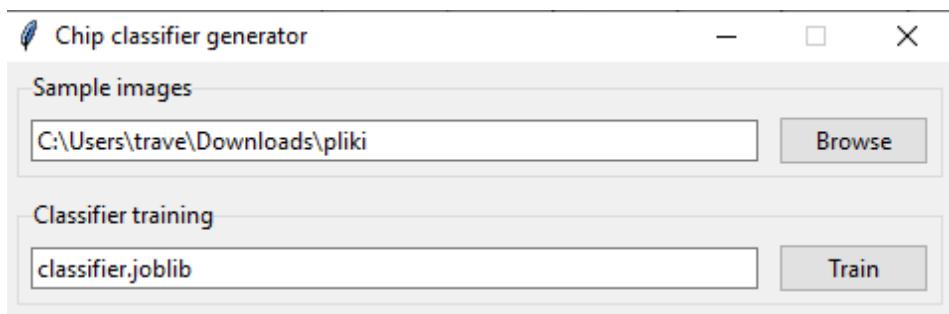
Grafika 18 - nakładka podziału stołu

Informacje o wykrytych kartach i żetonach nakładane są na wskazany obraz w trakcie detekcji (zazwyczaj obraz źródłowy). Takie podejście mogło być zastosowane, ponieważ założone zostało, że na kartach nie będą znajdująć się żetony i na żetonach nie będą znajdująć się fragmenty kart.

6. Instrukcja użytkowania aplikacji

6.1 Generator klasyfikatora

W celu stworzenia klasyfikatora należy uruchomić plik wykonywalny
`chip_classifier_generator_ui.exe`



Grafika 18 - interfejs `chip_classifier_generator_ui.exe`

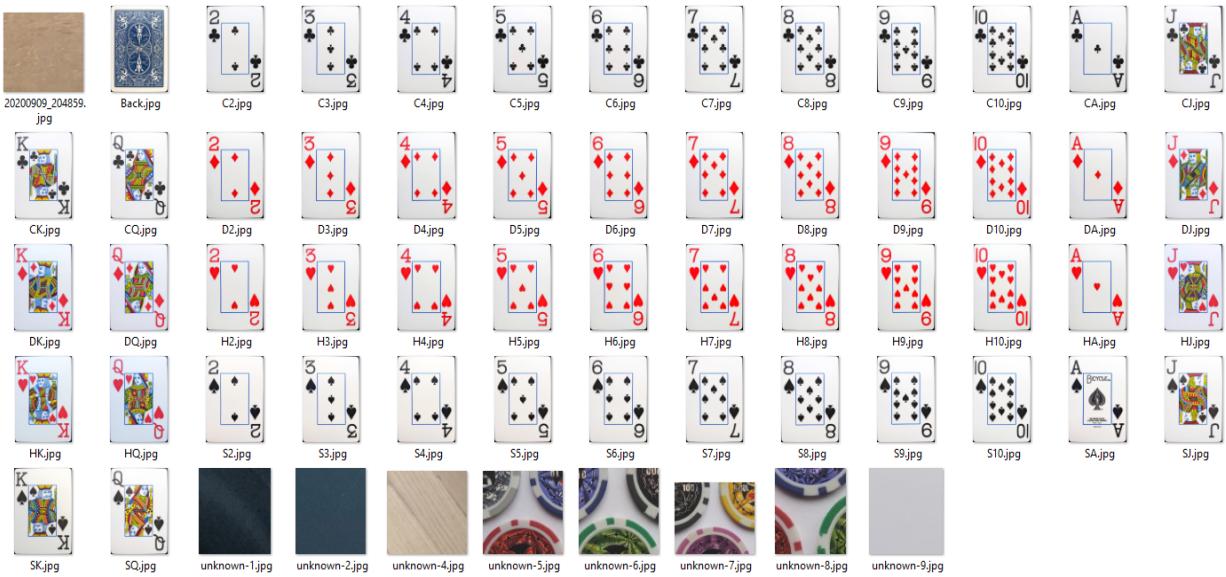
Pojawi nam się interfejs z którego wybieramy ścieżkę do zdjęć treningowych w "sample images" oraz jak nazywamy jak klasyfikator ma się nazywać. Dane do uczenia powinny przyjmować format jpg oraz powinny być uporządkowane w następujący sposób:



Grafika 19 - przykładowy zbiór danych do uczenia

Wycinki kart przeznaczone do wyznaczania wartości powinny znajdować się w folderze głównym oraz być nazwane jak na przykładzie (np spades.jpg oznacza symbol pik, a two.jpg oznacza dwójkę), natomiast zdjęcia żetonów należy umieścić w folderze z nazwą koloru żetonu, gdzie sama nazwa pojedynczego zdjęcia nie ma znaczenia (musi natomiast mieć format JPG). W folderze unknown umieszczamy zdjęcia zawierające rzeczy, których

nie chcemy wyłapywać jako żetony (tzn.: zdjęcia kart, zdjęcie fragmentu stołu, kilka żetonów koło siebie).

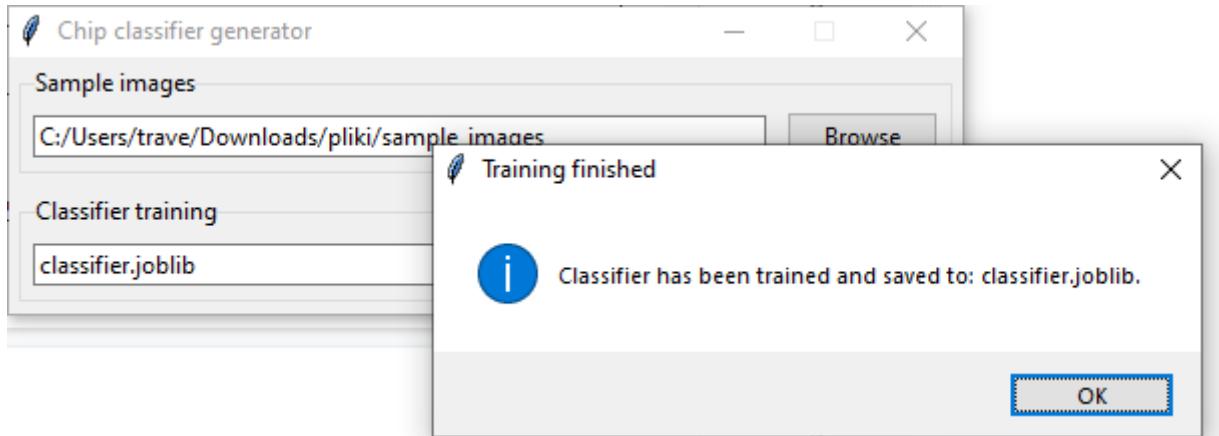


Grafika 20 - przykładowa zawartość folderu unknown

Program posiada przypisane wartości od koloru żetonu. Oznacza to, że np.: żółty żeton ma zawsze wartość 1000.

kolor żetonu	wartość
szary	1
czerwony	5
niebieski	10
zielony	25
czarny	100
fioletowy	500
żółty	1000

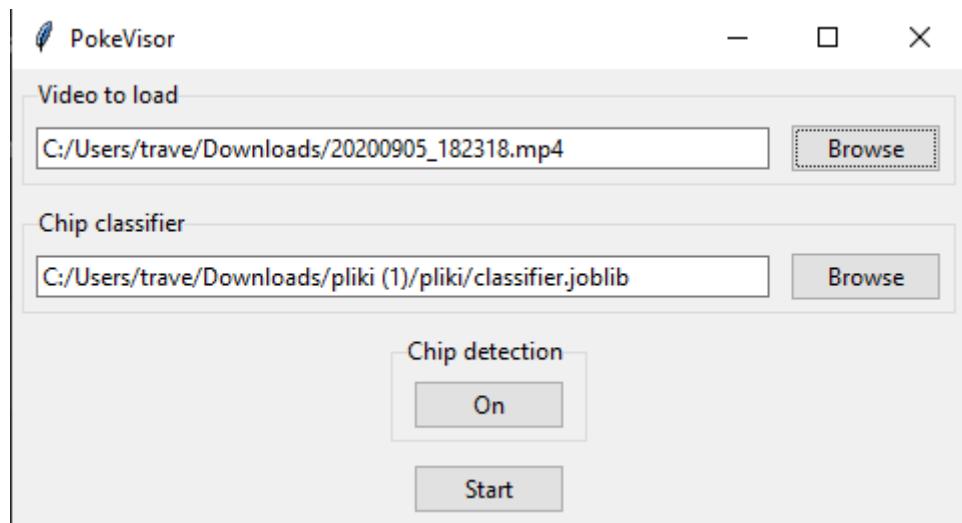
Po wybraniu do wyuczenia oraz nazwy klasyfikatora naciskamy przycisk Train i zaczynamy trenować klasyfikator. Po chwili powinien nam wyskoczyć poniższy komunikat:



Grafika 21 - ukończenie szkolenia klasyfikatora żetonów

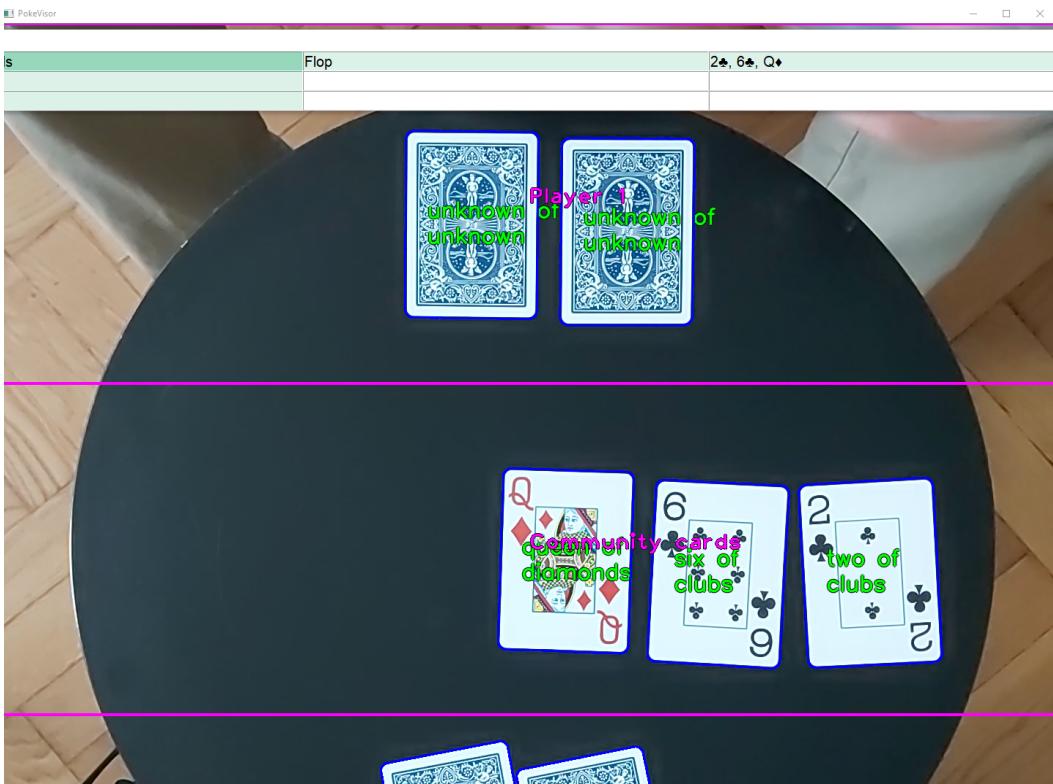
6.2 Analizator nagrań

W celu analizy obrazów należy uruchomić plik `pokevisor_video_ui.exe`.



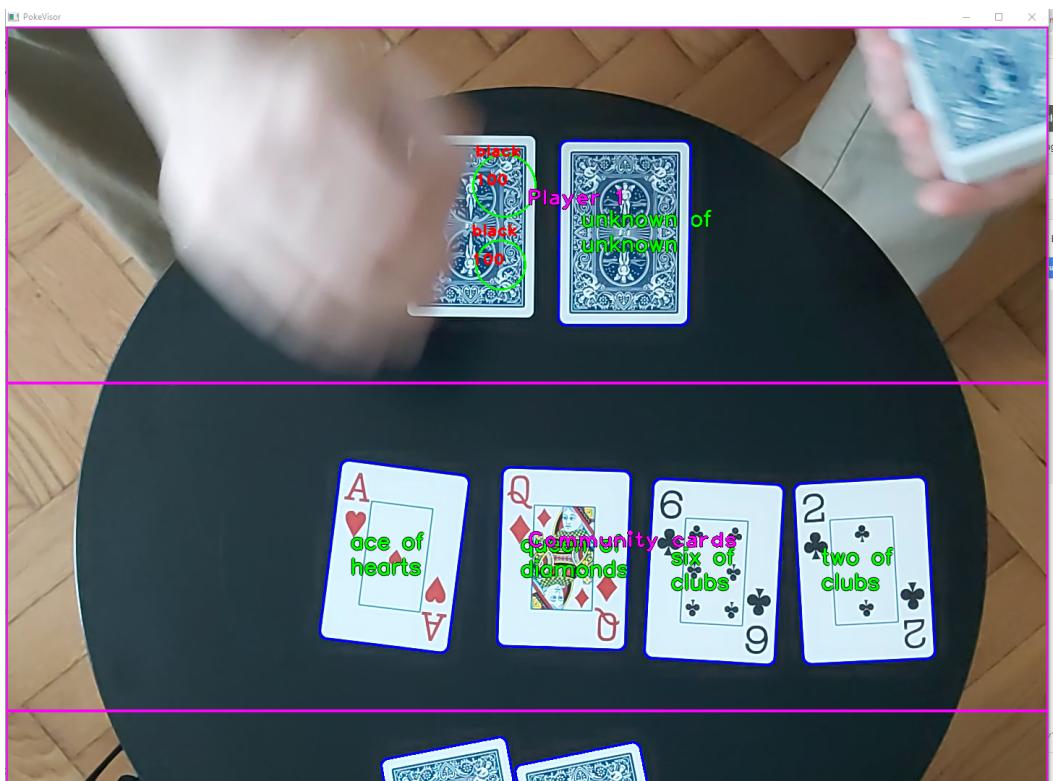
Grafika 22 - interfejs `pokevisor_image_ui.exe`

Pojawi nam się interfejs, na którym wybieramy ścieżkę do filmu oraz ścieżkę do klasyfikatora (który utworzyliśmy wcześniej) oraz czy chcemy wykrywać żetony. Po wybraniu ustawień i naciśnięciu przycisku Start przechodzimy do następnego etapu działania programu.



Grafika 23 - interfejs *pokevisor_image_ui.exe*

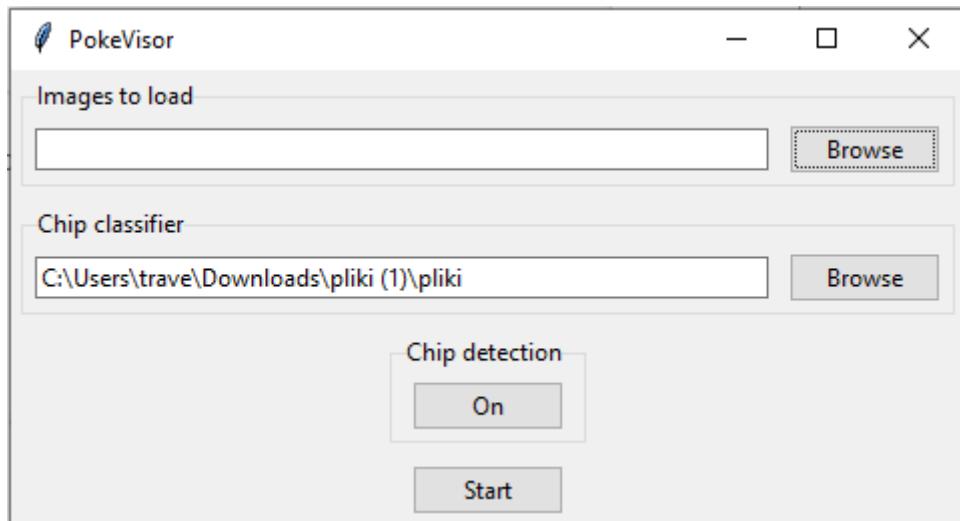
W tym etapie program wykrywa karty i/lub żetony co klatkę i wyświetla je na filmie. Ponieważ wszystko dzieje się w czasie rzeczywistym, program będzie czasami wykrywał na chwilę żetony w nieodpowiednich miejscach.



Grafika 24 - przykład błędного wykrycia żetonów

6.3 Analizator obrazów

W celu analizy obrazów należy uruchomić plik `pokevisor_image_ui.exe`.



Grafika 25 - interfejs `pokevisor_image_ui.exe`

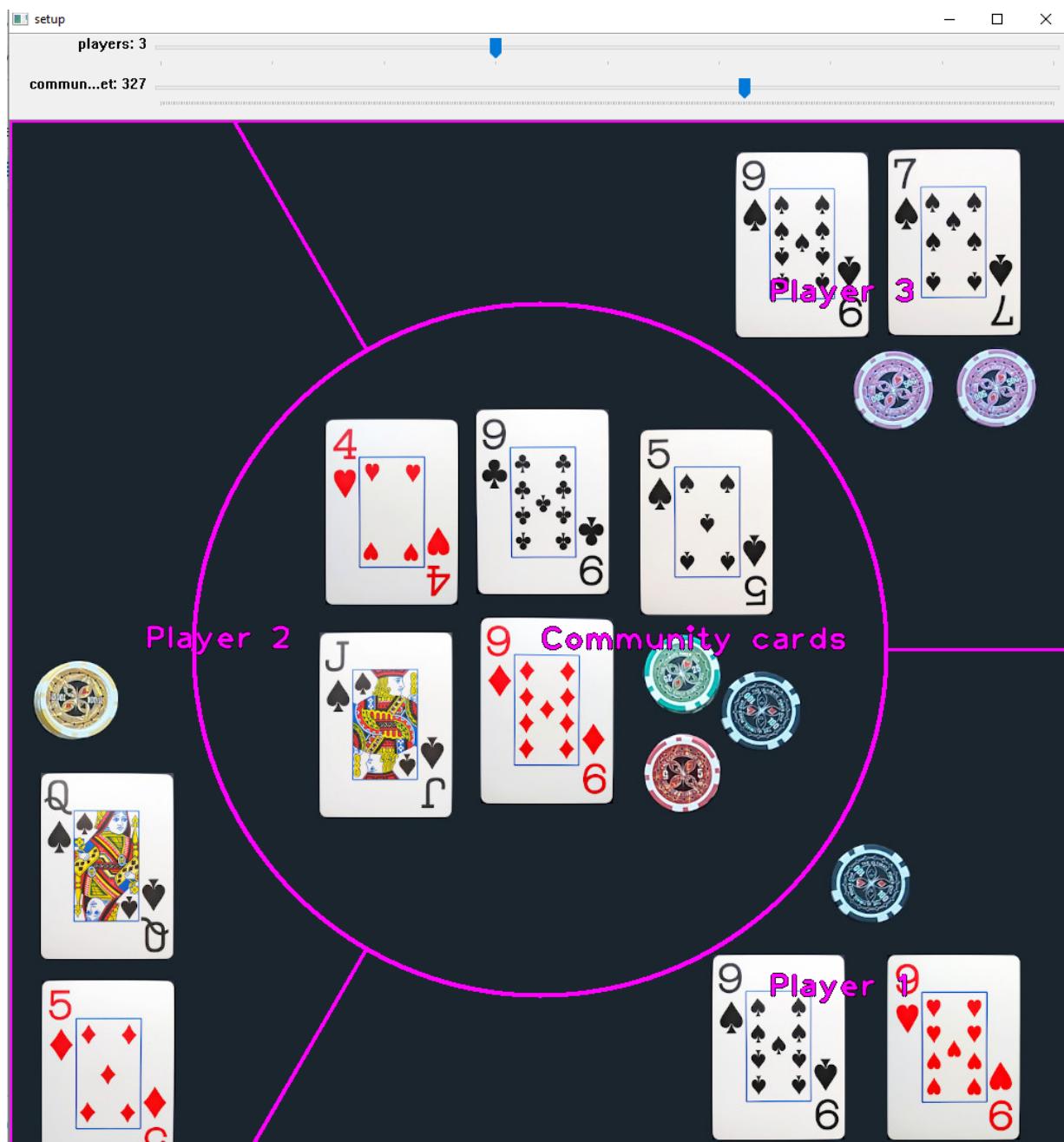
Pojawi nam się interfejs, na którym wybieramy ścieżkę do zdjęć, ścieżkę do klasyfikatora (który utworzyliśmy wcześniej) oraz czy chcemy wykrywać żetony. Po wybraniu ustawień i naciśnięciu przycisku Start przechodzimy do następnego etapu działania programu.



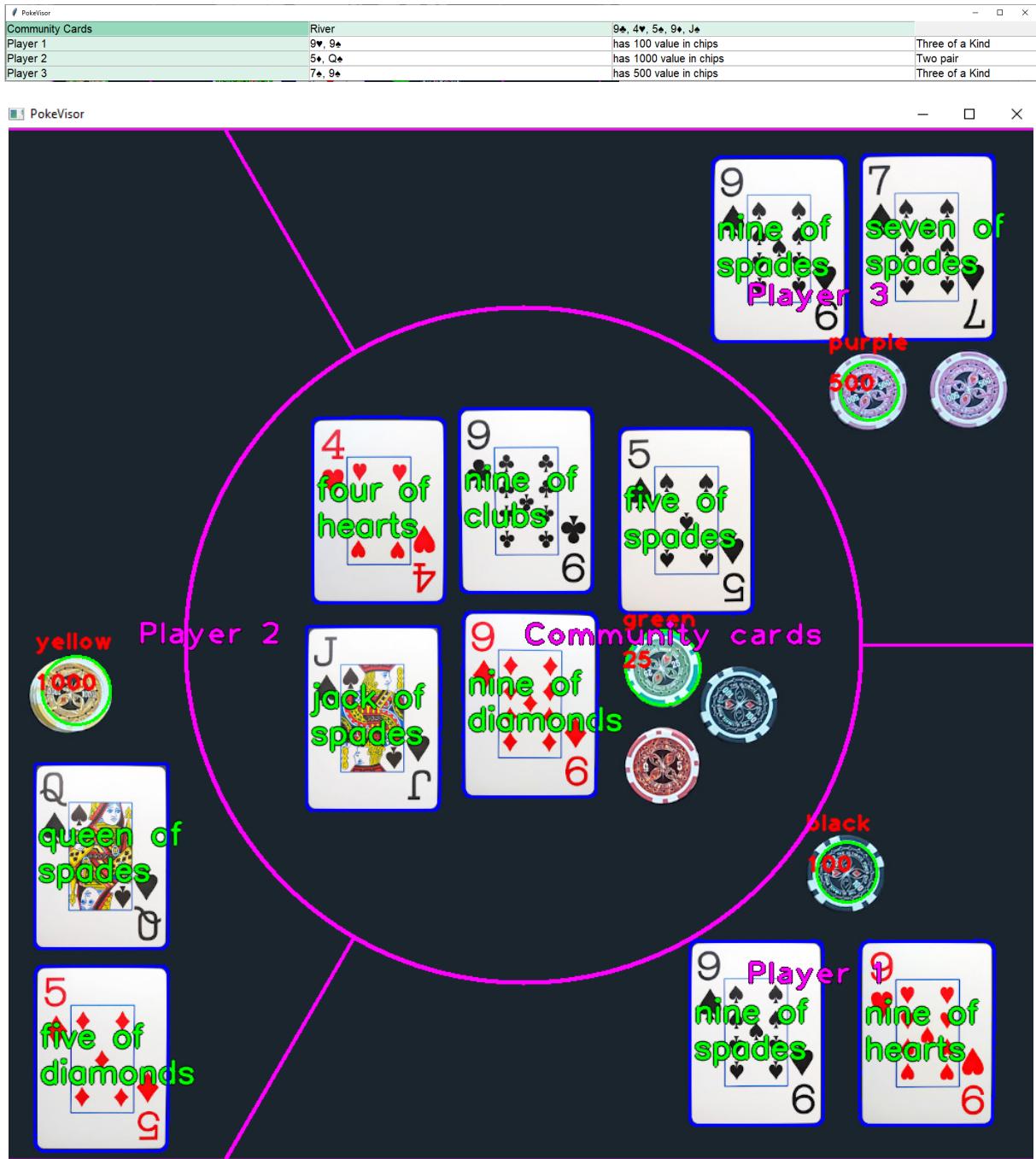
Grafika 26 - interfejs wybierania maski

W tym etapie wybieramy ilość graczy (w powyższym przypadku 3) oraz wielkość maski dla community cards. Ważne jest, aby wybrać odpowiednią rozdzielcość zdjęcia, ponieważ przy za dużej może ono nie zmieścić się na ekranie wyświetlania, a za mała utrudni identyfikowanie kart i żetonów. Drugiemu z problemów można zapobiec, odpowiednio ustawiając wartości w config.json dla spodziewanych wielkości kart i żetonów (np.: obraz 1440x1440 zmniejszony do 1000x1000).

W celu wybrania odpowiedniej maski dla graczy należy przestawić suwak players na odpowiednią wartość. Suwak pod nim powinien być ustawiony tak, aby wszystkie karty wspólne (community cards) znajdowały się w obrębie okręgu (lub prostokąta w przypadku 2 graczy). Następnie należy wcisnąć przycisk "s" na klawiaturze, żeby rozpoczęć wykrywanie.



Grafika 27 - ustawienie suwaka odpowiednie dla przykładowego obrazu



Grafika 28 - wynik wykrywania

Jak można zauważyć wynik wykrywania jest bardzo dobry dla kart, lecz niedokładny dla żetonów (nie wykrywa 1 fioletowego, 1 czarnego oraz 1 czerwonego żetonu). Jest to spowodowane złą konfiguracją pliku config.json, który posiadał poniższe wartości:

```
{  
    "card-detector": {  
        "card-min-area": 12500,  
        "card-max-area": 240000  
    },  
    "chip-detector": {  
        "chip-min-distance": 100,  
        "chip-min-radius": 20,  
        "chip-max-radius": 120  
    }  
}
```

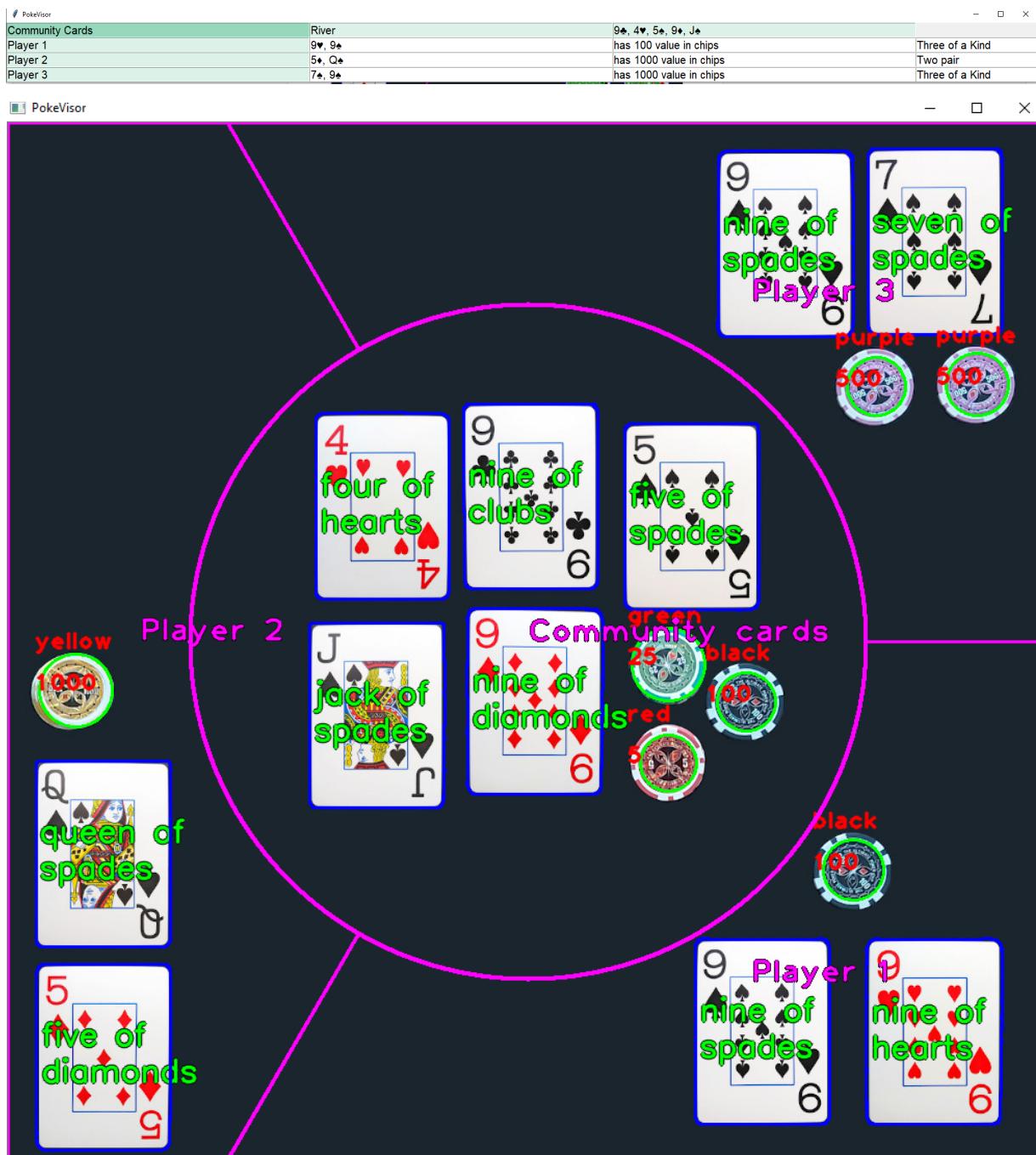
Grafika 29 - zła konfiguracja dla rozdzielczości z przykładu

Wartości te byłyby poprawne, gdyby powyższy przykładowy obraz był w rozdzielczości 1440x1440. Dla powyższego przykładu należy zastosować poniższą konfigurację:

```
{  
    "card-detector": {  
        "card-min-area": 12500,  
        "card-max-area": 240000  
    },  
    "chip-detector": {  
        "chip-min-distance": 80,  
        "chip-min-radius": 15,  
        "chip-max-radius": 110  
    }  
}
```

Grafika 30 - dobra konfiguracja dla rozdzielczości z przykładu

Po poprawieniu konfiguracji program wykrywa wszystkie żetony poprawnie. Zależnie od danych wejściowych (zdjęć bądź nagranie wideo) można zmieniać wartości zmiennych dla uzyskania jak najlepszego efektu. Tak długo, jak użytkownik będzie podawał dane w stałej formie, tak długo nie będzie dodatkowo zmieniać konfiguracji.



Grafika 31 - wynik wykrywania z poprawioną konfiguracją