Checkers

Rozpoznawanie obrazu z gry w warcaby oraz wizualizacja stanu gry na komputerze

Marta Tarka

Adrian Glapiński

Michał Najborowski

Radosław Leszkiewicz

Poznań, 19 lipca 2020 r.

Spis treści

1. Opis aplikacji

Checkers to aplikacja umożliwiająca rozpoznawanie obrazu z gry w warcaby oraz wizualizację stanu gry na komputerze. System pobiera obraz z kamery umieszczonej nad planszą do gry, wykrywa szachownicę oraz pionki ustawione na niej, a następnie wizualizuje stan gry, tj.:

- ustawienie pionków na planszy,
- liczbę pionków każdego gracza,
- określa, który gracz powinien wykonać ruch,
- wyświetla informację, czy poprzedni ruch został wykonany prawidłowo, czy nieprawidłowo i musi zostać powtórzony,
- wyświetla informację o wygranej.

2. Podział prac

Poniższa tabela opisuje wszystkie prace związane z realizacją projektu, z uwzględnieniem członków odpowiedzialnych za poszczególne zadania oraz ewentualnymi uwagami związanymi z ich wykonaniem.

Opis zadania	Osoby realizujące zadanie	Uwagi
Implementacja funkcjonalności odbierania obrazu z kamery w systemie	Marta Tarka	W przypadku uruchamiania aplikacji w systemie Linux, konieczne jest odblokowanie portu TCP 8080
Implementacja funkcjonalności wykrywania planszy	Marta Tarka	
Implementacja	Marta Tarka, Adrian	

funkcjonalności wykrywania pionków	Glapiński	
Wybór rozwiązań technicznych	Adrian Glapiński	Wybór odpowiednich frameworków/bibliotek i języka programowania
Implementacja bazowej wersji interfejsu użytkownika wraz z funkcjonalnością wyświetlania planszy z pionkami	Adrian Glapiński	
Implementacja klasy odpowiedzialnej za równoległe wykonywanie logiki programu na osobnym od GUI wątku	Adrian Glapiński	
Implementacja komunikacji między klasami odpowiedzialnymi za wyświetlanie GUI i za wykonywanie logiki	Adrian Glapiński	Komunikacja odbywa się z wykorzystaniem mechanizmu sygnałów/gniazd (signals/slots)
Wyświetlanie w GUI informacji nt. liczby pionków	Radosław Leszkiewicz	
Wyświetlanie informacji w GUI o aktualnej turze w grze	Radosław Leszkiewicz	
Implementacja logiki odpowiedzialnej za sprawdzanie, czy ruch został	Michał Najborowski	

wykonany poprawnie, ile pionków każdego koloru znajduje się na planszy oraz który gracz powinien wykonać następny ruch		

3. Wykorzystane technologie

Ze względu na problematykę projektu oraz znajomość narzędzi, do realizacji użyto języka Python w wersji 3.8 oraz dostępne biblioteki i narzędzia dedykowane dla tego języka. Zapewnia to stabilność działania systemu, jednocześnie umożliwia prosty rozwój aplikacji, a sam kod źródłowy jest czytelny.

- Python
    - Interpretowany, wieloparadygmatowy, wysokopoziomowy język programowania ogólnego przeznaczenia posiadający rozbudowany pakiet bibliotek standardowych. Jest dynamicznie typowany i zawiera automatycznie zarządzanie pamięcią. Charakteryzuje się czytelnością oraz klarownością kodu źródłowego. Dostępny na licencji *Python Software Foundation License*.
- Qt (PyQT)
    - Darmowy i otwarty zestaw narzędzi oparty o widżety służący do tworzenia graficznych interfejsów użytkownika, jak i również wieloplatformowych aplikacji, które działają na różnych platformach, takich jak Linux, Windows, macOS, czy Android z wykorzystaniem jednej bazy kodu, będąc jednocześnie natywną aplikacją o natywnych możliwościach i szybkości.
    - Biblioteka ta została napisana w języku C++ i jest dla tego języka dedykowana, jednakże istnieje w pełni funkcjonalne API nazywane PyQt, które umożliwia wykorzystywanie narzędzi udostępnianych przez Qt w języku Python.

    - Klasy Qt wykorzystują mechanizm sygnałów/gniazd (signals/slots) do komunikacji pomiędzy obiektami, który umożliwia w łatwy i bezpieczny sposób tworzenie reużywalnych komponentów oprogramowania.
    - Poza bogatą kolekcją widżetów GUI, Qt zawiera również m.in. abstrakcje gniazd sieciowych, obsługę wątków, Unicode, wyrażeń regularnych, bazy danych SQL, SVG, OpenGL, XML.
- OpenCV
    - Biblioteka open source wykorzystywana w systemach wizyjnych i uczeniu maszynowym, napisana w języku C++.
    - Dzięki swojej wydajności (osiągniętej m.in. przez korzystanie z instrukcji MMX i SSE, gdy są dostępne) umożliwia tworzenie aplikacji wizyjnych działających w czasie rzeczywistym.
    - Biblioteka ta jest wieloplatformowa, można z niej korzystać na systemach Linux, Windows, jak i macOS.
    - Biblioteka udostępnia API dla następujących języków: C++, Python, Java i MATLAB.
    - Posiada ponad 2500 zoptymalizowanych algorytmów, w tym kompleksowy zestaw algorytmów wizyjnych i maszynowych. Algorytmy te mogą być wykorzystywane do wykrywania i rozpoznawania twarzy, identyfikacji obiektów, klasyfikowania ludzkich działań w filmach wideo, śledzenia ruchów kamery, śledzenia poruszających się obiektów, wyodrębniania trójwymiarowych modeli obiektów, itp.
- scikit-image
    - Otwartoźródłowa biblioteka do przetwarzania obrazów dla języka Python. Zawiera algorytmy m.in. do segmentacji, transformacji geometrycznych, analiz oraz filtracji, działających na obrazach. Została zaprojektowana do współpracy z bibliotekami numerycznymi i naukowymi NumPy oraz SciPy. Dostępna na licencji BSD, napisana z użyciem języków: Cython, Python oraz C.
- NumPy
    - Biblioteka dla języka Python dodająca wsparcie dla dużych, wielowymiarowych tablic oraz macierzy, wraz z zaawansowanymi

funkcjami matematycznymi, by operować na powyższych strukturach. Numpy jest biblioteką otwartoźródłową, dostępną na licencji BSD. Została napisana w języku C oraz Python.

W procesie wytwarzania oprogramowania zostały wykorzystane następujące technologie/narzędzia:

- Git
    - Rozproszony system kontroli wersji, stworzony przez Linusa Torvaldsa. Stanowi wolne oprogramowania i został opublikowany na licencji GNU GPL v2. Jest przeznaczony do koordynowania prac między programistami, ale może być używany do śledzenia zmian w dowolnym zestawie plików. Celem tego narzędzia jest obsługa rozproszonych, nieliniowych przepływów pracy, zachowanie integralności danych oraz szybkie aktualizowanie zmian.
- GitHub
    - Hostingowy serwis internetowy przeznaczony dla projektów programistycznych wykorzystujących system kontroli wersji Git. Udostępnia darmowy hosting programów otwartoźródłowych i prywatnych repozytoriów. Oprócz standardowych funkcji Git, zapewnia również własne funkcje, takie jak śledzenie błędów, kontrolę dostępu, zarządzanie zadaniami.
- JetBrains PyCharm
    - Zintegrowane środowisko programistyczne (IDE) dla języka Python od firmy JetBrains. Zapewnia m.in. edycję i analizę kodu źródłowego, graficzny debugger, uruchamianie testów jednostkowych, integrację z systemem kontroli wersji. Jest to oprogramowania wieloplatformowe, dostępne na systemy Windows, GNU/Linux oraz macOS. Używany na licencji studenckiej, dostarczonej przez Politechnikę Poznańską.

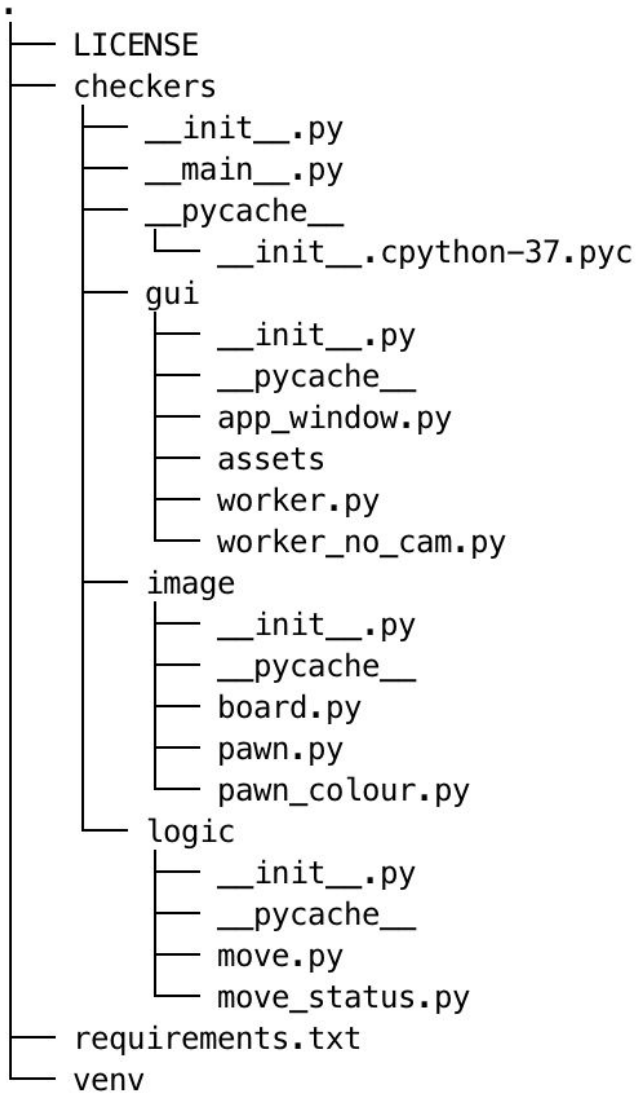
Narzędzia firm zewnętrznych, które są wykorzystywane do działania aplikacji:

- IP Webcam
    - Aplikacja na urządzenia z systemem Android, pozwalająca na udostępnienie dźwięku oraz obrazu z kamery przez sieć, zapisywanie

w wielu popularnych formatach oraz przesyłanie do serwisów udostępniających zdalną przestrzeń plików.

4. Architektura

Aplikacja posiada architekturę opartą o monolit, tym samym jest całkowicie samodzielna pod względem swojego zachowania, jednocześnie zachowując podstawowy podział klas ze względu na zastosowanie. Użycie tej architektury daje gwarancję, że aplikacja nie jest odpowiedzialna tylko za wykonanie pojedynczego zadania, ale potrafi wykonać od początku do końca każde z funkcjonalności wymienionych w założeniach. Chociaż dziś architektura monolitu jest rzadziej preferowana na rzecz architektury opartej o mikroserwisy, była ona wystarczająca do tego projektu, ze względu na jego małą skalę. Na poniższym zrzucie ekranu została przedstawiona hierarchia plików i klas w projekcie:



5. Funkcjonalności

- rejestrowanie obrazu z kamery telefonu
    - przy użyciu aplikacji IP Webcam, obraz zawierający planszę wraz z pionkami jest rejestrowany i przekazywany strumieniowo do modułu odpowiedzialnego za jego analizę oraz graficzną reprezentację
- wyświetlanie obrazu z kamery oraz graficznej reprezentacji planszy
    - obraz, który zostaje odebrany jest wstępnie obrabiany i przycinany, w taki sposób, aby możliwe było otrzymanie samej planszy do gry. Następnie poddawany jest transformacjom, koniecznym do poprawnego odczytania liczby i pozycji pionków każdego koloru, które następnie są mapowane na macierz wartości liczbowych,

reprezentujących pole puste, zajęte przez pionek biały, zajęte przez pionek czarny bądź oznaczające brak zdefiniowanego stanu. Po wykonaniu ruchu utworzona zostaje druga macierz. Różnica tych dwóch macierzy stanowi podstawę do obliczenia poprawności ruchu. Operacja tworzenia macierzy początkowej oraz macierzy końcowej jest inicjowana ręcznie z poziomu interfejsu użytkownika

- interfejs użytkownika
    - interfejs użytkownika zawiera przycisk, który uruchamia procedurę tworzenia macierzy i sprawdzenia poprawności ruchu. Oprócz tego wyświetla podstawowe informacje o stanie gry - liczbę pionków każdego z graczy, który z graczy aktualnie wykonuje ruch oraz zwycięzcę danej rozgrywki
- walidacja ruchu
    - podając na wejście różnicę między macierzą początkową, a macierzą końcową, aplikacja waliduje poprawność wykonania ruchu, dzięki czemu jest sprawdzane m.in. czy gracz, którego jest aktualnie kolej, wykonał ruch, czy liczba pionków po wykonaniu ruchu się zgadza, czy gracz zaatakował przeciwnika, w momencie, gdy musiał wykonać atak. Jeśli poprawność ruchu obliczona przez aplikację zostanie uznana za błędną, ruch będzie musiał zostać powtórzony do momentu, aż będzie poprawny

6. Szczegóły implementacyjne

Podsystem wizyjny

a) Wykrywanie planszy zrealizowano w następujących krokach:
    - pomniejszenie obrazu otrzymanego z kamery do 60% pierwotnego rozmiaru,
    - progowanie obrazu w skali szarości funkcją cv2.threshold z typem progowania thresh binary,
    - erozja obrazu na podstawie uprzednio określonego jądra o rozmiarze 5x5,

- wykrywanie konturów funkcją cv2.findContours,
- przycięcie obrazu do planszy 8x8 pól na podstawie wykrytych konturów.

b) Wykrywanie pionków:
   - na przyciętej kolorowej planszy następuje progowanie funkcją threshold_yen dostępnej w bibliotece skimage.filters,
   - zmiana intesywności kolorów w celu wykrycia czarnych pionków na ciemnych polach - wywołanie funkcji rescale_intensity dostępnej w bibliotece skimage.exposure,
   - usunięcie szumów z obrazu - wywołanie funkcji fastNlMeansDenoisingColored z biblioteki cv2,
   - dla każdego pola planszy wywoływany jest algorytm wykrywający krawędzie Canny Edge Detection z biblioteki cv2,
   - na koniec wywoływana jest funkcja Hough Circles z biblioteki cv2, znajdująca okręgi o podanych parametrach na obrazie w skali szarości, za pomocą transformacji Hough.

c) Określanie koloru pionka (występuje wyłącznie przy pozytywnej walidacji obecności pionka na polu):
   - obraz pola przekształcany jest w skalę szarości,
   - na podstawie wartości histogramu (cv2.calcHist) obliczany jest stosunek liczby pikseli o wartościach w przedziale: 1-100 oraz 101-256 do liczby wszystkich pikseli,
   - ustalono, że
     - gdy powyższa wartość dla pierwszego typu (piksele w przedziale 1-100) przekracza 60% - pionek jest czarny,
     - gdy powyższa wartość dla drugiego typu (piksele w przedziale 101-256) przekracza 60% - pionek jest biały,
     - gdy wartość nie znajduje się w żadnym z przedziałów - niemożliwe jest określenie koloru pionka.

Podsystem graficznego interfejsu użytkownika (GUI)

Implementacja graficznego interfejsu użytkownika oparta jest o dwie podstawowe klasy:

- AppWindow,
- Worker.

Instancje obu klas komunikują się ze sobą z wykorzystaniem mechanizmu sygnałów/gniazd (signals/slots).

AppWindow jest klasą dziedziczącą po dostępnej domyślnie w bibliotece Qt bazowej klasie wszystkich obiektów interfejsu użytkownika - QWidget. Zadaniem AppWindow jest wyświetlanie na ekranie okna z wszystkimi elementami interfejsu; można do nich zaliczyć:

- planszę do gry w warcaby,
- etykieta (label) z obecnym stanem rozgrywki,
- przycisk umożliwiający aktualizację stanu gry.

W konstruktorze AppWindow wywoływane są kolejno dwie prywatne metody:

- __init_worker_thread() - odpowiada za utworzenie instancji obiektu klasy Worker oraz nowego wątku, do którego przeniesiony zostaje nowo utworzony obiekt Worker. Nowy wątek jest tworzony poprzez wykorzystanie konstruktora QThread(), dostępnego w bibliotece Qt. Przed ostatnią instrukcją metody, jaką jest inicjalizacja wątku, łączone są sygnały instancji klasy Worker z odpowiednimi gniazdami instancji klasy AppWindow oraz na odwrót. Dodatkowo, dzięki połączeniu sygnału z metody QThread::started z gniazdem metody Worker::capture_video po rozpoczęciu wątku obraz z kamery jest nieustannie przechwytywany w nieskończonej pętli.
- __init_ui() - odpowiada za utworzenie określonych elementów interfejsu w odpowiednim położeniu oraz inicjalizację ich początkowych wartości. Sygnał przycisku odpowiedzialny za aktualizację stanu planszy łączony jest z gniazdem powiązanym z metodą AppWindow::draw_checkerboard.

Wśród istotnych metod klasy AppWindow można wymienić także funkcję draw_checkerboard, przyjmującą parametr board_matrix (z domyślną wartością None), będący macierzą położeń pionków. Jej działanie polega na wyczyszczeniu zawartości siatki zawierającej pola planszy wraz z pionkami a następnie dodaniu do niej nowych, zaktualizowanych pól na podstawie wejściowej macierzy.

Worker to klasa odpowiedzialna za przetwarzanie logiki programu oraz przechowywanie aktualnego stanu rozgrywki. Dziedziczy ona po bazowej klasie każdego z obiektów biblioteki Qt - QObject. Operacje z wykorzystaniem tej klasy wykonywane są na oddzielnym, drugim wątku - dzięki temu równolegle może być wykonywana główna pętla zdarzeń aplikacji graficznego interfejsu użytkownika Qt oraz pętla odpowiedzialna za nieustanne przechwytywanie obrazu z kamery. Klasa Worker posiada następujące pola służące do przechowywania obecnego stanu rozgrywki:

- before_matrix - przechowuje macierz położeń pionków przed wykonaniem najnowszego ruchu. Początkową wartością jest None.
- after_matrix - przechowuje macierz położeń pionków po wykonaniu najnowszego ruchu. Początkową wartością jest None.
- player_colour - przechowuje wartość typu wyliczeniowego PawnColour; służy do określenia, który z graczy wykonuje w danym momencie ruch. Początkową wartością jest PawnColour.WHITE.

Główną metodą tej klasy jest capture_video(), która służy do wywołania odpowiednich funkcji odpowiedzialnych za kolejno:

- przechwycenie obrazu z kamery i przesłanie sygnału do klasy AppWindow, by wyświetlić obraz w oknie,
- wykrycie planszy i pionków (funkcja ta zwraca macierz położenia pionków, która jest przypisywana do zmiennej after_matrix),
- walidację wykonanego ruchu i wysłanie sygnału do klasy AppWindow w celu zmiany informacji wyświetlanych w oknie (w zależności od wyniku walidacji).

Ostatnia z wymienionych powyżej funkcji to metoda prywatna klasy Worker o nazwie __make_move. W zależności od wyniku walidacji wykonanego ruchu jej działanie różni się; np. w przypadku niepoprawnego ruchu wysyłany jest sygnał aby wyświetlić komunikat o niepoprawnym ruchu i ten sam gracz proszony jest o ponowne wykonanie ruchu - rozmieszczenie wyświetlanych pionków na planszy nie zmienia się, dopóki nie zostanie wykonany poprawny ruch.

Podsystem walidacji ruchu

Implementacja logiki odpowiedzialnej za walidowanie poprawności wykonanego ruchu jest oparta o dwie klasy typu wyliczeniowego:

- *PawnColour* - określa stan danego pola/kolor pionka znajdującego się na tym polu i jej możliwe wartości to *EMPTY* (w przypadku gdy pole jest puste)*, BLACK, WHITE, UNDEFINED* (w przypadku, gdy aplikacja nie jest w stanie jednoznacznie stwierdzić, jaki pionek znajduje się na danym polu). W pliku zawierającym klasę znajduje się również metoda *opposite(colour)*, która po podaniu wartości odpowiadającej danemu kolorowi pionka, zwraca kolor jego przeciwnika,
- *MoveStatus* - określa status wykonanego ruchu i jej możliwe wartości to *CORRECT, INCORRECT, NO_CHANGE, UNDEFINED* (w przypadku, gdy aplikacja nie jest w stanie jednoznacznie stwierdzić, czy ruch został wykonany), *GAME_OVER* (status ten jest zwracany po wykonaniu ostatniego ruchu w danej rozgrywce),

oraz o metody zawarte w pliku *move.py*:

- *check_move(before_matrix, after_matrix, pawn_colour)* - główna metoda, której zadaniem jest zwalidowanie poprawności wykonanego ruchu, jej argumenty zostały opisane w rozdziale poświęconym graficznemu interfejsowi, zwraca pole klasy *MoveStatus*. Logika została podzielona na wiele pomniejszych metod pomocniczych, które zostają wywoływane przez metodę *check_move*. Na początku na podstawie obu macierzy, pobierane są pozycje początkowe i końcowe jednego ruchu dla obu kolorów pionków poprzez wywołanie metody *get_old_and_new_positions(before, after, pawn_colour)*. Na podstawie powyższych, zostają przeprowadzone wstępne walidacje, które mają zadanie wyeliminować najczęściej występujące błędy w rozgrywce przed przeprowadzeniem bardziej złożonej walidacji:
    - nie został wykonany żaden ruch - zostaje zwrócony status *NO_CHANGE*,
    - brakuje jednego lub więcej pionków - zostaje zwrócony status *UNDEFINED*,

- pionek przeciwnika został przesunięty podczas tury gracza - zostaje zwrócony status *INCORRECT*,
- więcej niż jeden pionek gracza został przesunięty - zostaje zwrócony status *INCORRECT*.

Następnie zostaje wykonana seria operacji:

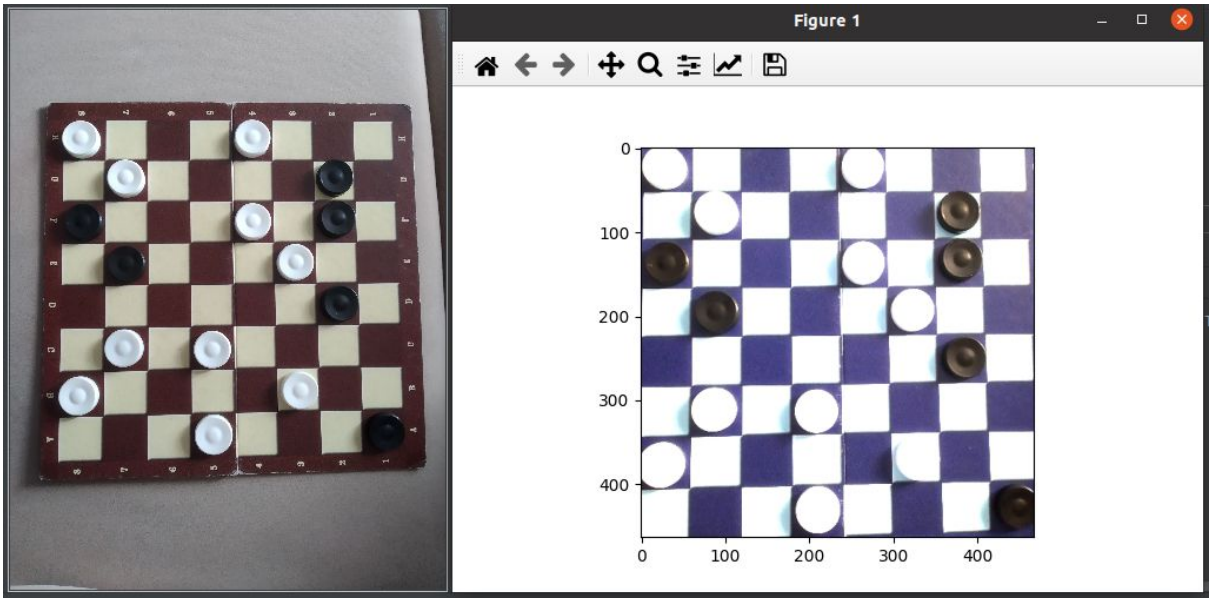
- pobranie pary poprawnych, możliwych pozycji końcowych w odniesieniu do pozycji sprzed wykonania ruchu dla pionka oraz wartości logicznej określającej, czy ruch był atakiem, poprzez wywołanie metody *get_correct_next_positions(before_matrix, old_position, opponent_color).* Metoda ta uwzględnia, czy gracz, który wykonuje ruch, musi zaatakować przeciwnika,
- sprawdzenie, czy faktyczna pozycja pionka po wykonaniu ruchu jest jedną z poprawnych, możliwych pozycji końcowych pobranych w poprzednim punkcie oraz sprawdzenie, czy jest to ruch atakujący ostatni pionek przeciwnika; jeśli nim jest, to zwracany zostaje status *GAME_OVER;* jeśli nim nie jest i faktyczna pionka jest jedną z poprawnych, możliwych pozycji końcowych, to zwracany jest status *CORRECT*; jeśli nie występuje żadne z powyższych, zwracany jest status *INCORRECT*

7. Trudności w implementacji

Pierwszą napotkaną trudnością okazało się dobranie algorytmu wykrywania szachownicy. Większość rozwiązań przedstawionych w projektach innych użytkowników była dostosowana do konkretnej planszy, nie sprawdzała się więc w przypadku innej szachownicy, zmienionej skali, czy też oświetlenia, niż dobrane przez autora. Dodatkowym utrudnieniem było posługiwanie się planszą nieposiadającą konkretnej granicy oddzielającej pola graniczne od ramki. W rezultacie zaprojektowano własne rozwiązanie, bazujące na progowaniu obrazu, erozji oraz filtrowaniu ze znalezionych konturów takich obszarów, które, ze względu

na rozmiar, z dużym prawdopodobieństwem mogą być polami planszy. Następnie obliczano współrzędne rogów szachownicy i przycinano obraz.

Następną trudnością, również z zakresu przetwarzania obrazów, było wykrywanie czarnych pionków na ciemnych polach planszy. Zaobserwowano, iż w takim przypadku dotychczas zaimplementowany algorytm, składający się z wykrywania krawędzi (Canny Edge Detection) oraz wyszukiwania okręgów transformacją Hough (HoughCircles), jest niewystarczający. Zdecydowano się więc na połączenie metody threshold_yen z biblioteki skimage.filters oraz rescale_intensity z skimage.exposure w celu zmiany intensywności kolorów. Otrzymany efekt przedstawiono poniżej.



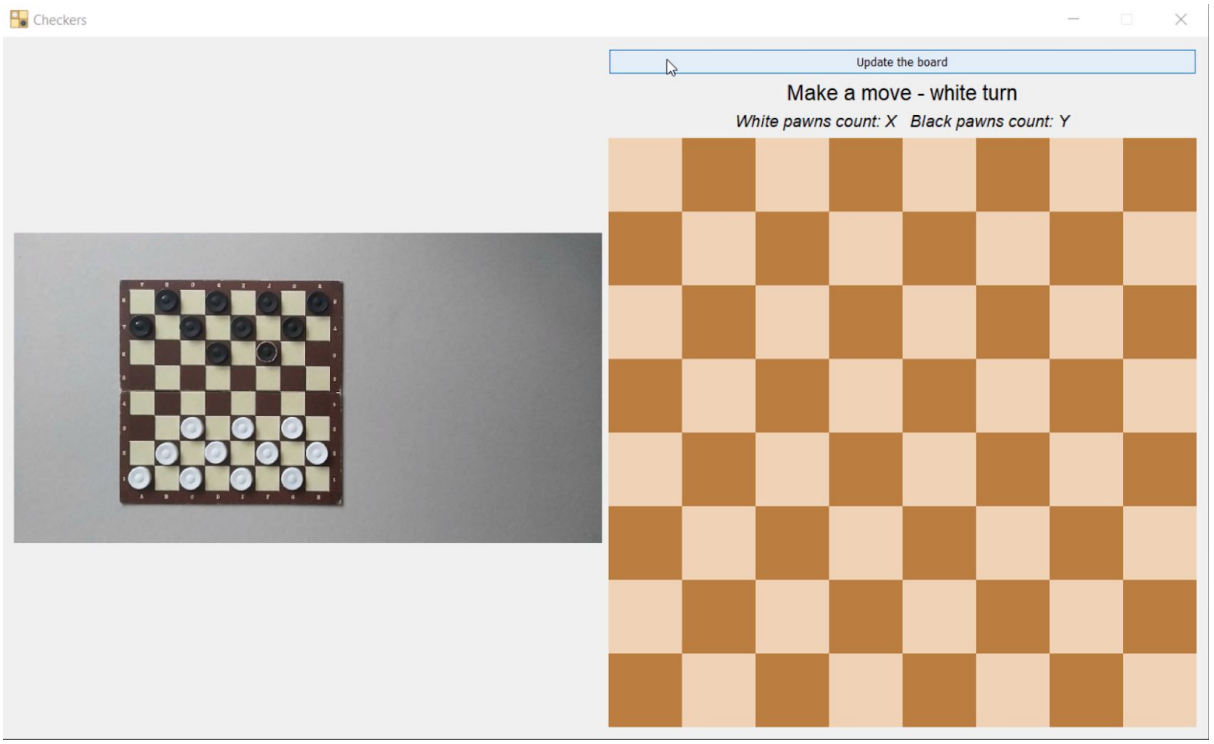
Wówczas wywołanie powyższego algorytmu wykrywania pionków odniosło pożądany skutek - wszystkie pionki zostały prawidłowo oznaczone.

8. Instrukcja użytkowania aplikacji

Poniżej zilustrowano przebieg rozgrywki przy użyciu aplikacji, z uwzględnieniem wykonywania przez graczy zarówno poprawnych, jak i niepoprawnych ruchów, zgodnie z zasadami gry w warcaby. Film z całej rozgrywki, ze względu na ograniczenia wielkości plików przesyłanych mailem, został umieszczony na dysku

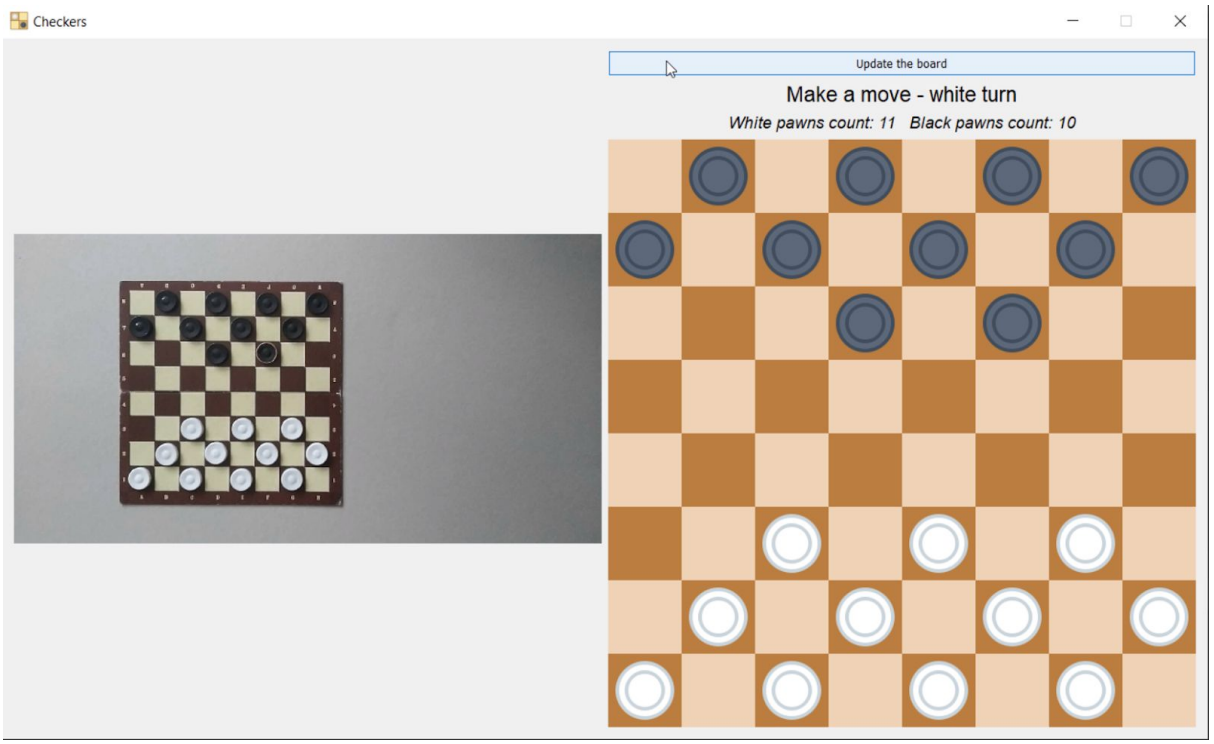
google i jest dostępny pod adresem: Podstawy teleinformatyki - projekt, grupa L45-Z6.

1. W celu rozpoczęcia rozgrywki oraz wizualizacji z użyciem aplikacji konieczne jest odpowiednie ustawienie kamery nad planszą do gry oraz pionków na poszczególnych polach, a następnie wciśnięcie przycisku znajdującego się powyżej wizualizacji planszy z napisem: “Update the board”.



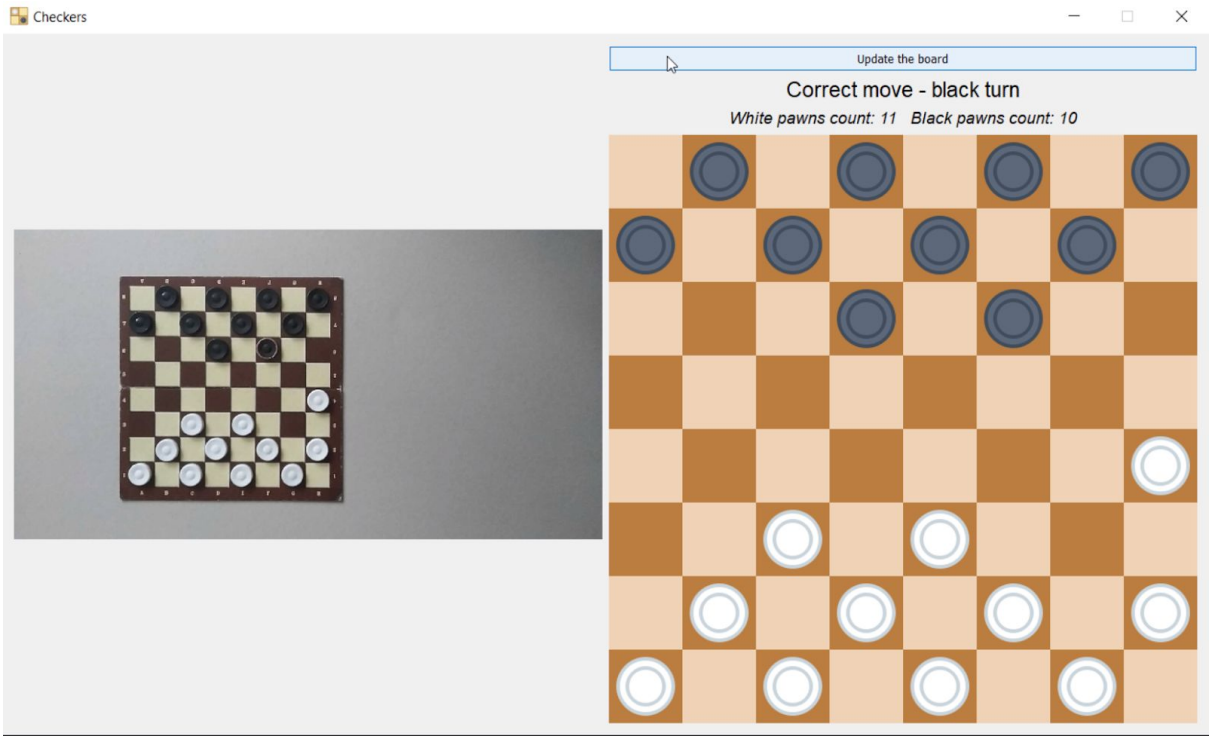
2. Po aktualizacji planszy zwizualizowany jest początkowy układ pionków. Powyżej planszy znajdują się informacje:

- o kolorze gracza, którego jest obecny ruch,
- liczbie białych pionków na planszy,
- liczbie czarnych pionków na planszy.



3. Po każdorazowym wykonaniu ruchu przez gracza konieczne jest ponowne wciśnięcie przycisku “Update the board” w celu aktualizacji aktualnego stanu na planszy oraz sprawdzeniu poprawności ruchu.

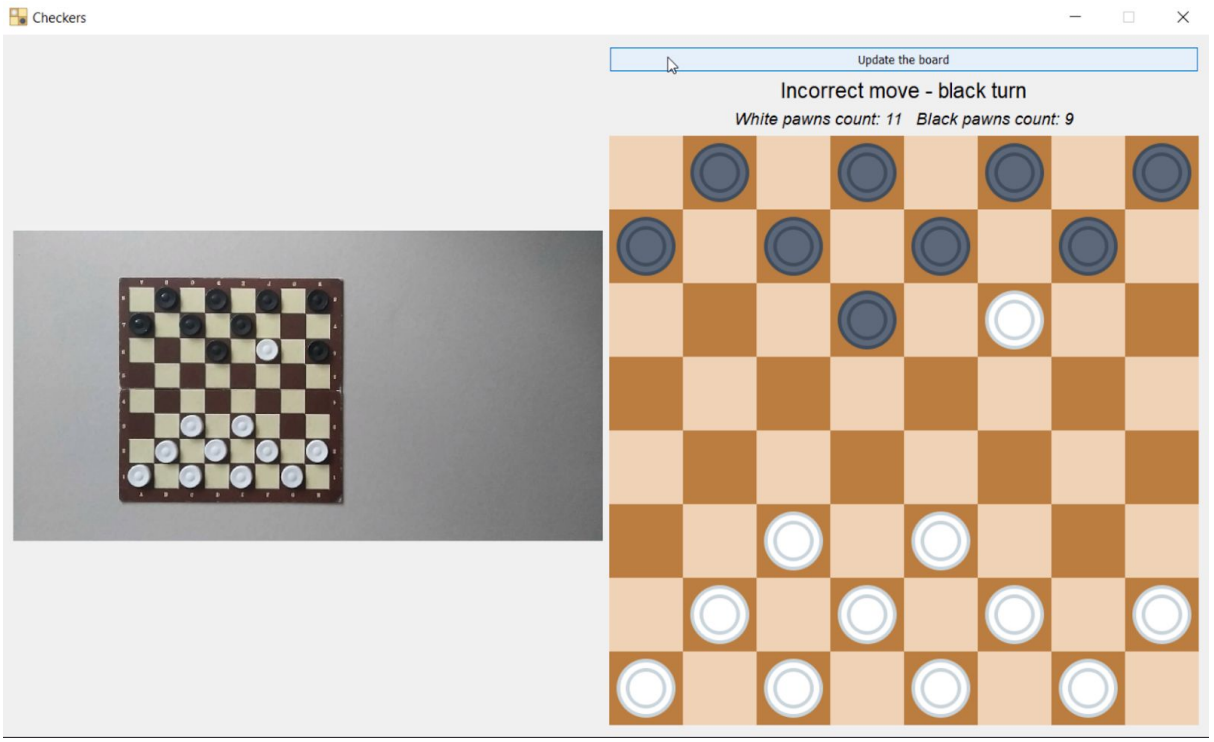
Na poniższej ilustracji przedstawiono ruch gracza białego wykonany z pola G3 na H4. Po aktualizacji planszy widok z kamery jest równoważny ze stanem wizualizacji, natomiast nad planszą widnieje napis: “Correct move - black turn” informujący o poprawnym ruchu gracza białego i aktualnej turze przeciwnika.



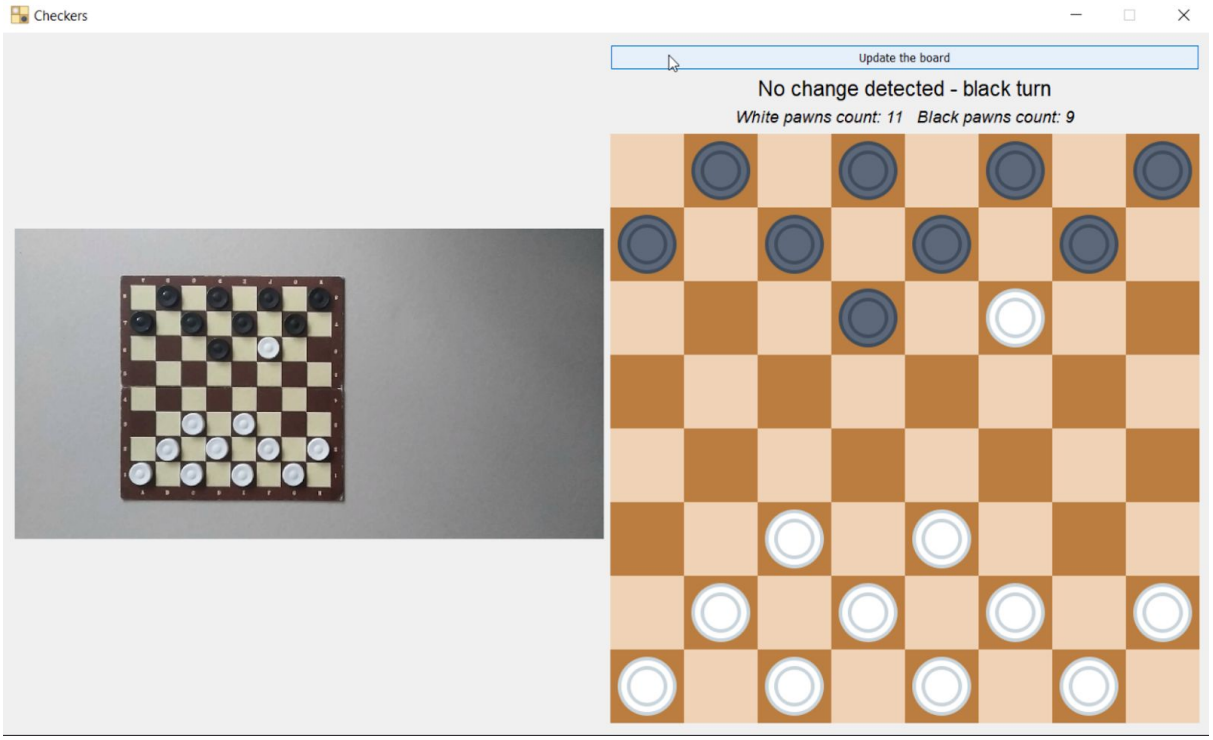
4. Na poniższej ilustracji przedstawiono przykład próby wykonania niepoprawnego ruchu. Zawodnik grający czarnymi pionkami wykonał ruch z pola G7 na H6, kiedy zgodnie z zasadami powinien zbić biały pionek znajdujący się na polu F6. W związku z tym miał dwie możliwości ruchu:

- E7 -> G5,
- G7 -> E5.

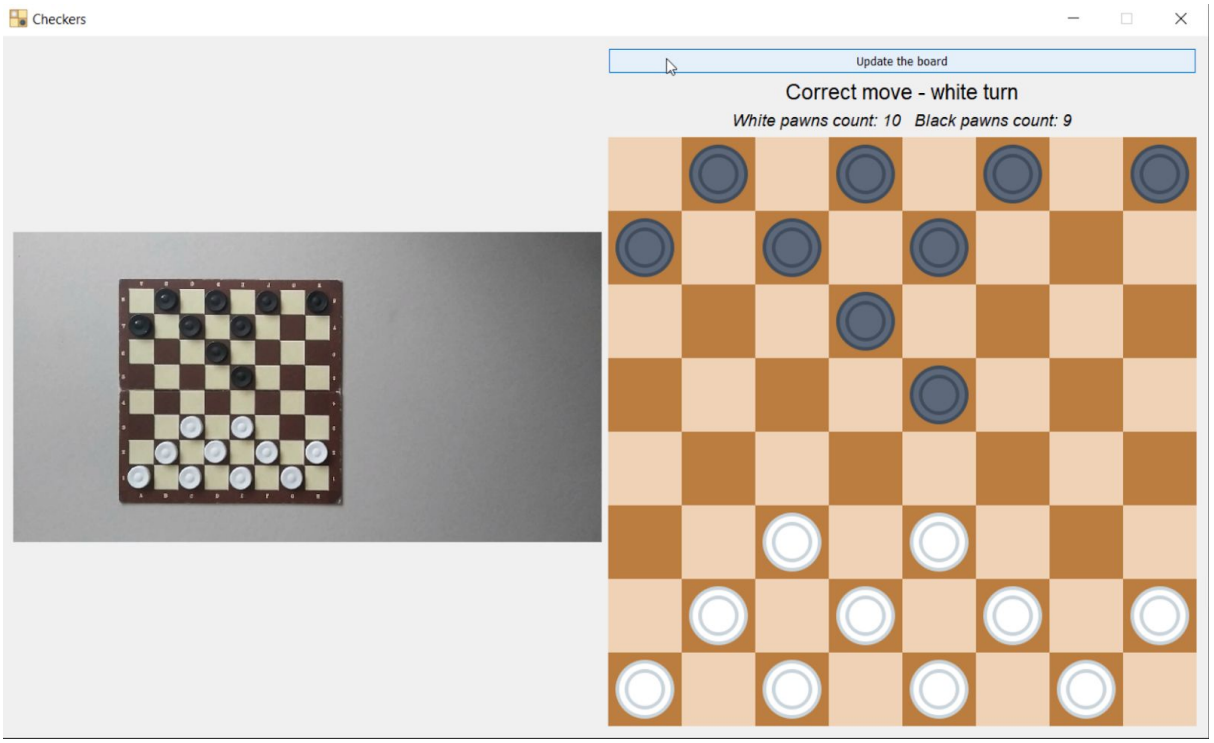
Po wciśnięciu przycisku “Update the board” nad planszą pojawił się napis: “Incorrect move - black turn”, informujący o niepoprawnym ruchu gracza oraz konieczności powtórzenia ruchu, a zwizualizowana plansza nie uległa zmianie.



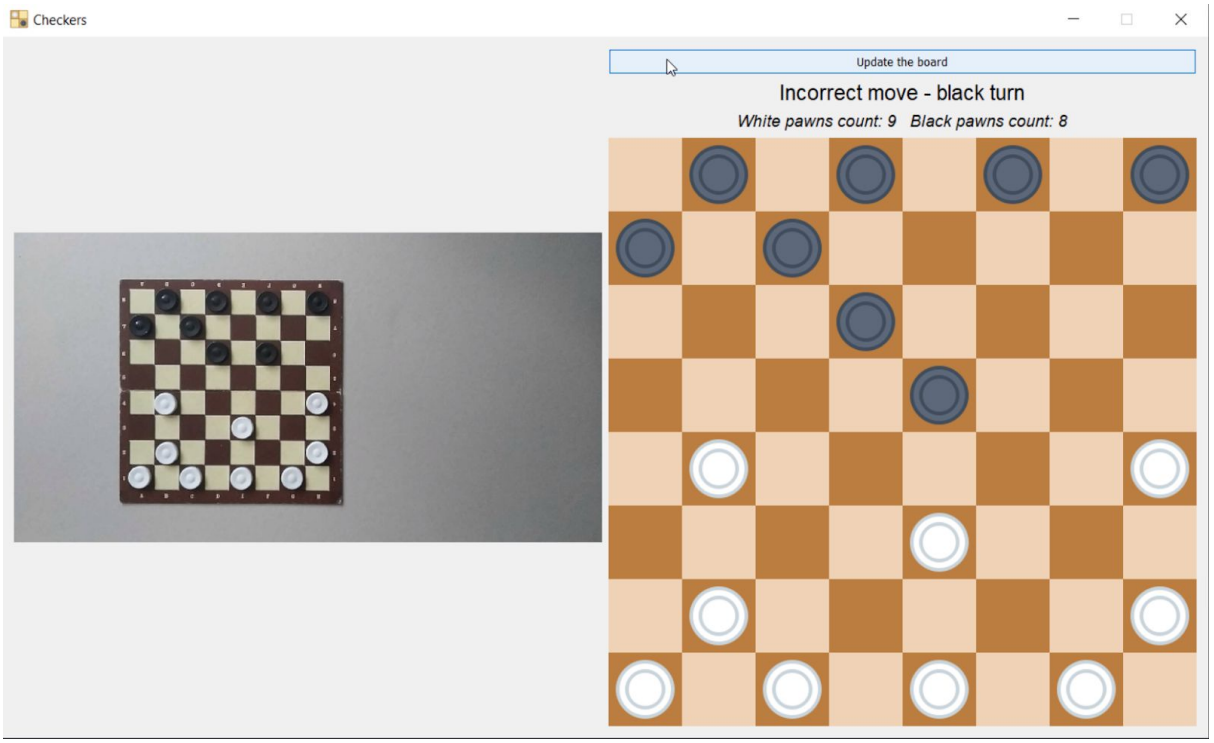
5. Po powrocie czarnego pionka za poprzednie pole oraz wciśnięciu przycisku: “Update the board” zostaje ukazany napis: “No change detected - black turn” informujący o nie wykryciu ruchu gracza nieprzerwanej turze zawodnika grającymi czarnymi pionkami.



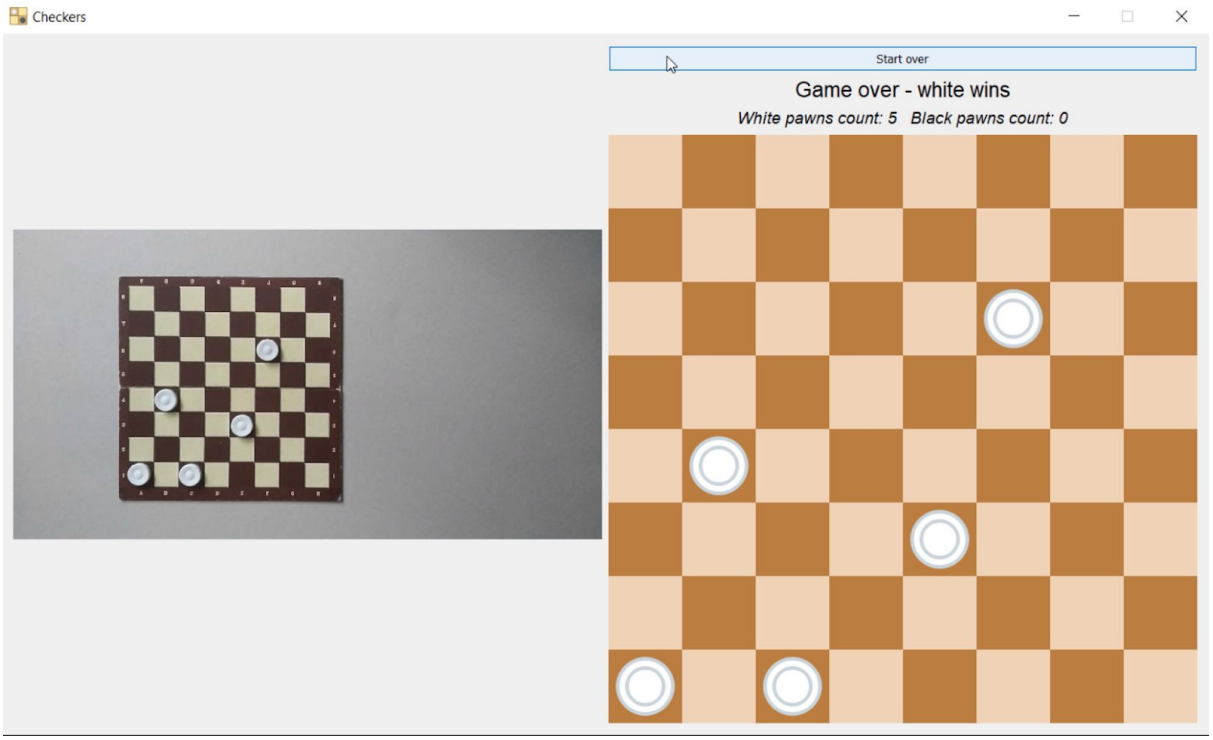
6. W kolejnym kroku przedstawiono wykonanie prawidłowego ruchu przez gracza czarnego, czyli zbicie białego pionka. Po aktualizacji planszy, nad wizualizacją pojawił się napis: “Correct move - white turn”, informujący o poprawnym ruchu gracza oraz zmianie tury na przeciwnika. Poniżej zaktualizowane zostały również liczebności pionków graczy.



7. Na poniższej ilustracji przedstawiono próbę wykonania nieprawidłowego ruchu przez zawodnika grającego czarnymi pionkami - ruch wstecz z pola E5 na F6. Po aktualizacji gracz został ponownie poinformowany o nieprawidłowym ruchu i konieczności powtórzenia, a wizualizacja planszy nie uległa zmianie.



8. Rozgrywka kończy się, gdy wszystkie pionki jednego z graczy zostają zbite. Wówczas wyświetla się napis: “Game over - {white/black} wins” informujący, który z graczy został zwycięzcą. W tym momencie możliwe jest rozpoczęcie rozgrywki od nowa za pomocą przycisku “Start over”, które musi być jednak poprzedzone na nowo ustawieniem wszystkich pionków graczy na pozycjach startowych.


Checkers
Start over
Game over - white wins
White pawns count: 5 Black pawns count: 0