**POLITECHNIKA POZNAŃSKA**

WYDZIAŁ INFORMATYKI I TELEKOMUNIKACJI

Instytut Informatyki

**Dominik Hażak**

**Paulina Kukuła**

**Jakub Małecki**

**Dokumentacja z zajęć projektowych**

# Rozpoznawanie obrazu z gry Lau Kata Kati oraz wizualizacja stanu gry na komputerze

9 września 2020

**Spis treści:**

## 1. Opis aplikacji

Przygotowana aplikacja umożliwia rozpoznawanie meczu gry dwuosobowej z pełną informacją (np. warcaby) rozgrywanej na rzeczywistej planszy do gry. W szczególności aplikacja ta została przygotowana do rozpoznawania partii gry w Lau Kata Kati. Rozgrywka jest przechwytywana przez kamerę skierowaną na planszę z góry. Program rozpoznaje pola planszy i pionki o zdefiniowanych wcześniej kolorach oraz analizuje ich układ i sprawdza, czy gra przeprowadzana jest zgodnie z zasadami. Ponadto aplikacja wizualizuje aktualny stan rozgrywki w oknie programu na wirtualnej planszy.

Temat ten został wybrany, ponieważ dzięki niemu członkowie grupy mogli rozszerzyć projekt przygotowywany na inne zajęcia (Sztuczna Inteligencja). Podczas pracy nad tamtym projektem również skupiono się na grze Lau Kata Kati, tworząc maszynę grającą.

## 2. Podział prac

W tabeli (tabela 1) znajduje się opis prac, jakie wykonywała każda z pracujących nad projektem osób. W niektórych sytuacjach (np. jeśli jedna z osób napotkała pewne trudności) nad danym problemem pracowali wszyscy członkowie grupy, pomagając sobie wzajemnie.

| Imię i nazwisko | Wykonane zadania |
|---|---|
| Dominik Hażak | ● Pobieranie obrazu z kamery<br>● Detekcja elementów gry na obrazie<br>● Przetworzenie danych z detekcji na format zrozumiały dla programu<br>● Tworzenie dokumentacji |
| Paulina Kukuła | ● Implementowanie zasad gry (wbudowanie w program zasad, zgodnie z którymi sprawdzana będzie poprawność ruchów)<br>● Sprawdzanie, czy akcja (zmiana na planszy) odpowiada zasadom<br>● Tworzenie dokumentacji |
| Jakub Małecki | ● Interfejs graficzny programu<br>● Wyświetlanie komunikatów dotyczących gry i działania systemu<br>● Wizualizacja stanu gry (przedstawienie na ekranie rzeczywistego położenia pionków, z uwzględnieniem prawidłowości ruchu)<br>● Tworzenie dokumentacji |

Tabela 1: Podział prac

## 3. Funkcjonalność aplikacji

- **Wczytywanie obrazu z kamery lub pliku** – program wczytuje grafikę z podanego źródła za pomocą biblioteki graficznej, po czym przetwarza ją w odpowiedni sposób, tj. skaluje oraz obraca w razie potrzeby. Obrócenie zdjęcia pozwala na poprawne rozpoznanie planszy, a skalowanie zapewnia stały rozmiar elementów na zdjęciu, dzięki czemu nie jest wymagane dostosowywanie elementów algorytmu rozpoznającego do zmiennych rozmiarów obrazów wejściowych.

- **Rozpoznawanie planszy gry Lau Kata Kati** – gra Lau Kata Kati ma charakterystyczną planszę, która jest kodowana przez program w postaci macierzy liczb całkowitych. Rozpoznanie układu pionków na planszy polega na określeniu miejsc na zdjęciu, które zawierają koła, ponieważ one są polami planszy.

- **Rozpoznanie zajętości pola planszy** – po znalezieniu odpowiedniej ilości pól na planszy rozpoczyna się określenie stanu każdego pola. Stan jest określany na podstawie koloru zawarty wewnątrz koła. Program ma ustalone wartości graniczne w palecie HSV dla koloru pionków odpowiedniego gracza. Zajęte pole reprezentowane jest przez odpowiednią liczbę identyfikującą gracza. Pole puste z założenia jest białe. Wyjątkiem są pola w 4 rzędzie, ponieważ w odróżnieniu od pozostałych 4 rząd zawiera tylko jedno pole. W takim wypadku pole lewe i prawe w 4 rzędzie są reprezentowane przez liczby -1.

- **Detekcja ruchu** – program otrzymuje regularnie macierze liczb naturalnych reprezentujące rzeczywistą planszę gry w danym czasie. Cyklicznie wykonywane jest sprawdzenie czy nowo otrzymana macierz różni się od zapisanej wcześniej. Jeśli różnice zostaną wykryte, program rozpoczyna detekcję wykrytego ruchu starając się znaleźć z którego pola na które

przesunięty został pionek gracza. Rozpoznawany jest też rodzaj ruchu tj. standardowy ruchu lub bicie.

- **Walidacja wykonanego ruchu** – rozpoznany ruch jest przekazywany do silnika gry, który sprawdza czy ruch jest prawidłowy. Silnik rozgrywa grę wirtualną, dzięki czemu ma możliwość sprawdzenia stanu rozgrywki w każdym momencie. Ma on zaimplementowane reguły gry i na ich podstawie podejmowana jest decyzja o wykonaniu lub niewykonaniu ruchu.

- **Graficzna reprezentacja rozgrywki** – stan planszy zawartej w silniku gry przedstawiany jest graficznie w oknie programu. Okno zawiera planszę widzianą z góry, której pola są kolorowane odpowiednio w zależności od ich stanu. Obok wirtualnej planszy znajduje się pole tekstowe, które wyświetla komunikaty o prawidłowości wykonywanych ruchów.

## 4. Wykorzystane technologie

Podczas pracy korzystaliśmy z kilku głównych technologii:

- **Java 11** – obiektowy język programowania ogólnego zastosowania. Jest współbieżny i oparty na klasach, cechuje się silnym typowaniem, służy do tworzenia programów źródłowych kompilowanych do kodu bajtowego. Zaletą Javy jest jej niezależność od architektury, system wyjątków, a także sieciowość oraz możliwość obsługi programowania rozproszonego.

- **JavaFX 11.0.2** – platforma służąca do tworzenia aplikacji desktopowych a także aplikacji internetowych oferujących bogaty, dynamiczny, jednoekranowy interfejs (RIA, ang. *Rich Internet Application*). W przyszłości JavaFX ma zastąpić *Swing* jako standardową bibliotekę GUI (ang. *Graphical User*

*Interface*) dla platformy Java. W projekcie platforma ta wykorzystana została do wyświetlenia stanu rozgrywki na ekranie.

- **Kotlin** – statycznie typowany język programowania, który działa na maszynie wirtualnej Javy. Kotlin został użyty w części projektu w celu uproszczenia kodu programu, ponieważ Kotlin jest językiem zawierającym nowoczesne rozwiązania i łatwiejszą składnię bez strat w prędkości kompilacji i działania programu.

- **OpenCV 4.3.0** – wieloplatformowa biblioteka wykorzystywana do przetwarzania obrazu w czasie rzeczywistym. Biblioteka napisana jest w języku C++, który zapewnia niskopoziomową kontrolę nad danymi graficznymi. W programie biblioteka jest wykorzystana do odczytywania obrazu z wybranego źródła i rozpoznanie odpowiednich elementów planszy tj. pól i pionków. Jest ona w pełni darmowa i oparta na licencji BSD, która umożliwia umieszczenie biblioteki w projekcie.

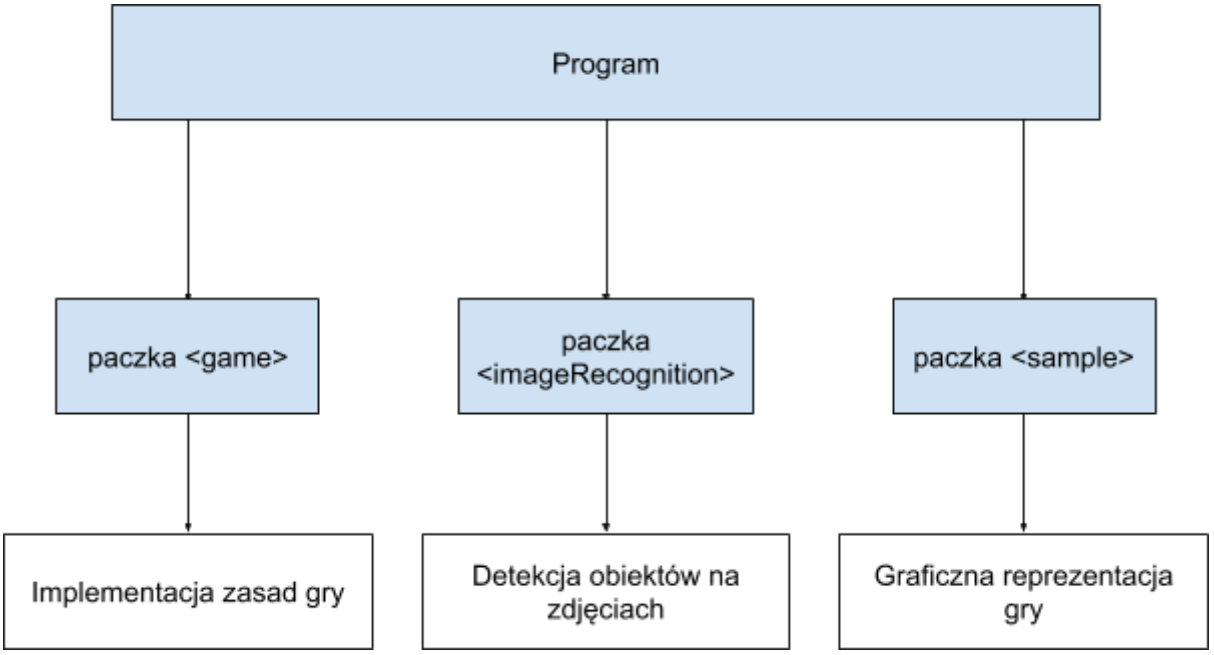
Ponadto podczas pracy nad projektem wykorzystane zostały następujące narzędzia:

- **IntelliJ IDEA** – stworzone przez firmę JetBrains IDE (zintegrowane środowisko programistyczne) dla języka Java. Zapewnia między innymi takie funkcje, jak uzupełnianie kodu (poprzez analizę kontekstu), łatwą nawigację po kodzie, refaktoryzację kodu oraz jego debugowanie.

- **GitHub** – serwis internetowy wykorzystujący system kontroli wersji Git, przeznaczony głównie dla projektów programistycznych. Udostępnia on darmowy hosting programów *open source* i prywatnych repozytoriów. Github posiada wiele opcji ułatwiających współpracę w projekcie, z których jedną na najważniejszych jest rozdzielenie projektu na poszczególne gałęzie (ang. *branches*), aby współtwórcy nie przeszkadzali sobie wzajemnie przy swoich częściach projektu.

## 5. Architektura rozwiązania

Cała funkcjonalność programu jest zawarta w jednym pliku wykonywalnym. Dzięki takiemu rozwiązaniu program można uruchomić nawet w specjalnie zaprojektowanych systemach, ponieważ nie zależy on od usług zewnętrznych. Program jest wewnętrznie podzielony zgodnie z założeniami programowania obiektowego i paczkami języka Java (schemat 1). Każda główna funkcjonalność jest zawarta w osobnej paczce i składa się z kilku klas wykonujących konkretne zadania danej funkcjonalności.




*Schemat 1 - Struktura programu z podziałem na paczki*

- **Paczka game** – paczka napisana w języku Kotlin zawierająca implementację zasad gry Lau Kata Kati oraz silnik gry. Klasy zawarte:
    - **Klasa Game** – klasa ta implementuje interfejs Runnable i służy do uruchamiania wątku z silnikiem gry,

    - **Klasa GameCore** – klasa ta zawiera implementację zasad gry i algorytmów wymaganych przez silnik gry do poprawnego działania i przeprowadzania rozgrywki,
    - **Klasa Player** – klasa ta reprezentuje gracza przechowując informację o jego numerze oraz typie

- **Paczka imageRecognition** – paczka napisana w języku Kotlin wykorzystująca bibliotekę OpenCV. Zadaniem klas tej paczki jest wczytanie i przetworzenie obrazu wejściowego i wyeksportowanie macierzy liczb całkowitych. Paczka zawiera klasy:
    - **Klasa ImageGrabber** – klasa za pomocą biblioteki opencv wczytuje obraz z podanego źródła, przetwarza go i przekazuje do dekodera w celu znalezienia i odkodowania pól planszy. Dodatkowo klasa implementuje interfejs Runnable, dzięki czemu może cyklicznie pobierać kolejne obrazy wejściowe.
    - **Klasa Decoder** – klasa przyjmuje przetworzony obraz w celu rozpoznania pól planszy za pomocą algorytmu HoughCircles, który jest zawarty w bibliotece OpenCV. Wynikiem działania tej klasy jest macierz liczb całkowitych reprezentująca zakodowany stan planszy z obrazu.
    - **Klasa FileNamesComparator** – jest to klasa uzupełniająca, która implementuje interfejs Comparator. Klasa ta powstała w drodze potrzeby sortowania sekwencji ruchów zapisanych jako zdjęcia, ponieważ standardowe sortowanie powodowało wczytywanie ruchu numer 11 przed ruchem numer 2.

- **Paczka sample** – paczka napisana w języku Java wykorzystująca platformę JavaFX. Zadaniem klas tej paczki jest uruchomienie aplikacji, a także graficzne wyświetlenie stanu gry oraz informacji związanych z ruchami
    - **Klasa Main** – klasa główna, służy do uruchomienia projektu. W klasie tej zawarte są również najważniejsze informacje dotyczące okna aplikacji, takie jak nazwa okna czy jego wymiary. Uruchamiany jest

również wątek *grabberThread* służący do uruchomienia klasy *ImageGrabber*.

- **Klasa Controller** – klasa służąca do modyfikowania zawartości okna aplikacji. Znajdują się w niej metody:
  - *fillBoard* – odpowiada za wypełnienie planszy w aplikacji odpowiednimi kolorami pionków,
  - *wyswietlWiadomosc* – wypisuje informacje dotyczące ruchu w oknie tekstowym znajdującym się w aplikacji.
- **Klasa Rules** – klasa odpowiadająca za zasady gry związane z ruchem pionka – podstawowy ruch oraz bicia, a także za porównanie stanu planszy znajdującej się w pobieranym obrazie, ze stanem planszy znajdującej się w aplikacji. Klasa implementuje interfejs Runnable, dzięki czemu może cyklicznie pobierać obrazy do porównania. Znajdują się w niej metody:
  - *checkDifferences* – sprawdzająca różnice pomiędzy stanem planszy pobranej z zewnątrz, ze stanem planszy w aplikacji
  - *checkMove* – sprawdzająca, czy zmiana na planszy była spowodowana podstawowym ruchem
  - *checkBicie* – sprawdzająca, czy zmiana na planszy była spowodowana biciem pionka przeciwnika
  - *checkLength* – sprawdzająca, czy w trakcie ruchu pionek faktycznie przesunął się o odpowiednią liczbę pól
  - *getFieldNumber* – pobierająca numer pola, na którym wykonana została akcja
  - *run* – wywołująca wcześniej wymienione metody
- **sample.fxml** – plik definiujący graficzny interfejs, oparty na języku XML. Zdefiniowane są w nim obiekty znajdujące się wewnątrz okna aplikacji, takie jak okręgi (*Circle*) będące polami, po których przemieszczają się pionki, czy pole tekstowe (*TextArea*), w którym wyświetlane są komunikaty.

## 6. Interesujące problemy i ich rozwiązania

- **Problem z niską jakością obrazu pochodzącego z kamery** – w trakcie pracy nad projektem okazało się, że obraz pochodzący z kamery internetowej nie jest wystarczająco dobrej jakości, żeby możliwe było poprawne rozpoznanie kół za pomocą algorytmu Hough Circles. Rozwiązaniem tego problemu jest przygotowanie sekwencji ruchów w postaci obrazów zachowanych na dysku komputera. Program zamiast pobierać klatkę z kamery, wczytuje kolejne zdjęcie, które przygotowano w rozdzielczości ułatwiającej działanie biblioteki graficznej.

- **Problem z niedeterministycznym działaniem programu** – w trakcie testów programu okazało się, że nie za każdym razem program działa w taki sam sposób. Zaobserwowano, że są dwa sposoby działania programu, z czego tylko jeden poprawny dla zestawu testowego. Powodem takiego stanu rzeczy okazało się być losowanie gracza początkowego przez silnik gry. Gra wczytywała testowy zestaw ruchów, który zawsze zaczynał od tego samego gracza, ale silnik gry w sposób losowy z szansą 50% wybierał gracza rozpoczynającego grę.

- **Problem z biciem dwóch pionków podczas ruchu jednego gracza** – podczas przechwytywania informacji na temat obecnego stanu planszy, system nie wykrywał informacji, że pionek, który w trakcie obecnego ruchu wykonał bicie, miał możliwość zbicia co najmniej jednego pionka więcej - system automatycznie oddawał ruch drugiemu graczowi. Okazało się, że problem stanowiła jedna zakomentowana linijka kodu, przez którą nie dochodziło do ponownego sprawdzenia możliwości bicia.

- **Problem związany z wypisywaniem w interfejsie graficznym informacji o błędnym ruchu** – system informuje gracza o wykonaniu błędnego ruchu poprzez wypisanie w polu znajdującym się po prawej stronie planszy

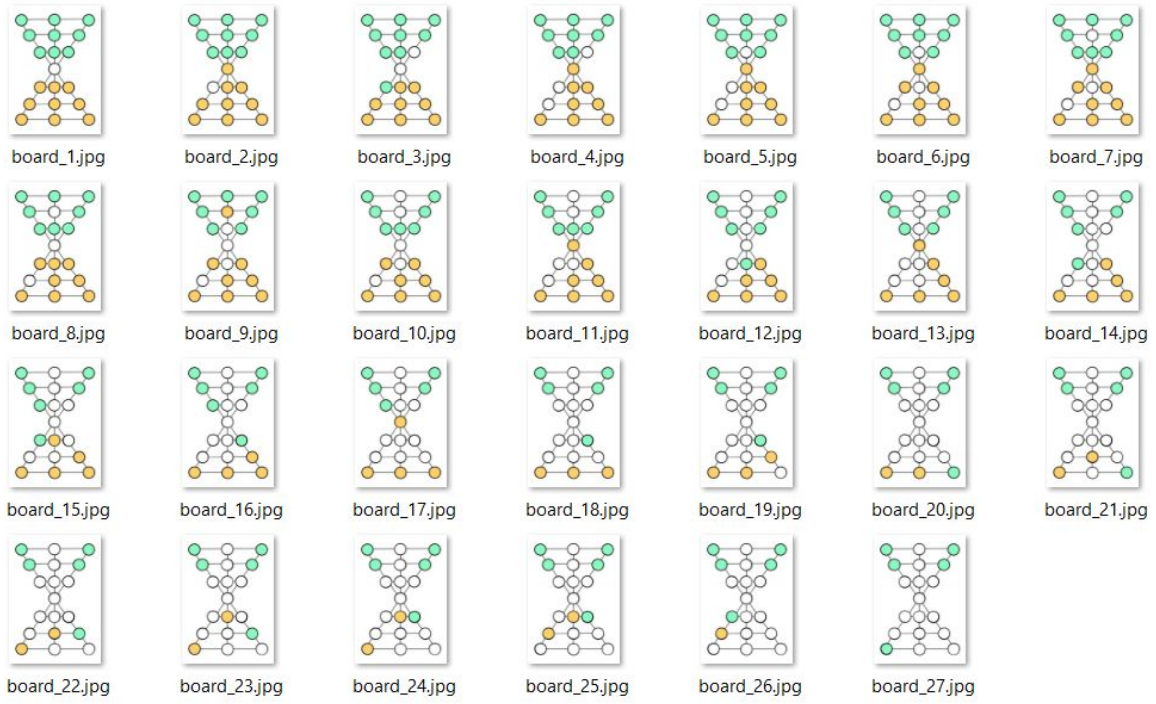
wiadomości “Błędny ruch” bądź “Błędny ruch. Należy wykonać bicie”. Okazało się, że przy pobraniu danych z obrazu, informacja ta wypisywana była trzykrotnie. Problem ten udało się rozwiązać dzięki zastosowaniu “flagi” - zmiennej typu *boolean*, która sprawdza, czy dla danego obrazu wypisany został już błąd.

- **Problem z prawidłową kolejnością pobierania obrazów z folderu** – w początkowej wersji aplikacji pobierającej obrazy z folderu, system zamiast pobierać je w kolejności *board_1 -> board_2 -> board_3 -> … -> board_n*, pobierał je w kolejności *board_1 -> board_10 -> board_11 -> … -> board_19 -> board_2 -> …* . Aby naprawić ten problem, potrzebne było utworzenie osobnej klasy *FilesComparator*, w której numer podany na końcu nazwy pliku był “wyciągany” i przekazywany do klasy *ImageGrabber*, w której obrazy były dalej przetwarzane już w prawidłowej kolejności.

## 7. Instrukcja użytkowania aplikacji

Przykładowa rozgrywka zaprezentowana została na nagraniu dostępnym pod linkiem: https://www.youtube.com/watch?v=GiLFW51EC-Q.

W folderze *test* (rys.1 ) znajdują się obrazy o nazwie *board_1*, *board_2*, …, *board_27* służące jako dane wejściowe. Użytkownik, w celu sprawdzenia poprawności działania programu, może modyfikować je w dowolny sposób, aby otrzymać oczekiwany wynik.




*Rysunek 1 - zawartość folderu "test"*

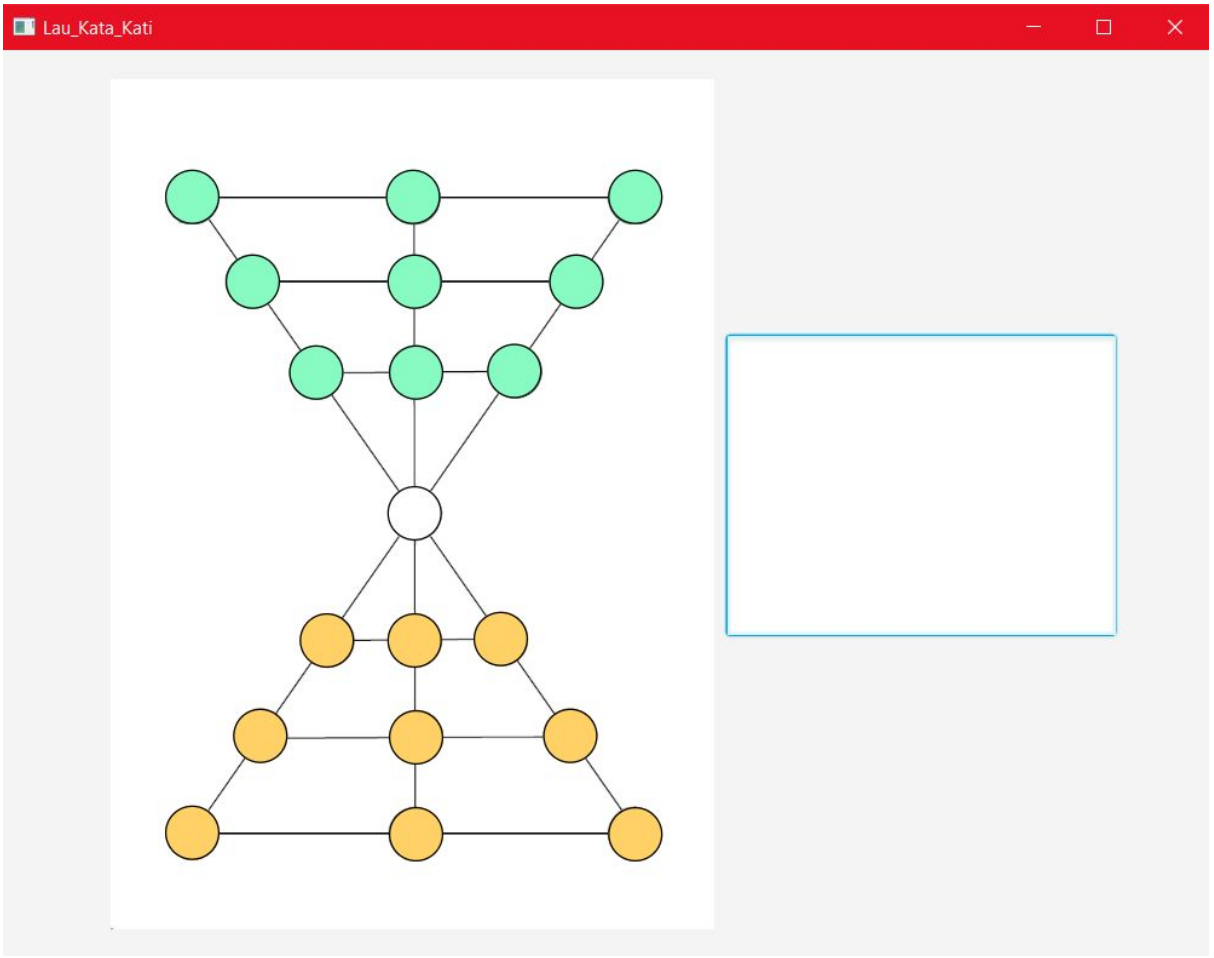
W celu prawidłowego uruchomienia programu, należy pobrać i zaimportować następujące biblioteki:

- **JavaFX SDK** w wersji 11.0.2 (do pobrania na stronie https://gluonhq.com/products/javafx/ ),
- **OpenCV** w wersji 4.3.0 (do pobrania na stronie https://opencv.org/opencv-4-3-0/ )

Oprócz tego, należy skonfigurować *runtime* dla języka Kotlin (środowisko IntelliJ IDEA proponuje tę opcję automatycznie), a także w edytorze konfiguracji projektu, w polu *VM Options* umieścić następujące komendy:

--module-path <ścieżka do zainstalowanego interfejsu JavaFX>/javafx-sdk-11.0.2/lib
--add-modules=javafx.controls,javafx.fxml
-Djava.library.path="<ścieżka do projektu>/bin"

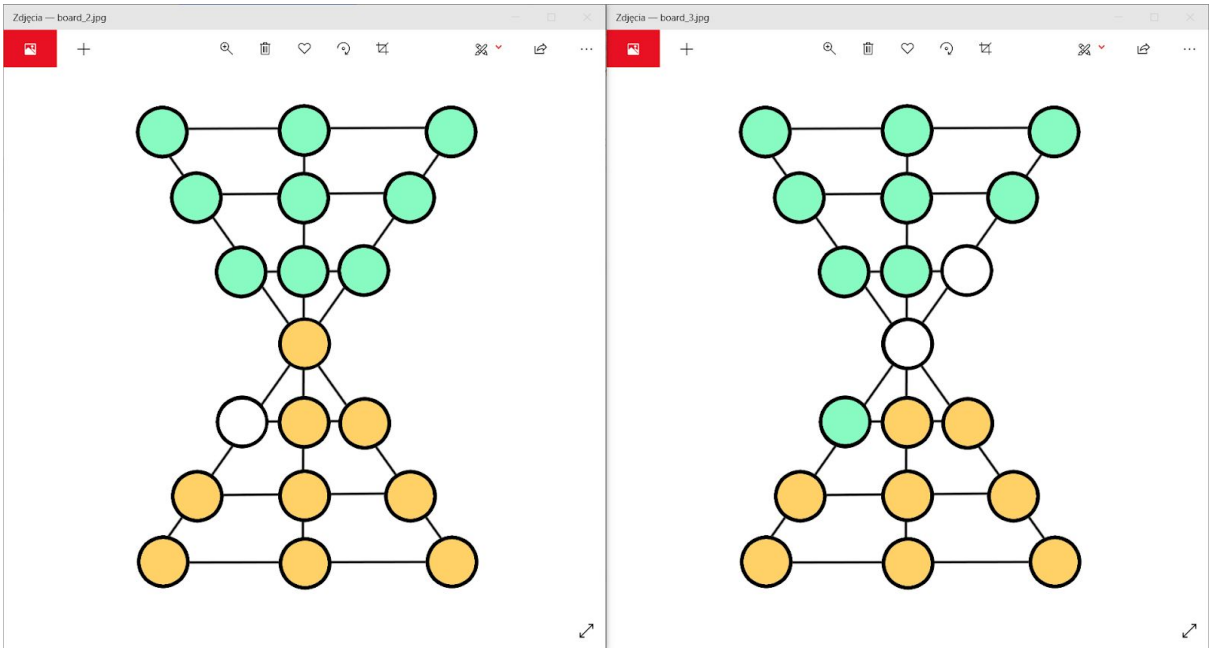
Po uruchomieniu projektu, na ekranie pojawi się okno z planszą gry (rys. 2), oraz z oknem tekstowym, w którym wyświetlane są informacje o wykonanym ruchu (lub o tym, że wykonany został błędny ruch).



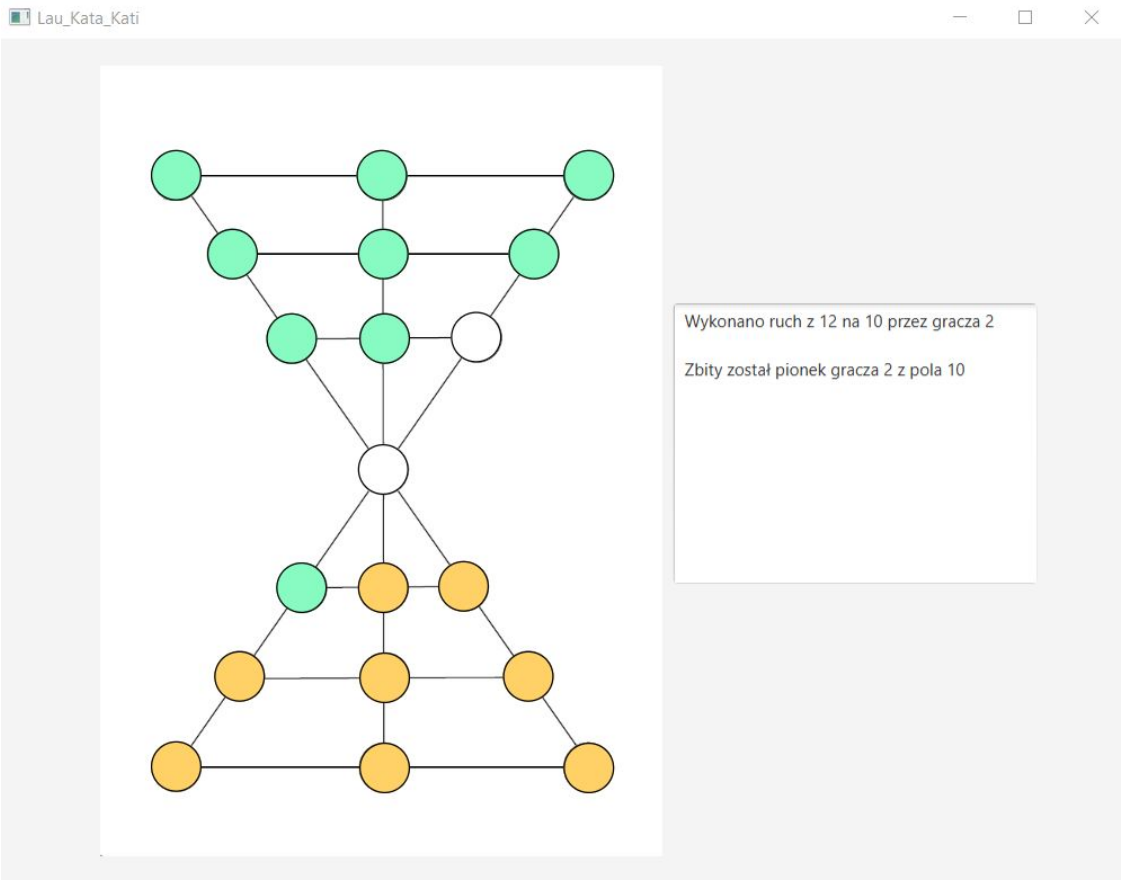

*Rysunek 2 - okno z planszą oraz polem tekstowym*

Następnie program pobiera kolejne obrazy z folderu i porównuje je ze stanem planszy, po czym podejmowana jest decyzja, czy ruch został wykonany prawidłowo i można zaktualizować stan planszy w programie, czy wystąpił błędny ruch i plansza nie zostanie zaktualizowana.

Na poniższym przykładzie (rys. 3) pobrany jest obraz *board_3*, na którym pionek gracza 1 (pierwotnie znajdującego się na polu o indeksie 8) dokonuje bicia pionka gracza 2 (znajdującego się na polu o indeksie 10, co prezentuje obraz *board_2*). Program sprawdza poprawność wykonanego ruchu - ruch jest prawidłowy, więc plansza jest aktualizowana – pionek gracza 1 przemieszcza się z pola 8 na pole 12, natomiast pionek gracza 2 jest usuwany z planszy (rys. 4).

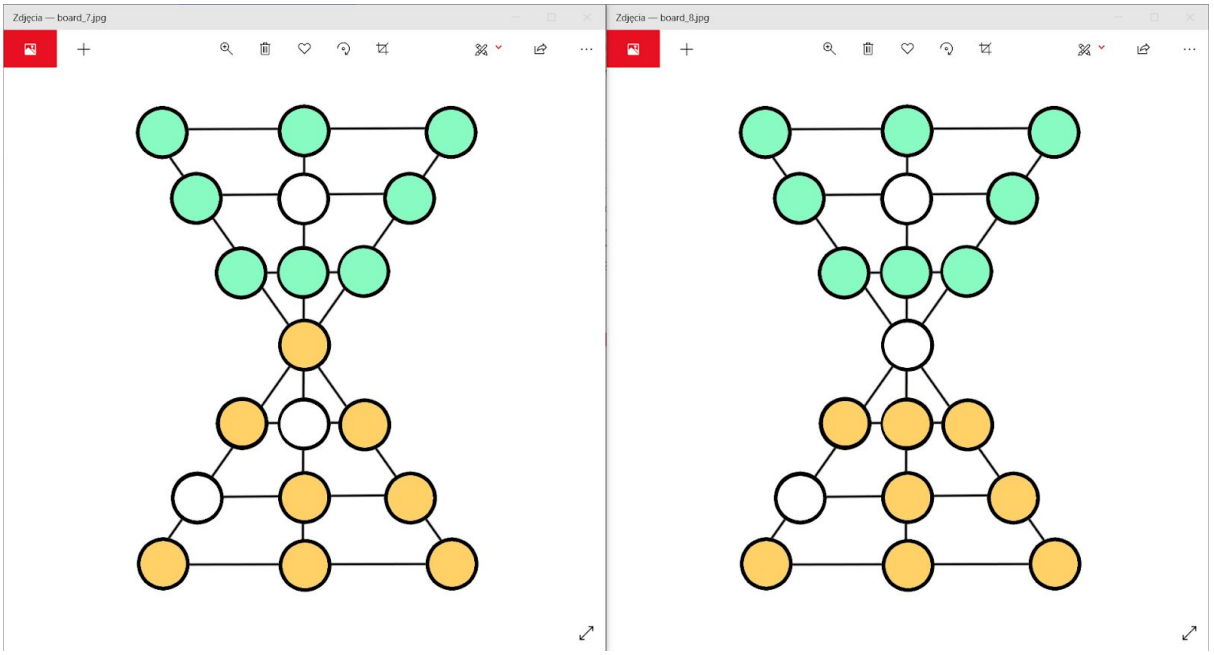



*Rysunek 3 - porównanie obrazów board_2 i board_3*

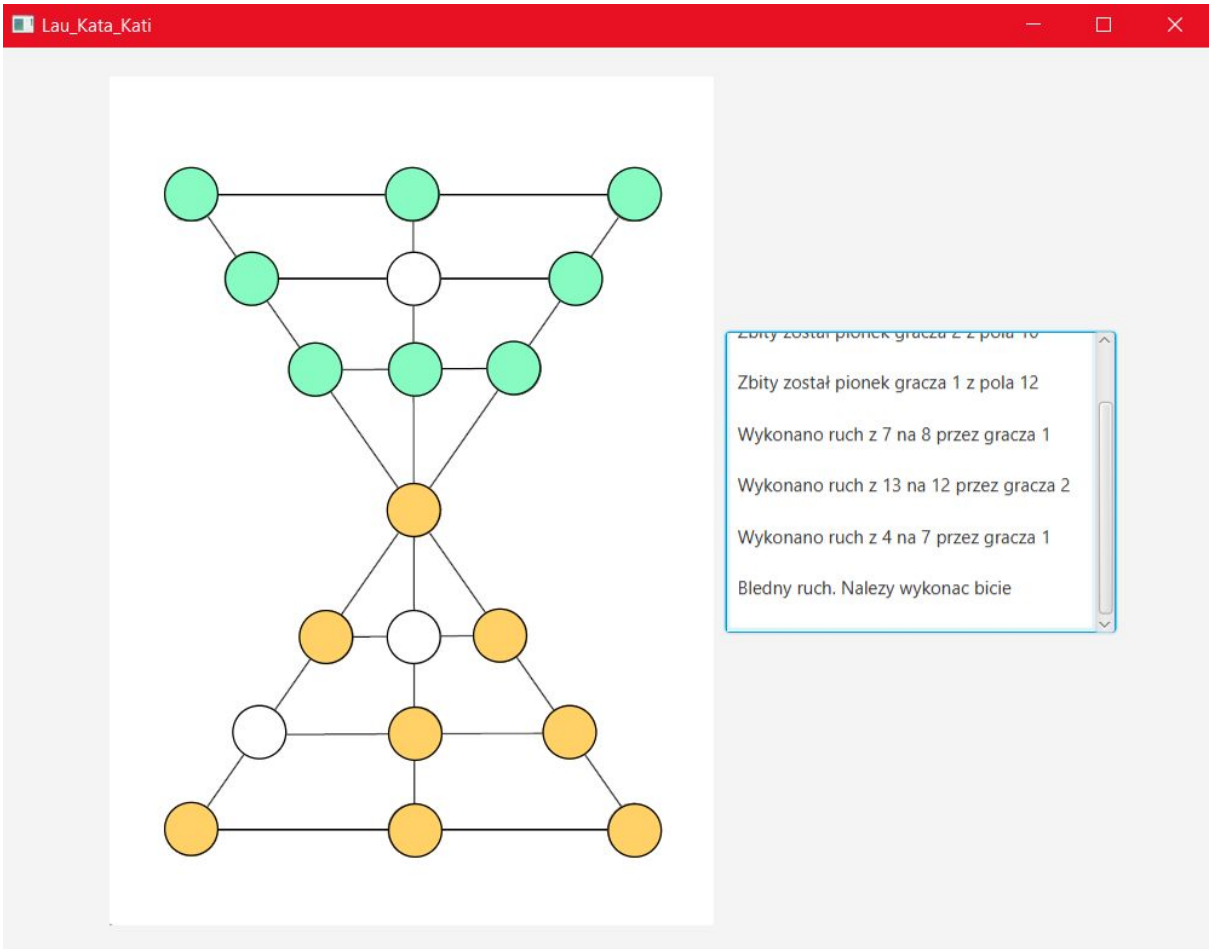



*Rysunek 4 - okno z przetworzonym ruchem z obrazu board_3*

Następny przykład przedstawia sytuację, w której jest wykonany błędny ruch – jak można zauważyć na obrazku *board_7* (rys. 5), pionek gracza 2 (pomarańczowy) powinien zbić pionek gracza 1, ponieważ bicie jest obowiązkowe. Jednak na obrazku *board_8* widać, że pionek gracza 2 przemieścił się w drugą stronę (rys. 5). Reguły gry zinterpretowały ten ruch jako błędny, po czym wyświetlony został na ekranie komunikat “*Bledny ruch. Nalezy wykonac bicie”* , a plansza w grze nie została zmodyfikowana (rys. 6).

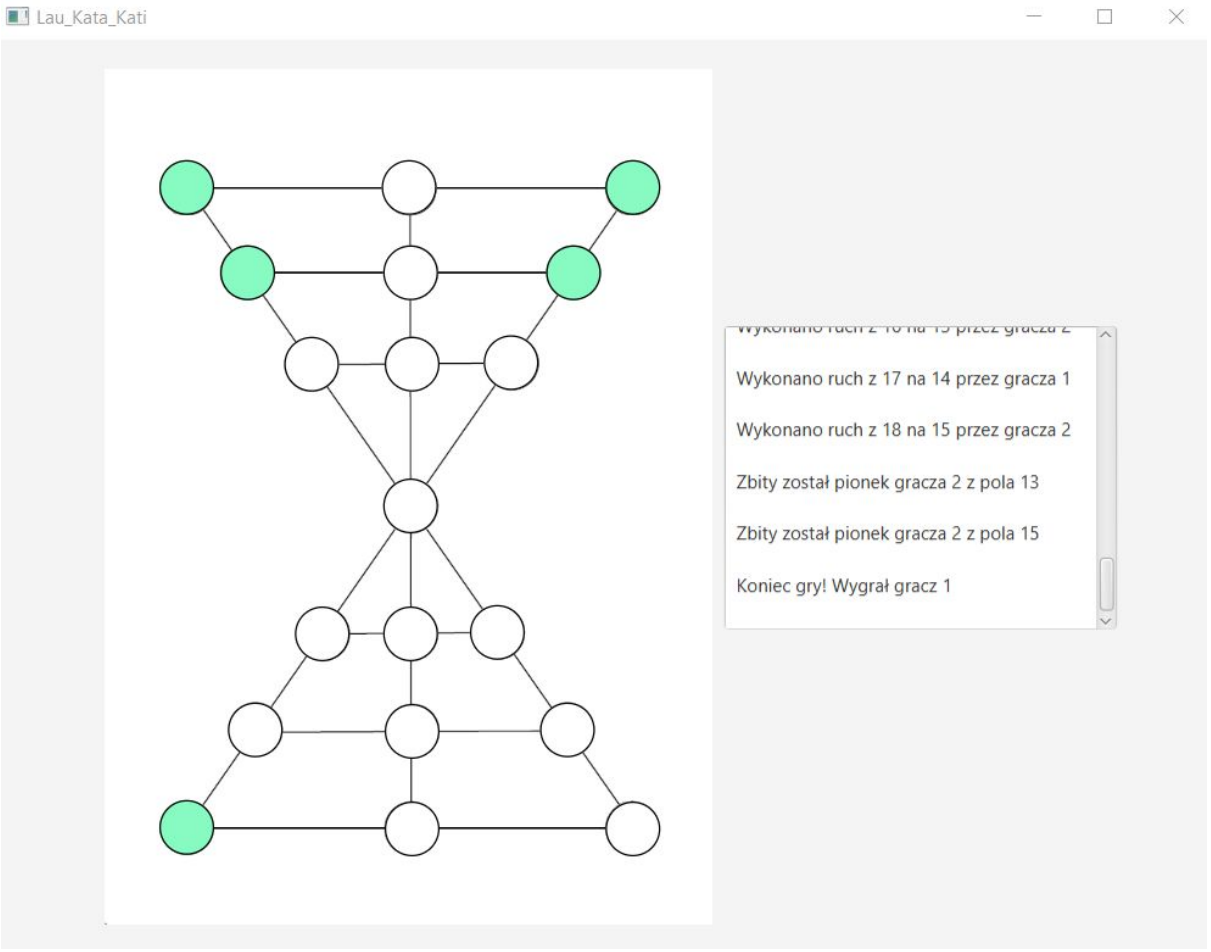



*Rysunek 5 - porównanie obrazów board_7 oraz board_8*




*Rysunek 6 - okno z informacją o błędnym ruchu i brak aktualizacji stanu gry*

Gra toczy się do czasu, aż na planszy nie pozostanie żaden z pionków jednego z graczy – w chwili zakończenia rozgrywki w polu tekstowym pojawia się odpowiedni komunikat, informujący o zwycięzcy (rys. 7).




*Rysunek 7 - koniec gry*