Podstawy Teleinformatyki

*GuziecCheckers* - rozpoznawanie obrazu z gry w warcaby oraz wizualizacja stanu gry na komputerze

Politechnika Poznańska

Informatyka, BSI-1, rok III

Poznań, 09.06.2017

Spis treści

[1. Autorzy projektu 3](#_Toc484905879)

[2. Uzasadnienie wyboru tematu 3](#_Toc484905880)

[3. Podział prac pomiędzy członków zespołu 3](#_Toc484905881)

[4. Wymagania 3](#_Toc484905882)

[5. Opis projektu 4](#_Toc484905883)

[6. Panel konfiguracji (etap pierwszy) 5](#_Toc484905884)

[7. Panel rozgrywki (etap drugi) 6](#_Toc484905885)

[8. Elastyczność aplikacji 7](#_Toc484905886)

[9. Napotkane problemy i zmiany koncepcji 7](#_Toc484905887)

[10. Wybrane technologie 8](#_Toc484905888)

[11. Wybrane środowisko programistyczne oraz narzędzia 8](#_Toc484905889)

[12. Architektura rozwiązania 9](#_Toc484905890)

[13. Zasada działania aplikacji 12](#_Toc484905891)

[14. Instrukcja użytkowania aplikacji 12](#_Toc484905892)

[15. Pozostałe możliwości 14](#_Toc484905893)

[17. Zagadnienia związane z implementacją 16](#_Toc484905894)

# Autorzy projektu

|  |  |  |
| --- | --- | --- |
| **Imię i nazwisko** | **Indeks** | **Email** |
| Tomasz Braczyński | 122053 | tomekbraczynski@o2.pl |
| Daniel Hildebrand | 122048 | daniel20-04@wp.pl |
| Mateusz Korolow | 122030 | mateusz.korolow@gmail.com |

# Uzasadnienie wyboru tematu

Temat ten wybraliśmy z kilku powodów, pierwszym z nich jest chęć przetestowania systemu wizyjnego w oparciu o bibliotekę EmguCV, a był to jedyny przedmiot, który pozwalał nam na takie przedsięwzięcie. Chcemy również stworzyć aplikację, która pomogłaby niedoświadczonym graczom rozwinąć swoje umiejętności poprzez sprawdzanie poprawności wykonywanych ruchów. Ostatnim z powodów jest fakt, że lubimy warcaby.

# Podział prac pomiędzy członków zespołu

1. Projektowanie logiki aplikacji – cała grupa.
2. Planowanie oraz implementacja interfejsu graficznego oraz interakcji   
   z użytkownikami – Daniel Hildebrand.
3. Planowanie oraz implementacja fragmentów kodu programu związanych   
   z wykrywaniem oraz wizualizacją możliwych do wykonania sekwencji ruchów gracza – Tomasz Braczyński.
4. Implementacja zagadnień związanych z przetwarzaniem obrazu przy użyciu wrappera EmguCV, implementacja logiki aplikacji oraz refaktoryzacja kodu – Mateusz Korolow.

# Wymagania

1. funkcjonalne:

* Możliwość wstępnej konfiguracji parametrów przed rozpoczęciem rozgrywki.
* Możliwość rekonfiguracji parametrów w dowolnym czasie trwania rozgrywki.
* Podgląd w czasie rzeczywistym możliwych do wykonania sekwencji ruchów wyświetlanych na dwóch osobnych listach (dla obu graczy z osobna).
* Wizualizacja wybranej (z listy) przez gracza możliwej do wykonania sekwencji ruchów.
* Możliwość ukrycia aktualnie wyświetlanej wizualizacji wybranej sekwencji ruchów poprzez wciśnięcie odpowiedniego przycisku.
* Wyświetlenie linii kalibracyjnych (informujących graczy o poprawnej identyfikacji położenia pól szachownicy).
* Możliwość ukrycia i ponownego wyświetlenia linii kalibracyjnych w czasie rzeczywistym.
* Możliwość delikatnej zmiany pozycji położenia szachownicy w kadrze kamery (np. poprzez przypadkowe jej poruszenie) bez obawy o błędy w interpretacji lokalizacji jej pól oraz błędne wyświetlenie wybranej wcześniej sekwencji ruchów możliwych do wykonania przez danego gracza.

1. pozafunkcjonalne:

* System operacyjny Windows Vista i nowszy.
* Zewnętrzna kamera o rozdzielczości minimum FHD lub smartfon wraz   
  z zainstalowaną aplikacją DroidCam.
* Minimum 512 MB wolnej pamięci operacyjnej.
* Do zachowania płynności przetwarzania obrazu wymagany jest dwurdzeniowy, wielowątkowy procesor o taktowaniu minimum 2 GHz.
* Specjalnie przygotowana szachownica o wymiarach 10x10.
* 24 piony, po 12 na każdego uczestnika rozgrywki znacząco różniące się kolorami. Piony te muszą posiadać mniejszą średnicę niż bok pojedynczego pola planszy.

# Opis projektu

Wybrany przez nas projekt z założenia ma rozpoznawać obraz z gry w warcaby, wizualizować stan bieżącej rozgrywki na komputerze oraz wyświetlać możliwe do wykonania przez graczy sekwencje ruchów. Projekt zakłada umieszczenie przez użytkownika kamery nad planszą do gry podłączonej do komputera. Możliwe jest podłączenie kamery smartfona jako urządzenie zewnętrzne przy użyciu oprogramowania po stronie desktopowej jak i mobilnej o nazwie DroidCam). Podsystem do rozpoznawania obrazów w czasie rzeczywistym aktualizuje informacje o pozycjach pól szachownicy, przynależności pionów graczy używanych w rozgrywce do rozpoznanych wcześniej lokalizacji pól i przekazuje je do podsystemu wizualizacji. Część aplikacji, odpowiadająca za wizualizację, przedstawia podgląd z kamery skierowanej swoim obiektywem na szachownicę, oraz wizualizację wybranej przez użytkownika możliwej do wykonania sekwencji ruchów gracza. Użytkownik ma możliwość wykonania ruchu zgodnego   
z zasadami gry oraz wizualizacji wybranego przez siebie dowolnego ruchu zasugerowanego jako możliwy do wykonania, który znajduje się na liście możliwych sekwencji ruchowych gracza w aplikacji. Projekt składa się z panelu konfiguracji oraz panelu rozgrywki.

# Panel konfiguracji (etap pierwszy)

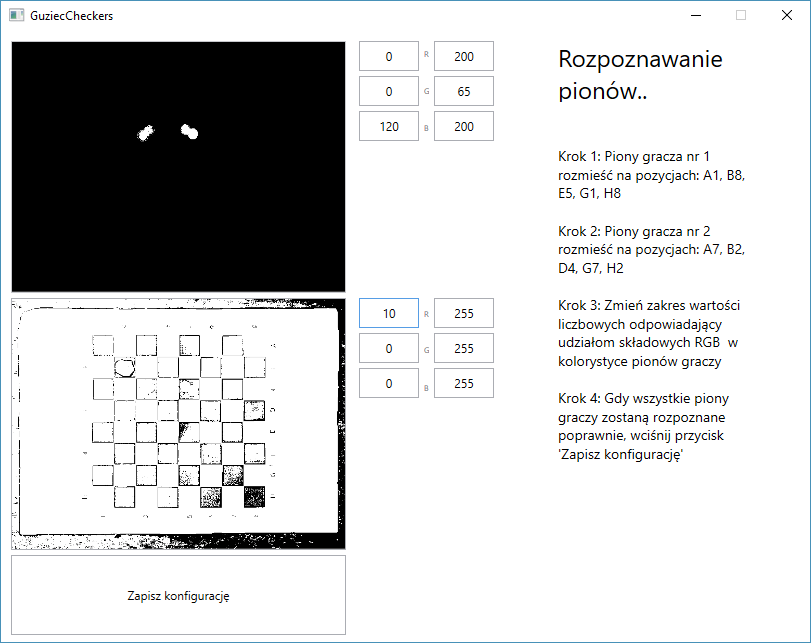
Panel konfiguracji umożliwia dostosowanie parametrów, które są wykorzystywane   
w późniejszej rozgrywce. W panelu tym wyświetlone są dwa okna monochromatycznego podglądu widoku z kamery umieszczonej nad szachownicą. Obok okien znajdują się pola typu textbox (po 6 przy każdym z okien), w które wpisujemy całkowite wartości z zakresu 0 – 255 reprezentujące składowe RGB (po 2 pola dla każdej ze składowych [min, max]), w przedziale których mieszczą się wartości kolorystyczne pionów rozpatrywanego gracza.  
Zmieniając wartości znajdujące się we wcześniej wspomnianych polach aktualizowany jest podgląd z kamery w przynależącym do aktualizowanych pól oknie. Aby poprawnie skonfigurować parametry rozgrywki należy rozstawić na szachownicy piony obu graczy oraz znajdujące się w polach wartości skonfigurować w następujący sposób:

1. Okno pierwsze:

* Kolor biały znajduje się w miejscach, w których na szachownicy rozstawione zostały piony gracza nr 1.
* Kolor czarny zajmuje pozostałą część okna monochromatycznego podglądu   
  z kamery.

1. Okno drugie:

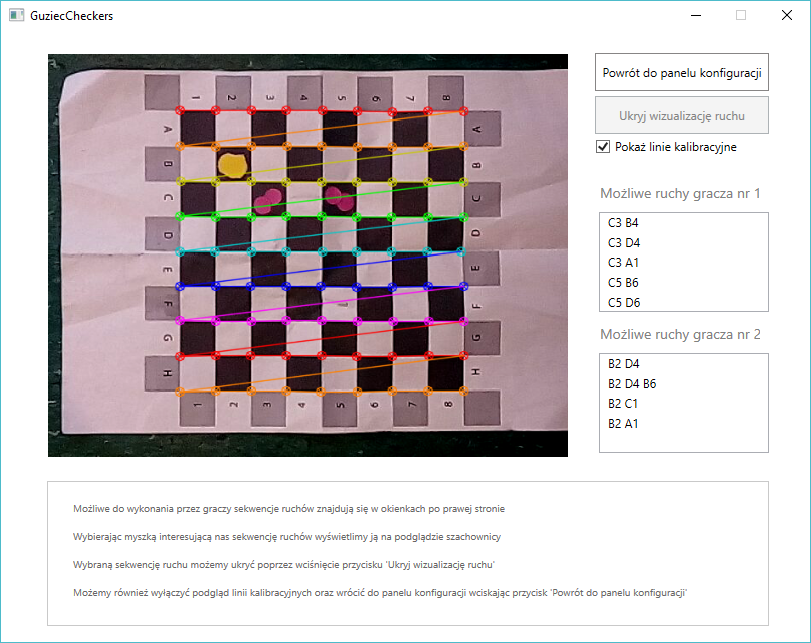
* Kolor biały znajduje się w miejscach, w których na szachownicy rozstawione zostały piony gracza nr 2.
* Kolor czarny zajmuje pozostałą część okna monochromatycznego podglądu   
  z kamery.



*Rysunek 1 – Widok przedstawia wygląd panelu konfiguracji.*

# Panel rozgrywki (etap drugi)

Panel rozgrywki jak sama nazwa mówi umożliwia przeprowadzenie rozgrywki pod nadzorem zaimplementowanego przez nas systemu wizyjnego. Składa się on z okna umożliwiającego podgląd z kamery rejestrującej zmiany na szachownicy w czasie rzeczywistym, dwóch list znajdujących się po stronie prawej, na których wyświetlane są możliwe do wykonania i rozpoznane przez naszą aplikacje sekwencje ruchów (każda lista reprezentuje możliwe do wykonania sekwencje ruchów innego gracza), przycisków realizujących kolejno powrót do panelu konfiguracji oraz ukrycie wyświetlonej wcześniej wizualizacji sekwencji ruchów, pola typu checkbox umożliwiającego włączenie bądź wyłączenie podglądu linii kalibracyjnych oraz sekcji informacyjnej. Kliknięcie dowolnej z wybranych sekwencji znajdujących się na jednej z list spowoduje jej wizualizacje   
w oknie podglądu szachownicy. Aby ukryć wizualizację sekwencji ruchowej trzeba wcisnąć wcześniej wspomniany przycisk. Można również zmienić wyświetlaną wizualizację sekwencji ruchów na inną poprzez kliknięcie wybranej przez nas innej sekwencji znajdującej się na jednej z list.

Rysunek 2 – Widok przedstawia wygląd panelu rozgrywki.

# Elastyczność aplikacji

Utworzona przez nas aplikacja jest napisana w taki sposób, by umożliwić wykorzystanie jej potencjału jak najszerszemu gronu użytkowników. Jest wiele pozytywnych cech charakteryzujących nasz produkt. Do najważniejszych z nich należą:

* Możliwość konfiguracji parametrów charakteryzujących kolorystykę pionów graczy.
* Możliwość powrotu do panelu konfiguracyjnego w trakcie trwania rozgrywki (przydatne w momencie gdy np. warunki oświetlenia w pomieszczeniu, w którym trwa rozgrywka będą się zmieniać w czasie jej trwania).
* Możliwość wykorzystania pionów o różnych kształtach (nie jest wymagane by miały one kształt okręgów, dlatego nie jesteśmy od nich uzależnieni mogąc do gry wykorzystać np. fragmenty kolorowych kartek [ważne jest by piony mieściły się   
  w wymiarach pola używanej szachownicy i zajmowały co najmniej 10% jego powierzchni]).
* Aktualizowanie informacji o położeniu pól oraz przynależących do nich pionów zmienia się w czasie rzeczywistym dzięki czemu nie musimy martwić się   
  o poruszenie używaną do gry szachownicą w momencie trwania rozgrywki a także błędnie wyświetlaną wizualizacją sekwencji ruchów możliwą do wykonania.
* Aplikacja napisana jest w sposób wielowątkowy dlatego nie odczuwamy dyskomfortu związanego z tzw. „zamrażaniem interfejsu” graczy w czasie wymaganego przez funkcjonalność aplikacji operacji przetwarzania obrazu.

# Napotkane problemy i zmiany koncepcji

Jak powszechnie wiadomo nie ma aplikacji, których napisanie nie stwarzało by żadnych problemów. Jako programiści czasami spędzamy więcej czasu przed debuggerem niż pisząc kod. W trakcie realizacji tego projektu napotkano wiele problemów jak   
i wielokrotnie zmieniono metodykę działania w celu osiągnięcia rezultatu, którym jest dzisiejszy wygląd aplikacji. Poniżej wymieniamy najważniejsze z nich:

* Wybór wersji i konfiguracja środowiska Visual Studio do współpracy z wrapperem OpenCV o nazwie EmguCV.
* Problem związany z błędami w implementacji współdzielenia dostępu do kamery pomiędzy stronami panelu konfiguracji oraz rozgrywki w technologii WPF.
* Zmiana sposobu wykrywania pól z metody wykrywającej obiekty typu RotatedRect na metodę FindChessboardCorners(), która zwraca wektor punktów odpowiadający wierzchołkom pól dostosowanej przez nas szachownicy.
* Zmiana sposobu wykrywania pionów znajdujących się na szachownicy z metody wykrywania okręgów na badanie procentowej zawartości pikseli o kolorystyce odpowiadającej zakresowi składowych RGB pionów danego gracza w zakresie lokalizacji wykrytego pola.
* Problemy w implementacji rekurencyjnej metody zajmującej się wykrywaniem   
  i wyświetlaniem możliwych do wykonania przez graczy sekwencji ruchów.
* Problemy związane z wydajnością przetwarzania obrazu z kamery w czasie rzeczywistym utrudniające płynne przeprowadzenie rozgrywki graczom.

# Wybrane technologie

W tej sekcji znajdują się wybrane przez nas technologie użyte do realizacji projektu wraz z wyjaśnieniem naszego wyboru. Są to:

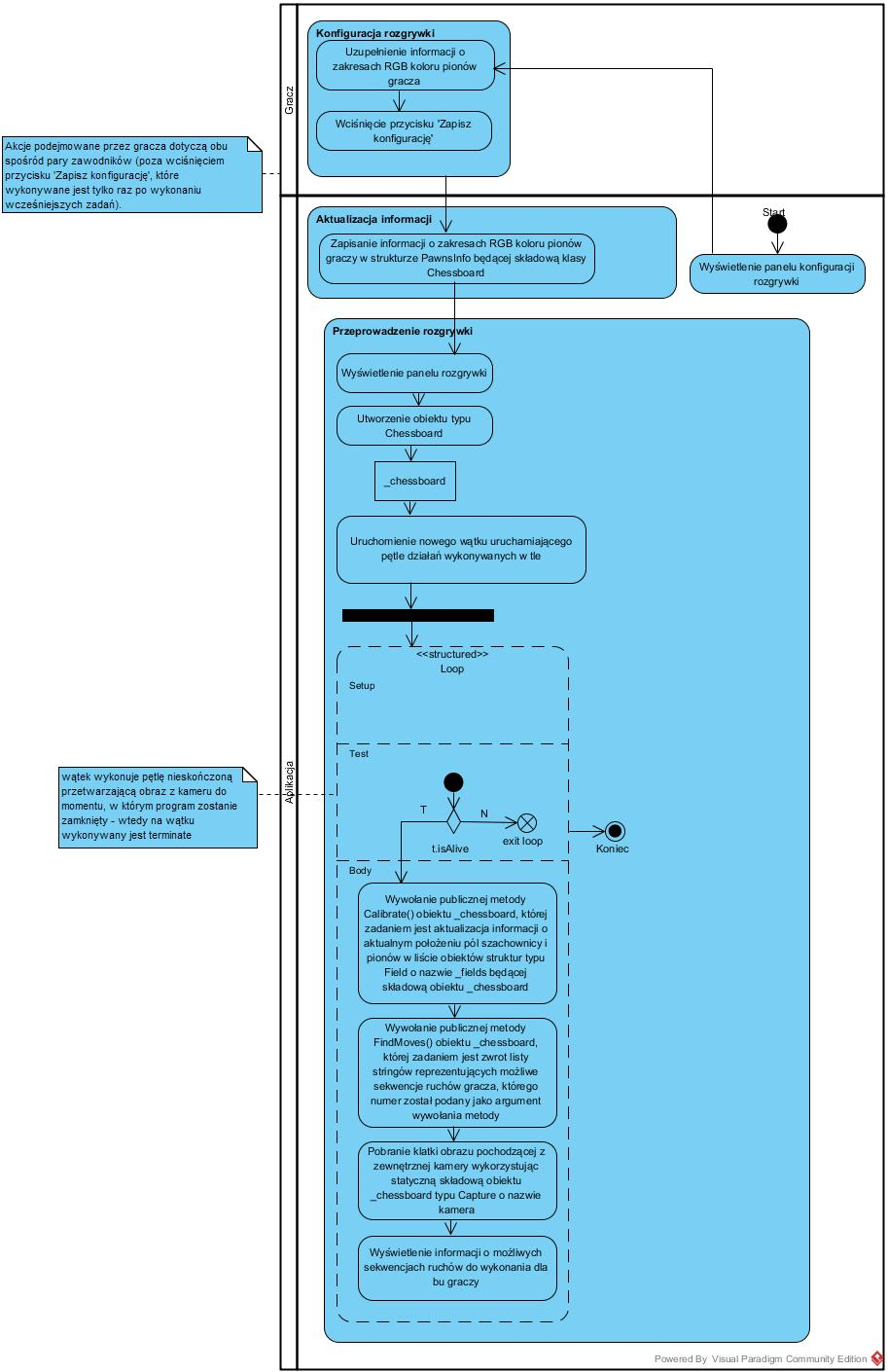
* Obiektowy język programowania C# - został wybrany, ponieważ jest opanowany przez nasz zespół w największym stopniu.
* Silnik graficzny WPF (Windows Presentation Foundation) - został użyty ze względu na wcześniej wybrany język programowania – C#.
* Wrapper EmguCV bazujący na bibliotece OpenCV - został użyty do obsługi kamery. Do tego wyboru skłoniła nas duża ilość dostępnych materiałów elearningowych pozwalających na lepsze opanowanie przechwytywania oraz przetwarzania obrazu z kamery.
* .NET Framework w wersji 4.5.2 został wybrany ponieważ jest w pełni kompatybilny z wrapperem EmguCV.

# Wybrane środowisko programistyczne oraz narzędzia

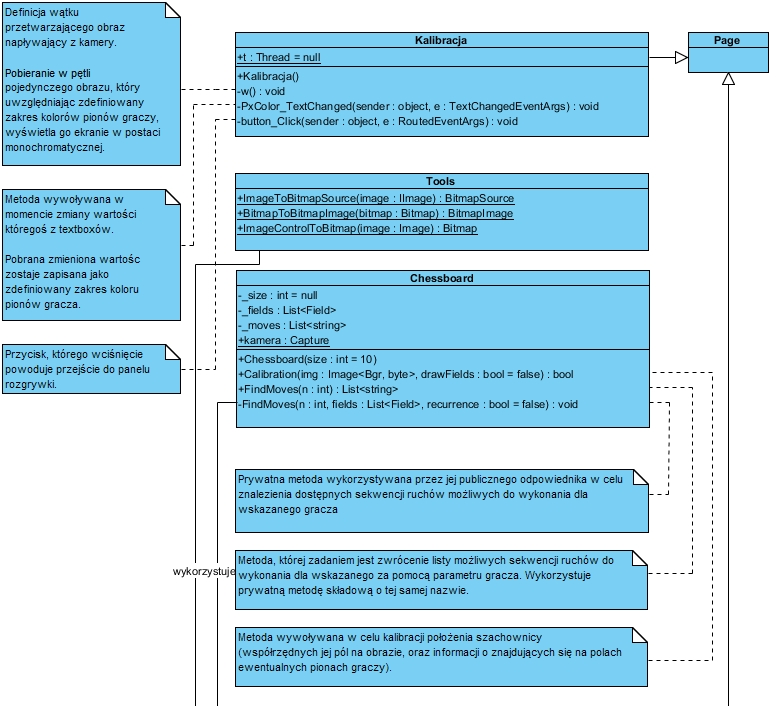
* Visual Studio 2015
* System kontroli wersji Git

# Architektura rozwiązania

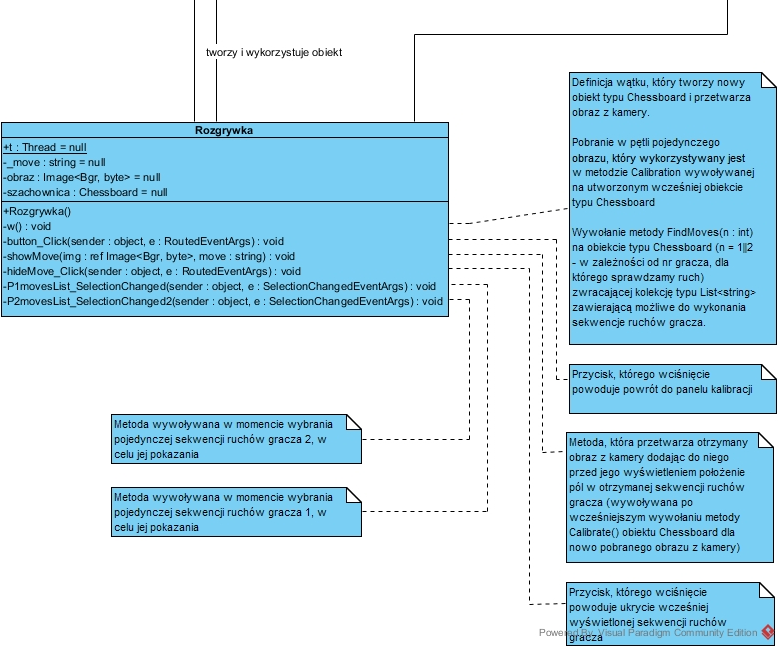
W tej sekcji znajdują się wybrane diagramy: diagram klas, diagram aktywności.



Rysunek 3 - Diagram aktywności – Diagram przedstawia proces kalibracji oraz trwającą rozgrywkę.



Rysunek 4 - Diagram klas – na tym rysunku zamieszczony jest pierwszy fragment diagramu klas. Widoczna jest tu klasa Kalibracja wykorzystywane do ustawienia zakresu przechwytywanych kolorów dla pionów obu użytkowników, klasa Tools służącą do przetwarzania bitmapy oraz klasę Chessboard służącej do opisu szachownicy.



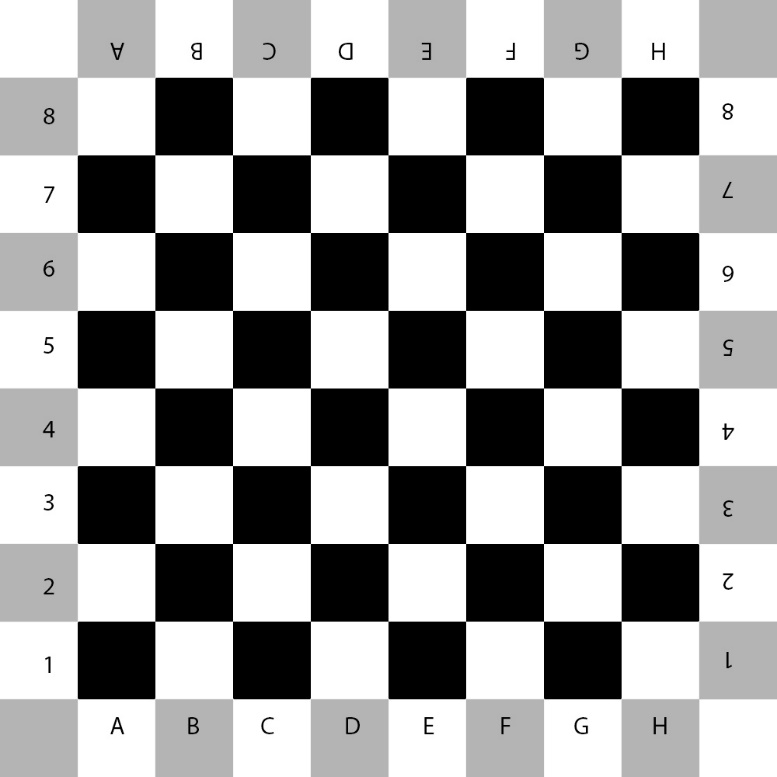
Rysunek 5 - Kontynuacja Rysunku 4 – na tym rysunku widoczna jest druga część diagramu klas. Znajduje się tutaj klasa Rozgrywka służąca do tworzenia wątku wykorzystywanego do pobierania obrazu z kamery, kalibracji aktualnego położenia pół oraz pionów graczy oraz wyświetlenia informacji o możliwych sekwencjach ruchowych. Klasa Rozgrywka odpowiada również za wizualizację możliwych ruchów które może wykonać gracz.

# Zasada działania aplikacji

1. Specjalnie przygotowana szachownica o wymiarach 10x10 (wymiary użytkowe 8x8) wykorzystywana przez metodę FindChessBoardCorners() pozwala na bezproblemowe znalezienie (punktów) narożników wszystkich pól użytkowych.
2. Na podstawie wektora punktów otrzymanego w poprzednim punkcie tworzone są instancje struktury Field reprezentującej pojedyncze pole szachownicy. Instancje te są dodawane do listy pól będącej składową klasy Chessboard.
3. Po dodaniu nowego pola do listy zliczana jest zawartość procentowa pikseli mieszczących się w zakresie kolorów reprezentujących pion danego gracza, które znajdują się między wierzchołkami dodanego do listy pola. Jeśli wartość ta przekracza 10% - do pola przypisywana jest informacja o stojącym na nim pionie danego gracza.

# Instrukcja użytkowania aplikacji

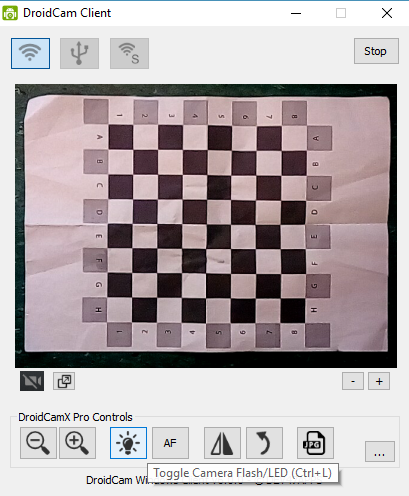
W celu poprawnego rozpoczęcia rozgrywki używając naszej aplikacji należy wykonać poniższe kroki:

1. Przygotować odpowiednio dostosowaną szachownicę o wymiarach 10x10 przedstawioną na rys. Dodatkowe pola są używane przez program jako markery by w łatwiejszy oraz szybszy sposób odnaleźć planszę.

*Rysunek 6 – Szachownica o wymiarach 10x10 jest specjalnie przygotowaną szachownicą z dodatkowymi polami. Pola te służą jako markery ułatwiające znalezienie planszy do gry przez aplikację znacząco przyspieszając działanie aplikacji.*

1. Podłączyć kamerę do komputera i umiejscowić ją nad polem rozgrywki. Najlepszą pozycją będzie ustawienie kamery nad środkowym polem tak by obiektyw kamery skierowany był na dół. Aplikacja wymaga, aby pole A1 znajdowało się w lewym górnym rogu.

Wskazówka: polecamy użycie aplikacji DroidCam umożliwiającej użycie kamery zamontowanej w smartfonie jako kamery zewnętrznej.



Rysunek 7 – Interfejs aplikacji DroidCam Client udostępniającej obraz z kamery smartfona. Na tym zrzucie ekranu możemy również zobaczyć odpowiednie ustawienie kamery oraz planszy do gry w warcaby.

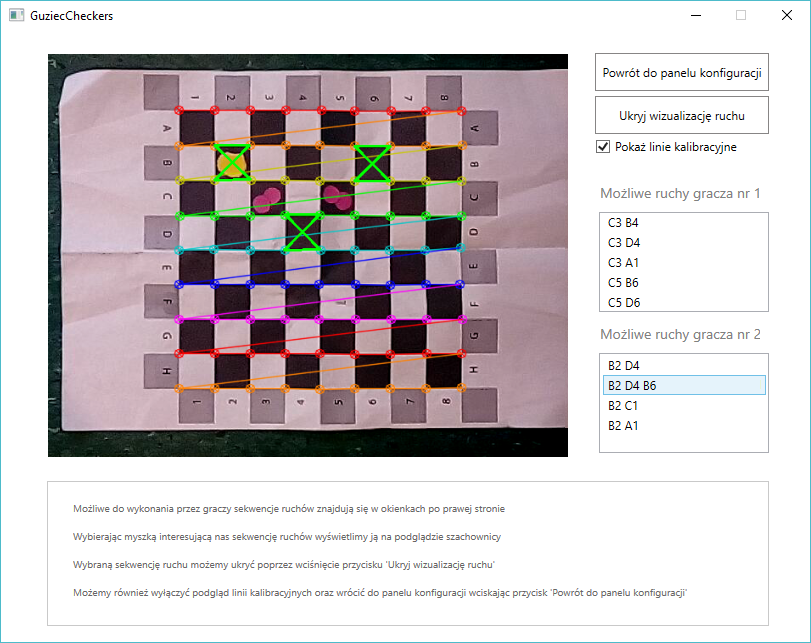
1. Uruchomić aplikację GuziecCheckers.
2. Po uruchomieniu aplikacji należy ją skalibrować postępując zgodnie z poniższymi krokami:

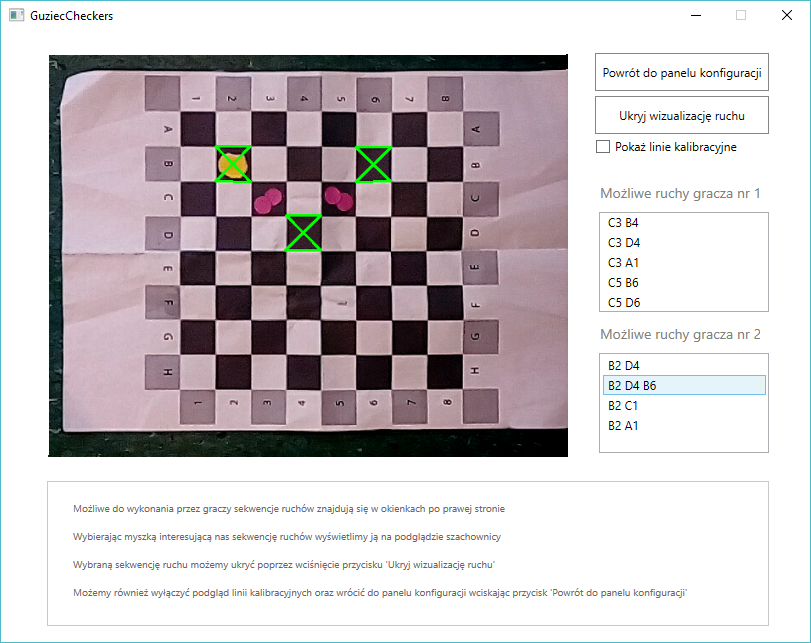
* Ustawić piony pierwszego gracza na pozycjach A1, B8, E5,G1,H8.
* Ustawić piony drugiego gracza na pozycjach A7, B2, D4, G7, H2.
* Manipulując wartościami liczbowymi odpowiadającymi składowym systemu barw RGB należy je zmienić tak, aby odpowiadały one kolorom pionów danego gracza.
* Zatwierdzić kalibrację poprzez naciśnięcie przycisku „Zapisz konfigurację”.

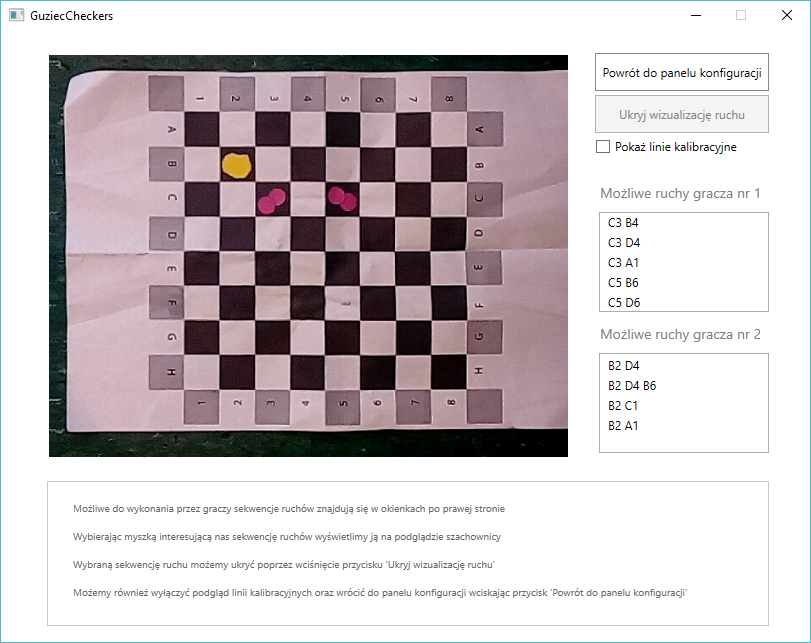
e) Rozpocząć rozgrywkę, którą można obserwować na ekranie komputera, dodatkowo można wyświetlać podpowiedzi prawidłowych ruchów.

# Pozostałe możliwości

Rozdział ten zawiera zrzuty ekranu ukazujące pozostałe możliwości napisanej przez nas aplikacji.

Rysunek 8 – Wizualizacja wybranej przez użytkownika możliwej do wykonania przez gracza nr 2 sekwencji ruchów z włączoną opcją pokazywania linii kalibracyjnych informujących o poprawnej detekcji pól szachownicy.

Rysunek 9 – Wizualizacja wybranej przez użytkownika możliwej do wykonania przez gracza nr 2 sekwencji ruchów z wyłączoną opcją pokazywania linii kalibracyjnych informujących o poprawnej detekcji pól szachownicy.

Rysunek 10 – Podgląd szachownicy z wyłączoną opcją pokazywania linii kalibracyjnych informujących o poprawnej detekcji pól szachownicy oraz bez wybranej celem wizualizacji sekwencji ruchów możliwych do wykonania przez gracza.

Możemy zauważyć, że mimo wyłączonej wizualizacji ruchu oraz wyłączonego podglądu linii kalibracyjnych, listy możliwych do wykonania sekwencji ruchów dla obu graczy posiadają wpisy co sugeruje prawidłowy przebieg kalibracji aplikacji (szachownica oraz znajdujące się na niej piony są rozpoznawane).

1. Plan rozwoju aplikacji

* Dalsza praca przy refaktoryzacji kodu.
* Poprawa wydajności aplikacji (szybkości przetwarzania pojedynczych klatek wideo).
* Poprawa interfejsu na bardziej „przyjazny” użytkownikom.
* Dodanie funkcjonalności zmiany podglądu z kamery na wizualizację 2D całej szachownicy i pionów znajdujących się w grze.
* Dodanie panelu rejestracji i logowania graczy.
* Dodanie statystyk gracza zbieranych podczas rozgrywki.
* Dodanie funkcjonalności włączania odmierzania czasu na wykonanie ruchu przez gracza.

# Zagadnienia związane z implementacją

W tym rozdziale przedstawione zostaną w skrócie klasy, ich metody i atrybuty oraz struktury wykorzystane w implementacji naszego projektu.

1. Klasa Tools – Pomocnicza klasa aplikacji wykorzystywana w miejscach wymagających przetwarzania Bitmap.
2. metody:

* ImageToBitmapSource – Przyjmująca jako parametr obiekt typu   
  Image<Bgr, byte> a zwracająca typ BitmapSource. Wykorzystywana w celu wyświetlenia przetworzonego przez EmguCV obrazu za pomocą kontrolki Image.
* BitmapToBitmapImage – Przyjmująca jako parametr obiekt typu Bitmap   
  a zwracająca obiekt typu BitmapImage.
* ImageControlToBitmap – Przyjmująca jako parametr kontrolkę typu Image a zwracająca wyświetlany aktualnie przez nią obraz w postaci obiektu typu Bitmap.

1. Klasa Chessboard – Główna klasa aplikacji odwzorowująca stan rzeczywistej szachownicy.
2. metody:
   * prywatne:
     + FindMoves – Metoda wewnętrzna wykorzystująca rekurencję w celu znalezienia wszystkich możliwych do wykonania sekwencji ruchowych.
   * Publiczne:
     + FindMoves – Metoda przeciążająca inną o tej samej nazwie metodę prywatną. Jako parametr przyjmuje liczbę z zakresu 1-2 reprezentującą numer gracza. Jej zadaniem jest przede wszystkim wywołanie jej rekurencyjnego odpowiednika i zwrócenie otrzymanych przez niego rezultatów w postaci znalezionych dla gracza o numerze równym zadanemu parametrowi możliwych do wykonania sekwencji ruchów pomijając ruchy w tył, jeśli nie są to bicia.
     + Calibration – Metoda przyjmująca jako parametry aktualnie pobrany   
       w procesie przetwarzania obraz z kamery oraz zmienną boolowską, której wartość ma wpływ na to czy wyświetlone zostaną linie kalibracyjne czy obraz z kamery ich pozbawiony. Zadaniem metody jest znalezienie aktualnego położenia wszystkich pól szachownicy   
       w rozpatrywanym obrazie oraz informacji o pionach, które się na nich znajdują. W przypadku niepowodzenia operacji zwracana jest przez metodę wartość false.
3. atrybuty:
   * prywatne:
     + \_size (int) – Liczba reprezentująca długość boku wykorzystywanej przez nas szachownicy. Domyślna wartość dla przygotowanej przez nas szachownicy to 10.
     + \_moves (List<string>) - Lista zawierająca ciągi znaków reprezentujące możliwe do wykonania sekwencje ruchów gracza, dla którego została wykonana metoda FindMoves. Lista wykorzystywana przez tę metodę   
       w celu zwrócenia wyniku.
   * publiczne:
     + \_fields (List<Field>) - Lista zawierająca znalezione po wykonaniu metody Calibration pola szachownicy. Lista pól wykorzystywana jest   
       w celu wyznaczenia możliwych do wykonania przez graczy sekwencji ruchów zwracanych przez metodę FindMoves.
     + kamera (Capture) – Statyczna składowa reprezentująca obiekt kamery skierowanej obiektywem na szachownicę. Wykorzystywana zarówno   
       w procesie kalibracji i procesie rozgrywki w celu pobierania klatek obrazu do przetwarzania w czasie rzeczywistym.
4. Struktura Field – Struktura opisująca pojedyncze pole szachownicy.
   1. atrybuty:

* column (char) – Oznaczenie kolumny, w której znajduje się pole.
* row (int) – Oznaczenie rzędu, w którym znajduje się pole.
* ownership (int) – Oznaczenie pionu znajdującego się na polu. 0 – Oznacza brak pionu, 1 – pion gracza nr „1”, 2 – pion gracza nr „2”.
* leftUp, leftDown, rightUp, rightDown (Point) – Współrzędne punktów na obrazie z kamery reprezentujących kolejno lewy-górny wierzchołek pola, lewy-dolny wierzchołek pola, prawy-górny wierzchołek pola oraz prawy-dolny wierzchołek pola.

1. Struktura PawnsInfo – Struktura przechowująca w swoich składowych informacje o zakresach kolorów pionów graczy ustalanych w panelu konfiguracji.
   1. atrybuty:

* minColorRange1 (static Bgr) – Dolny zakres kolorystyczny składowych RGB reprezentujących piony gracza nr 1.
* maxColorRange1 (static Bgr) – Górny zakres kolorystyczny składowych RGB reprezentujących piony gracza nr 1.
* minColorRange2 (static Bgr) – Dolny zakres kolorystyczny składowych RGB reprezentujących piony gracza nr 2.
* maxColorRange2 (static Bgr) – Górny zakres kolorystyczny składowych RGB reprezentujących piony gracza nr 2.

1. Struktura Move – Struktura reprezentująca elementarną część sekwencji ruchu   
   w formie następnego pola.
   1. atrybuty:
      * column (char) – Oznaczenie kolumny, w której znajduje się pole.
      * row (int) – Oznaczenie rzędu, w którym znajduje się pole.