# Projekt Programowanie w języku Java

Wydział Elektrotechniki Automatyki i Informatyki

Politechnika Świętokrzyska

| | |
|---|---|
| Studia: **Stacjonarne I stopnia** | Kierunek: **Informatyka** |
| Data oddania: **26.06.2017** | Grupa: **2ID15B** |
| Ocena: | 1. Bartłomiej Osak<br>2. Tomasz Pasternak |

Temat projektu:

## Warcaby

# 1. TEMAT PROJEKTU

Tematem naszego projektu jest jedna z najpopularniejszych gier planszowych – Warcaby. Gra posiada wiele odmian, spośród których uważane za dyscyplinę sportową są jedynie warcaby polskie (stupolowe) nazywane też międzynarodowymi. W większości krajów gra odbywa się jednak przeważnie na warcabnicy posiadającej 64 pola (często zastępowanej szachownicą), z wykorzystaniem 24 pionków (po 12 dla każdego z graczy). W naszym przypadku zrealizowaliśmy rozgrywkę na planszy o rozmiarze 8 x 8 pól. Rozgrywka odbywa się na ciemnych polach, a plansza posiada pola kolorowane na przemian na kolor jasny oraz ciemny.

Głównym założeniem naszego projektu było napisanie klasycznej gry Warcaby z opcjami gry oraz rozgrywką z drugim graczem poprzez połączenie sieciowe lub lokalnie na jednym komputerze. Cele zostały w pełni zrealizowane. Ponadto w projekcie uwzględniliśmy większość algorytmów ruchu oraz bić pionków, również tworzenie się damek.

# 2. REALIZACJA PROJEKTU

## a) Język programowania, biblioteki, środowisko:

Nasz projekt został napisany w całości w języku Java zgodnie z paradygmatem obiektowym stosując elementy dziedziczenia, hermetyzacji oraz wiele innych związanych z tym typem programowania.

Projekt został napisany w środowisku Eclipse jako projekt typu Gradle zgodnie z wymaganiami projektowymi. Środowisko ułatwiło nam podstawowe prace nad projektem takie jak: podpowiadanie składni, jej podświetlanie czy import potrzebnych bibliotek.

W projekcie korzystamy głównie z bibliotek:

- **Swing** – biblioteka graficzna stosowana do tworzenia aplikacji z graficznym interfejsem użytkownika. Jest ona nowszą, ulepszoną wersją biblioteki AWT. Stosując bibliotekę Swinga w naszym projekcie użyliśmy jej komponenty oraz większość funkcjonalności jaką oferuje.
- **AWT** – Abstract Window Toolkit, jest to pakiet zawierający niezależny od platformy systemowej zestaw klas do projektowania aplikacji w środowisku graficznym. Wykorzystywana między innymi do rysowania grafik 2D oraz obsługi myszy.
- **Lang** – podstawowa biblioteka języka Java, domyślnie importowana dla każdej definiowanej klasy w projekcie.
- **IO** – biblioteka zawierająca klasy implementujące operacje wejścia oraz wyjścia. W projekcie stosowana do obsługi wyjątków związanych z łączeniem sieciowym.

## b) Struktura projektu:

Projekt został podzielony na dwa pakiety:

- warcaby.serwer
- warcaby.klient

W powyżej przytoczonych pakietach przechowywane są klasy związane z prawidłową obsługą gry po stronie serwera oraz klienta. Są to klasy (równocześnie dla Klienta jak i Serwera):

- Kafelek
- Opcje
- Plansza
- Warcaby

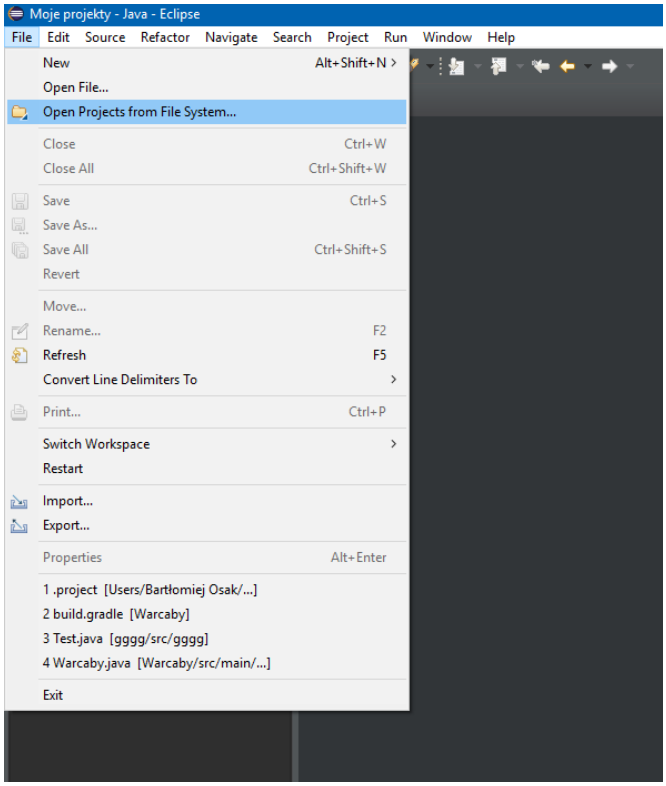
Ponadto każdy z pakietów zawiera plik o nazwie „package-info.java” potrzebny do przeprowadzanie prawidłowego opisu pakietu w dokumentacji generowanej za pomocą narzędzie Javadoc.

# 3. IMPORTOWANIE ORAZ KOMPILACJA PROJEKTU

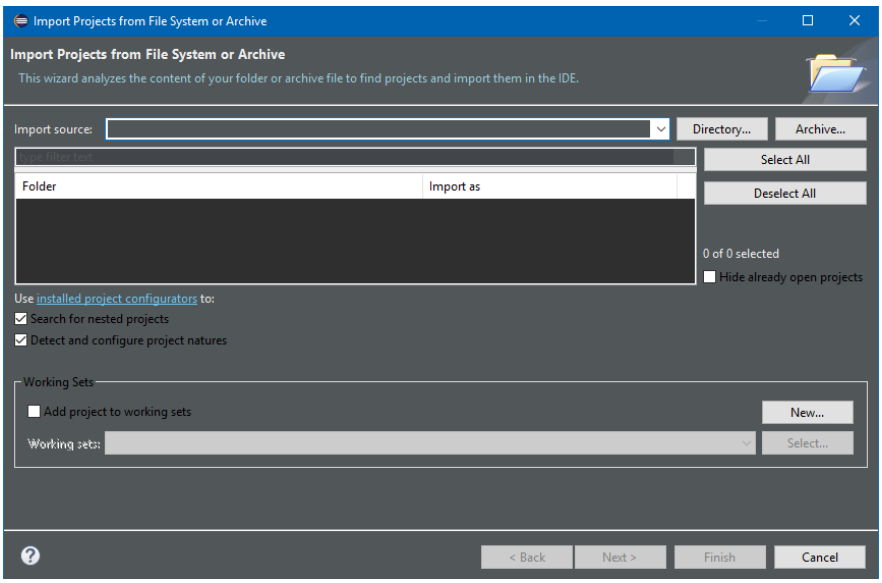
Projekt został napisany w IDE – Eclipse, lecz dzięki użyciu mechanizmu Gradle można go w łatwy sposób zaimportować do innych środowisk.

## a) Kompilacja projektu w Eclipse:

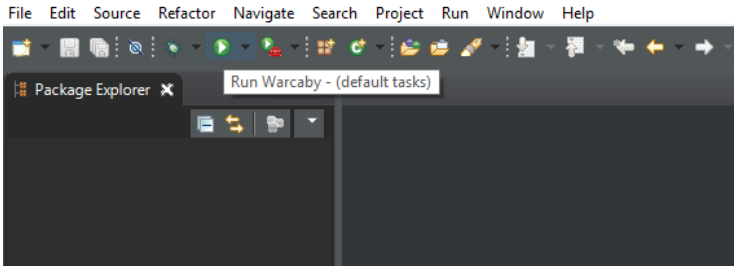
Należy uruchomić środowisko Eclipse. Następnie klikamy w lewym górnym rogu na przycisk „File” -> „Open Project from File System”:



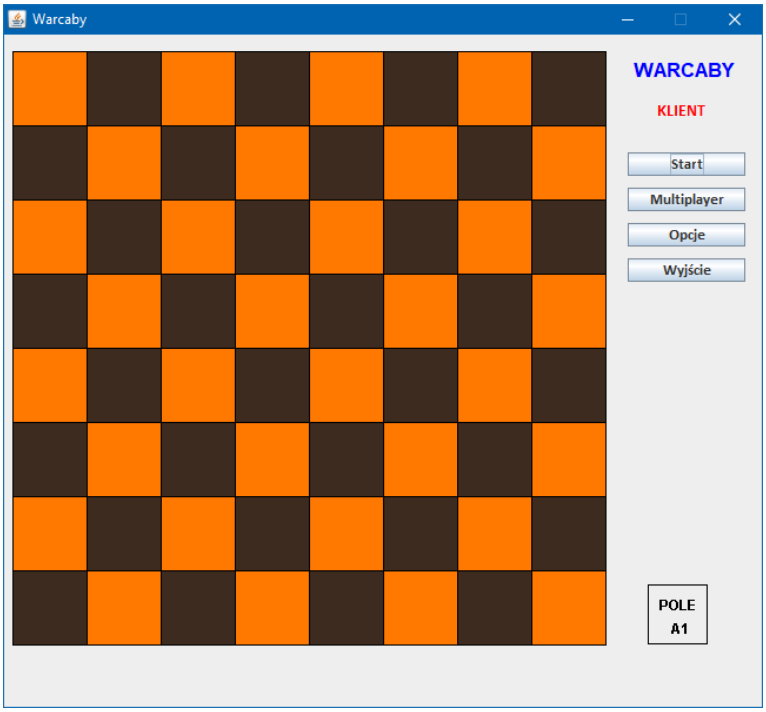

Po wybraniu tej opcji ukaże nam się ekran z prośbą o wskazanie ścieżki docelowej do folderu z projektem lub archiwum:



W przypadku pobrania projektu ze zdalnego repozytorium kodu jako archiwum należy wybrać opcję „Archive”, jeśli archiwum zostanie wypakowane należy wybrać opcję „Directory” wskazując ścieżkę docelową do folderu. Finalizując należy kliknąć na przycisk „Finish” i projekt zostanie poprawnie otwarty.
W celu jego skompilowania należy skorzystać z przycisku „Run”, który jest dostępy w pasku narzędzi:




Należy uważać, ponieważ każdy z dwóch pakietów posiada klasę „Warcaby”, która odpowiedzialna jest za uruchomienie gry. Gdy chcemy uruchomić grę lokalną należy skompilować „Warcaby” z poziomu pakietu Serwera lub Klienta. W przypadku gry sieciowej należy skompilować „Warcaby” na początku z poziomu pakietu Serwera, a następnie z poziomu pakietu Klienta. Po poprawnej kompilacji pojawi się okno rozgrywki.




**b) Kompilacja projektu w IntelliJ IDEA poprzez import Gradle:**

Gradle umożliwia nam szybki import projektu wraz z pobieraniem potrzebnych bibliotek. W naszym przypadku jest to zbędne działanie, ponieważ nasz projekt został napisany na podstawie bibliotek dostępnych w podstawowej Javie SE.
W celu importu naszego projektu w środowisku IntelliJ należy:

- w przypadku, gdy nie mamy otwartego żadnego projektu w środowisku wyświetli się ekran powitalny. Należy wybrać opcję Import Project i wskazać plik o nazwie „build” i rozszerzeniu „.gradle”. To wystarczy do zaimportowania w całości naszego projektu.

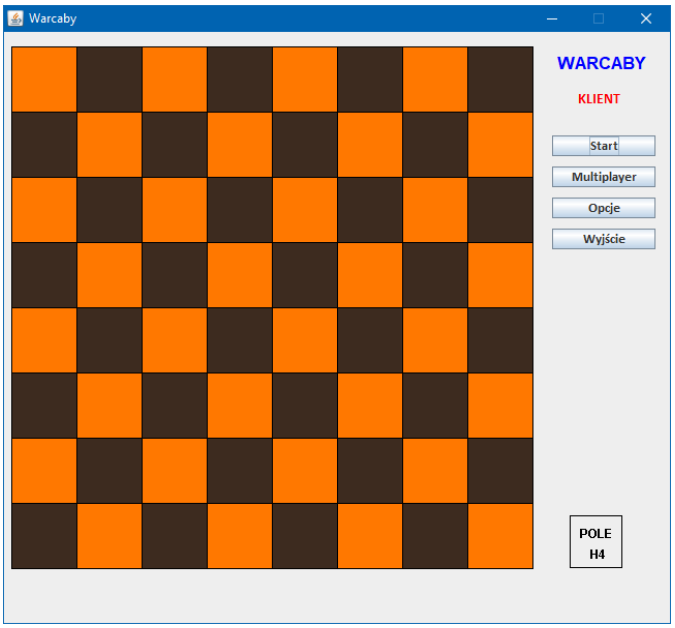
- w przypadku, gdy w środowisku mamy otwarty projekt należy w zakładce File wybrać opcję New -> Project from Existing Sources i wskazać plik „build.gradle”.

Po poprawnym zaimportowaniu możemy skompilować projekt trzymając się reguł opisanych w punkcie a). W środowisku IntelliJ należy wybrać klasę Warcaby a następnie prawym przyciskiem myszy kliknąć i wybrać opcję Run. Projekt zostanie poprawnie skompilowany i zostanie wyświetlony ekran rozgrywki.

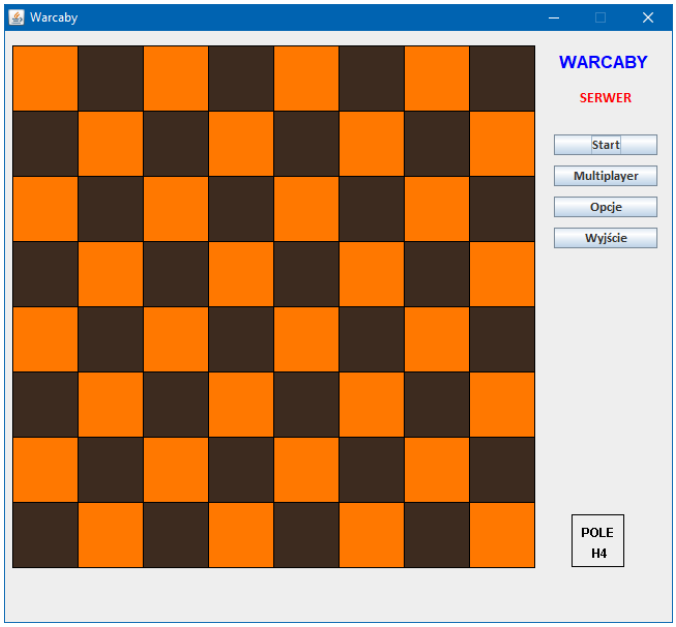
# 4. OPIS PROJEKTU

Projekt umożliwia przeprowadzenie rozgrywki w grę warcaby na planszy o liczbie 64 pól.
**Wygląd okna głównego aplikacji:**

- **dla klienta** (pierwszy gracz w rozgrywce sieciowej):



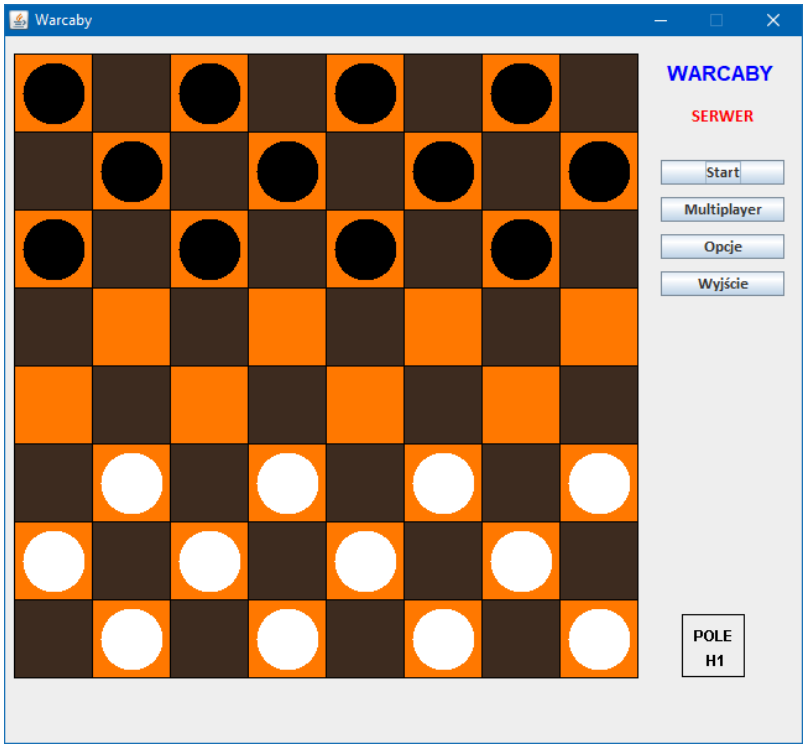

- **dla serwera** (drugi gracz w rozgrywce sieciowej):



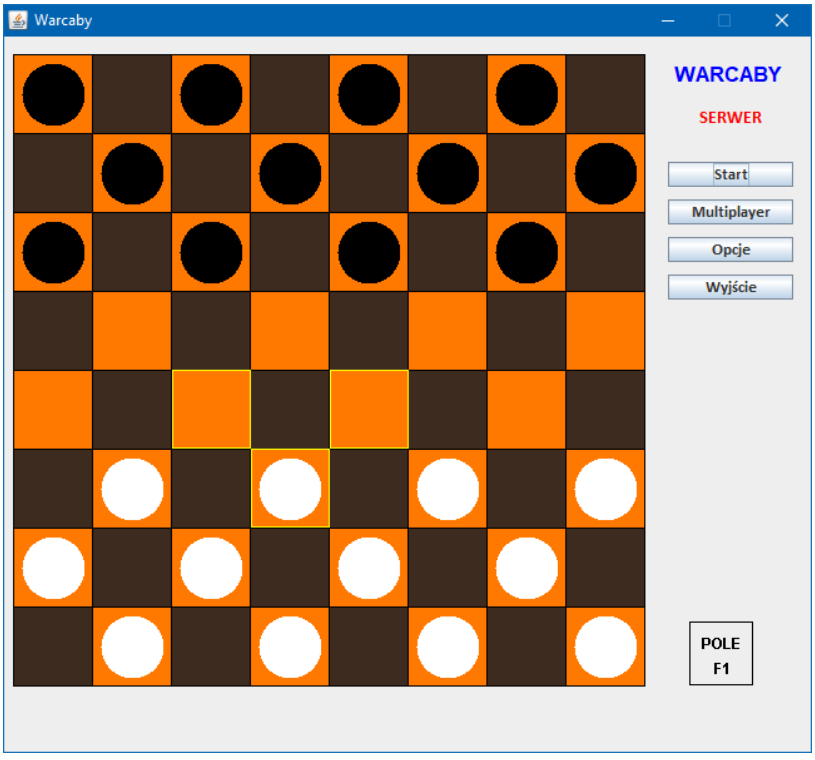
Gracz po stronie serwera jak i klienta posiada do wyboru 4 przyciski:

- **START**
- **MULTIPLAYER**
- **OPCJE**
- **WYJŚCIE**

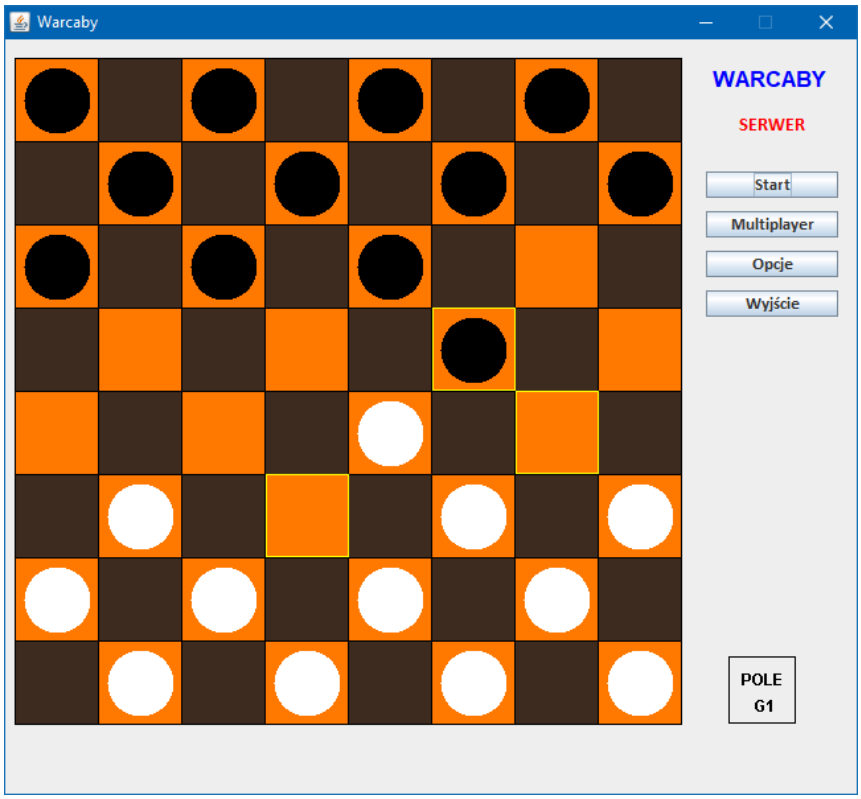
Wybór klawisza „START” spowoduje uruchomienie się gry lokalnej, czyli bez połączenia z siecią. Wygląd okna - widzimy również kafelek informacyjny podający informację o aktualnym położeniu kursora myszy gracza (wskazuje współrzędne pola planszy – szachownicy):



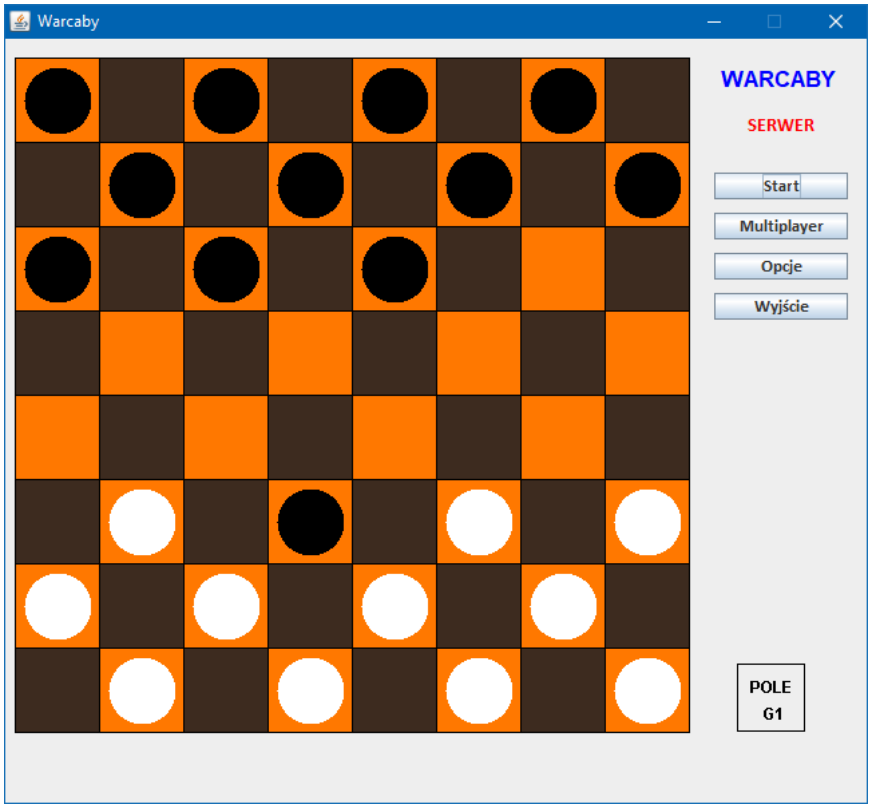

Teraz jest już możliwa rozgrywka lokalna. Klikając na danego pionka podświetlają się pola, na które dany pionek może być przeniesiony:



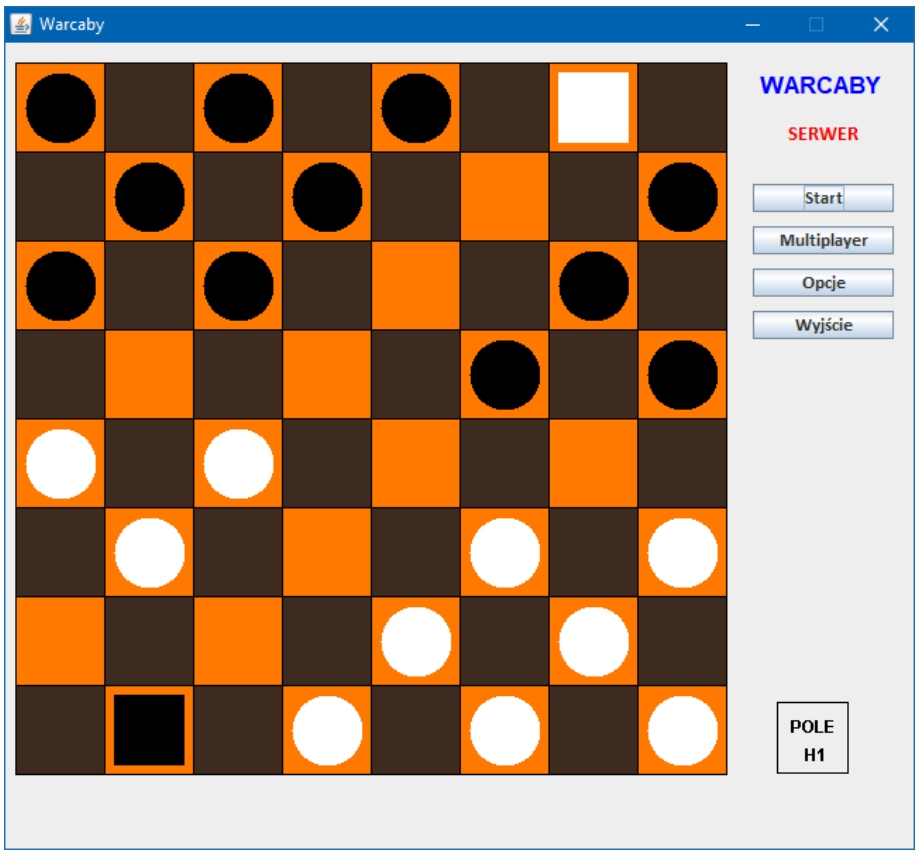
W przypadku możliwości bicia pojawi się podświetlenie na pole, na które możemy przejść danym pionkiem zbijając pionka koloru przeciwnego:



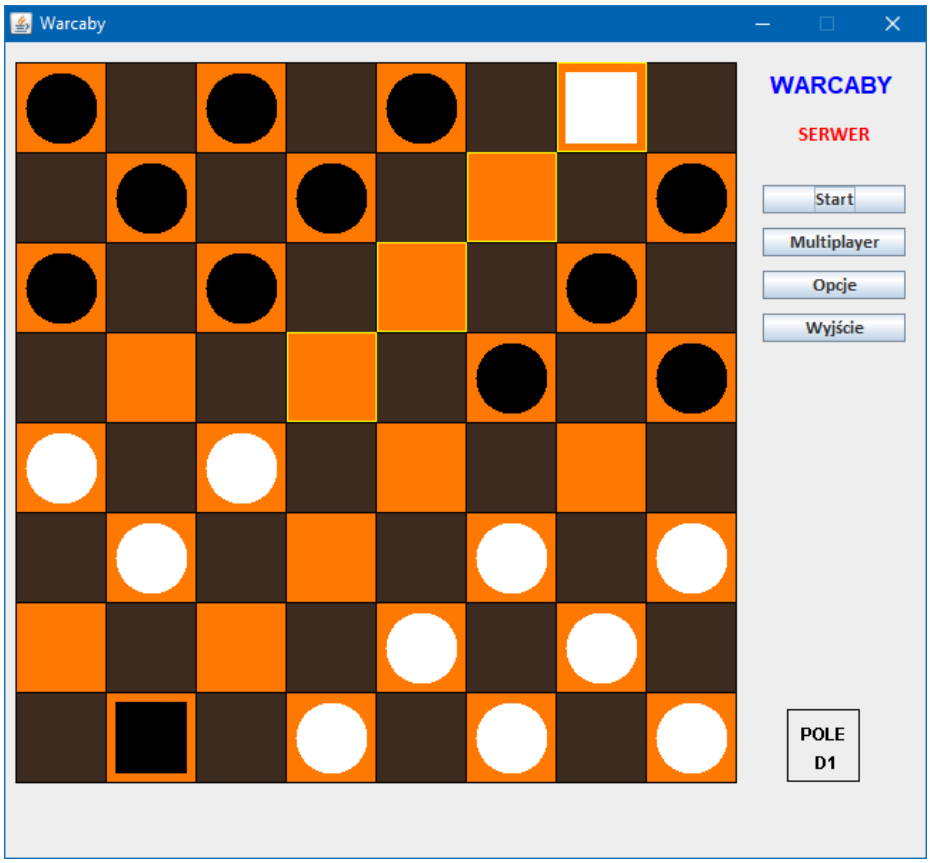

Oczywiście w przypadku zbicia pionek zbity znika i udostępnia wolne pole:



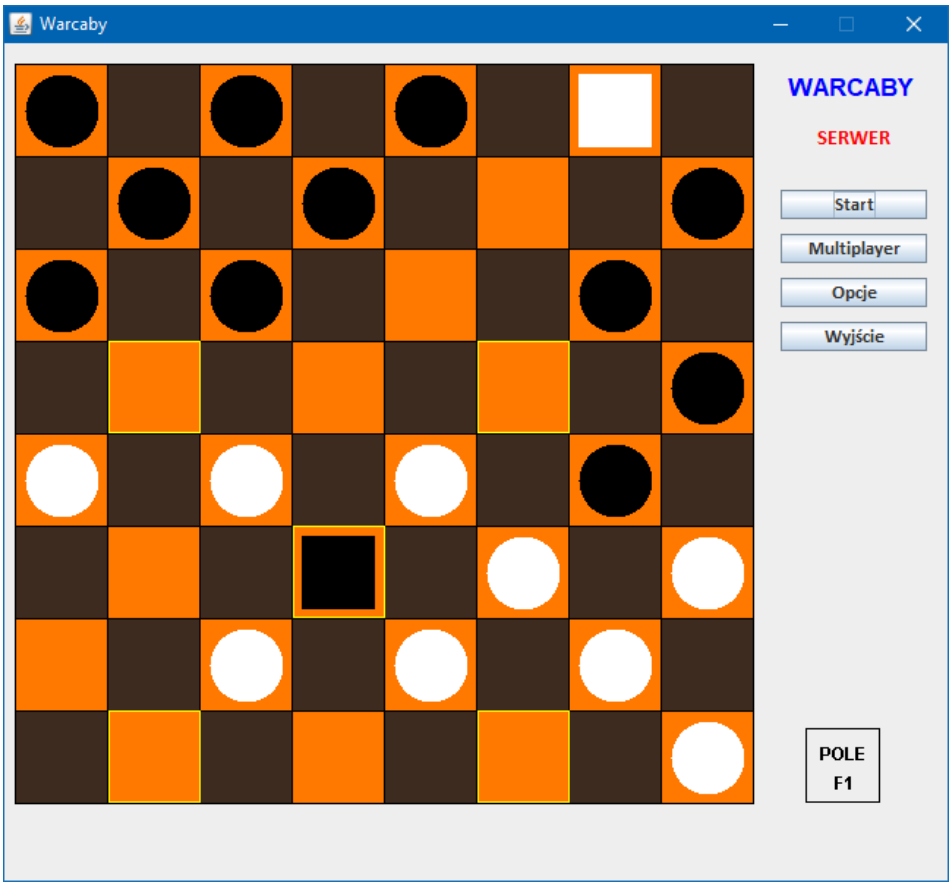
Zgodnie z zasadami rozgrywki zaimplementowaliśmy również występowanie damek, inaczej nazywanymi królówkami. Pojawią się wtedy, gdy pion koloru przeciwnego wejdzie na pola o indeksach A-H8 (dla ciemnych) oraz A-H1 (dla jasnych):



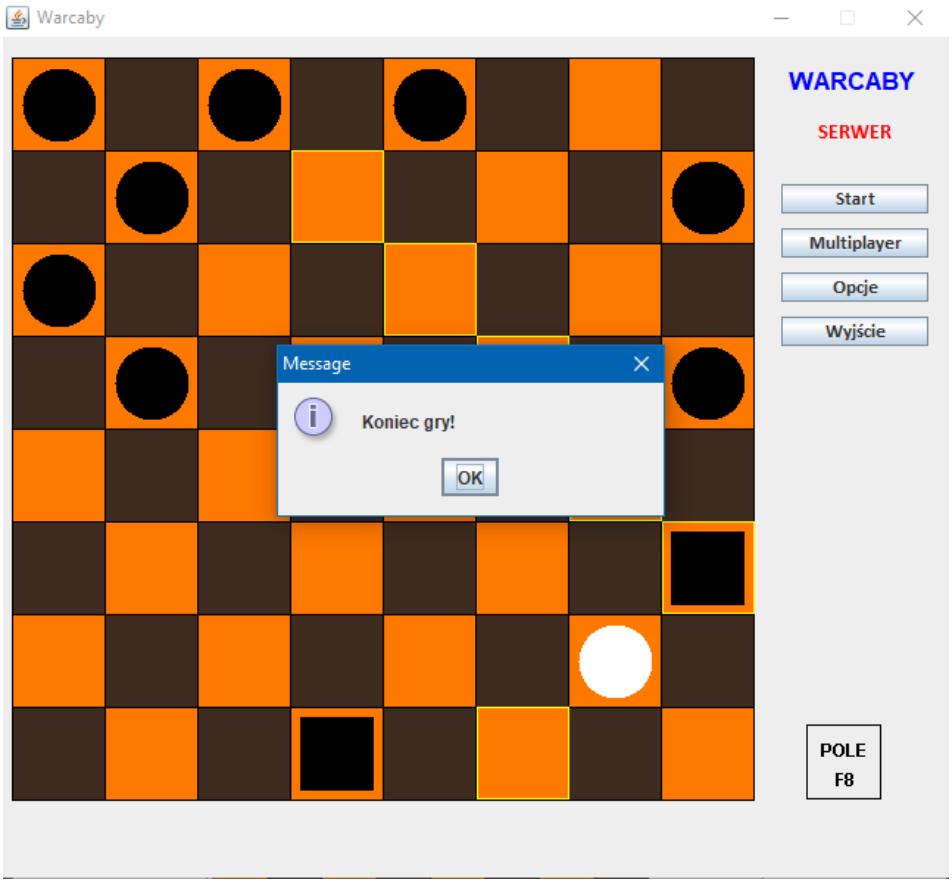

Oczywiście damki mają możliwości ruchu zgodnie z zasadami ogólnej rozgrywki warcabowej. Poniżej przykład:



W projekcie zaimplementowaliśmy również system bić dla damek. Damka może posiadać do 4 wariantów bić jednocześnie. Poniżej jeden z przykładów możliwego bicia:

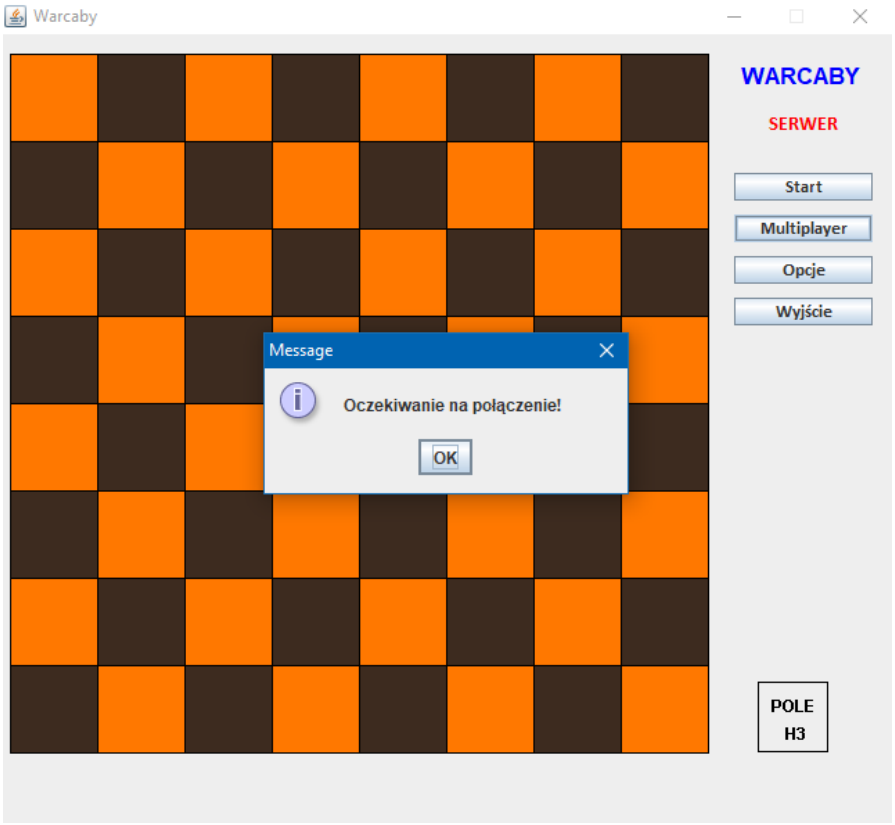



W przypadku wygranej jednej ze stron, czyli w przypadku zbicia wszystkich pionków przeciwnika wyświetlany jest komunikat:

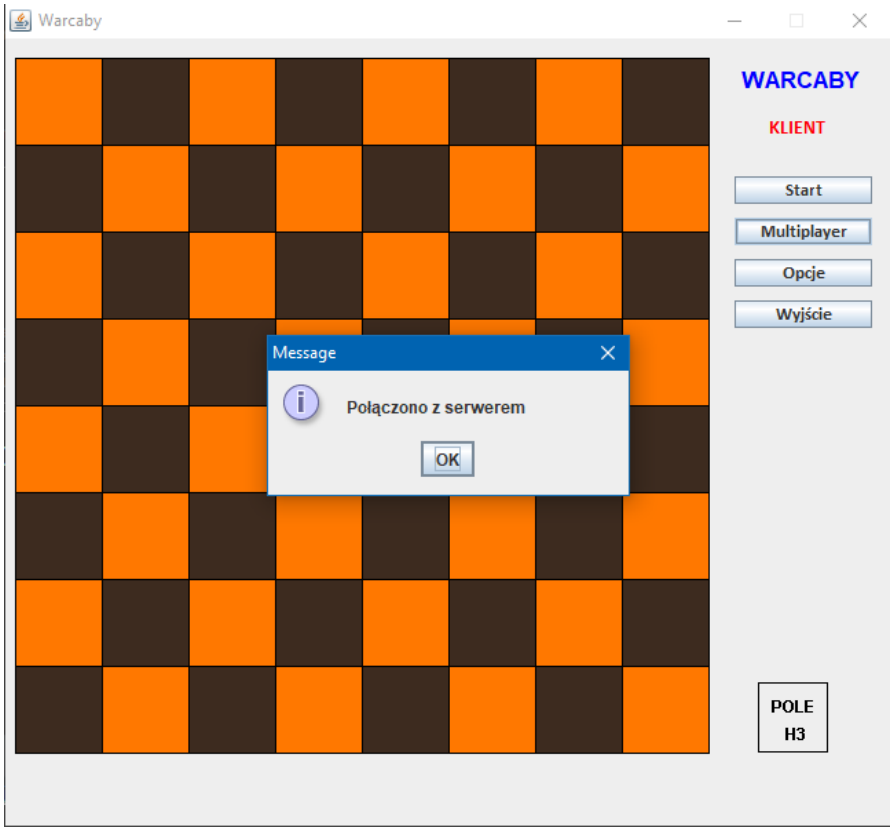


Użytkownik musi potwierdzić klikając na przycisk OK a następnie ma możliwość wyboru ponownej rozgrywki lokalnej lub sieciowej wybierając odpowiedni przycisk.

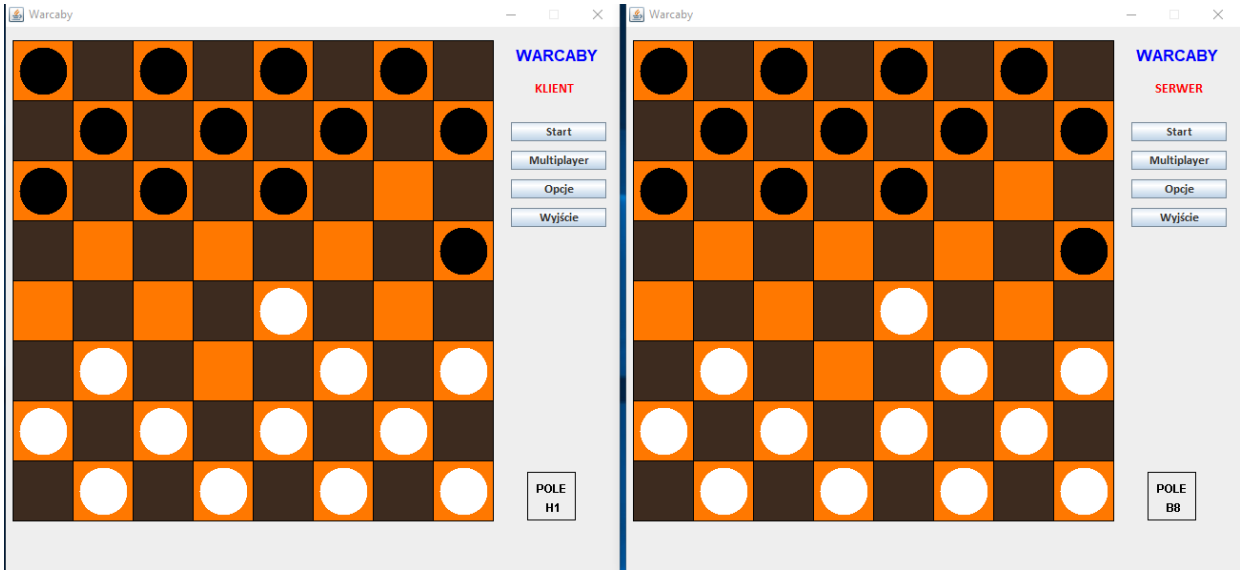
W przypadku rozgrywki sieciowej aplikację otwiera osoba, która zaprasza inną osobę do gry – identyfikujemy ją jako serwer. Gospodarz rozgrywki sieciowej musi uruchomić aplikację i wybrać przycisk Multiplayer. Po wybraniu tej opcji wyświetlany jest komunikat:



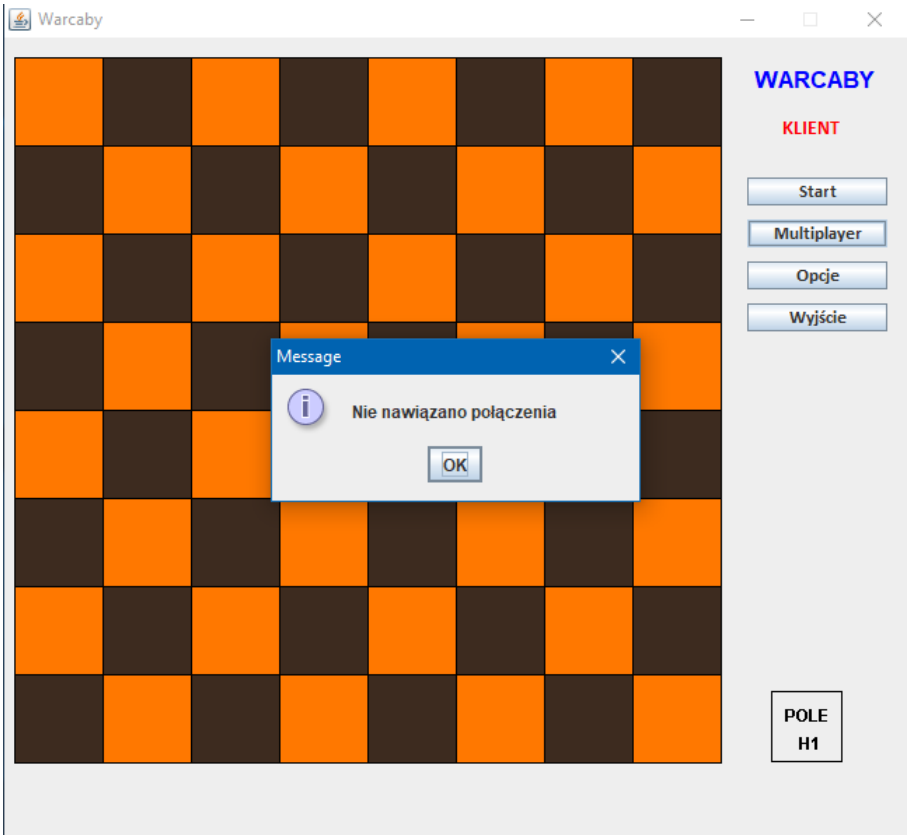

Klikając OK potwierdzamy wolę oczekiwanie na zgłoszenie się gracza z poziomu aplikacji klienckiej. Klikając na przycisk Multiplayer z poziomu drugiego gracza pojawia się:



Klikając OK pojawią się pionki na szachownicy i będzie można rozpocząć rozgrywkę:

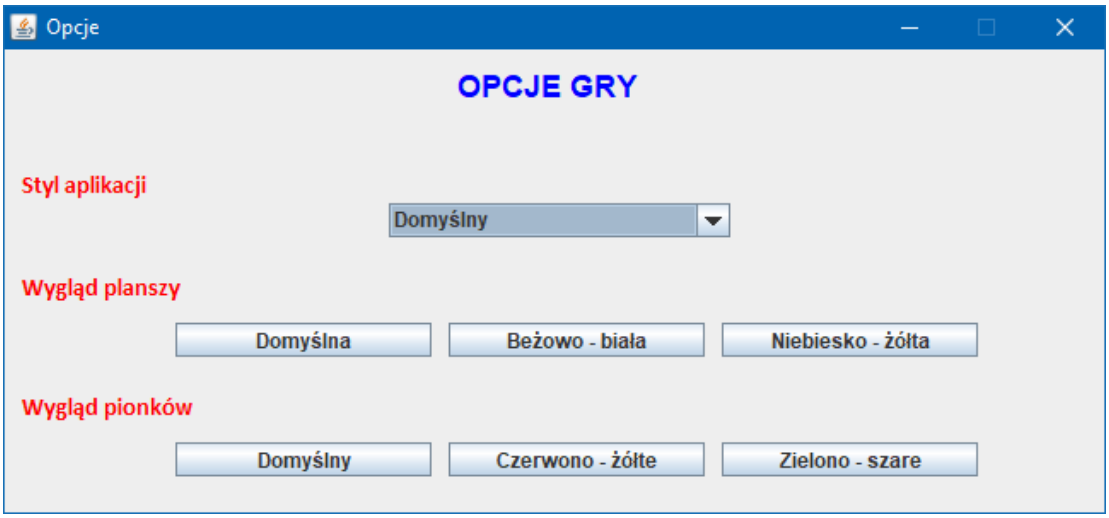



Każdy ruch gracza jest odwzorowywany na planszy drugiego gracza. W przypadku, gdy gracz – klient nie będzie mógł się połączyć z graczem będącym serwerem wyświetli się komunikat:




W tym wyjątkowym przypadku należy zaakceptować komunikat przyciskiem OK i gracz będzie miał do dyspozycji wszystkie opcje dostępne po prawej stronie okna głównego aplikacji.

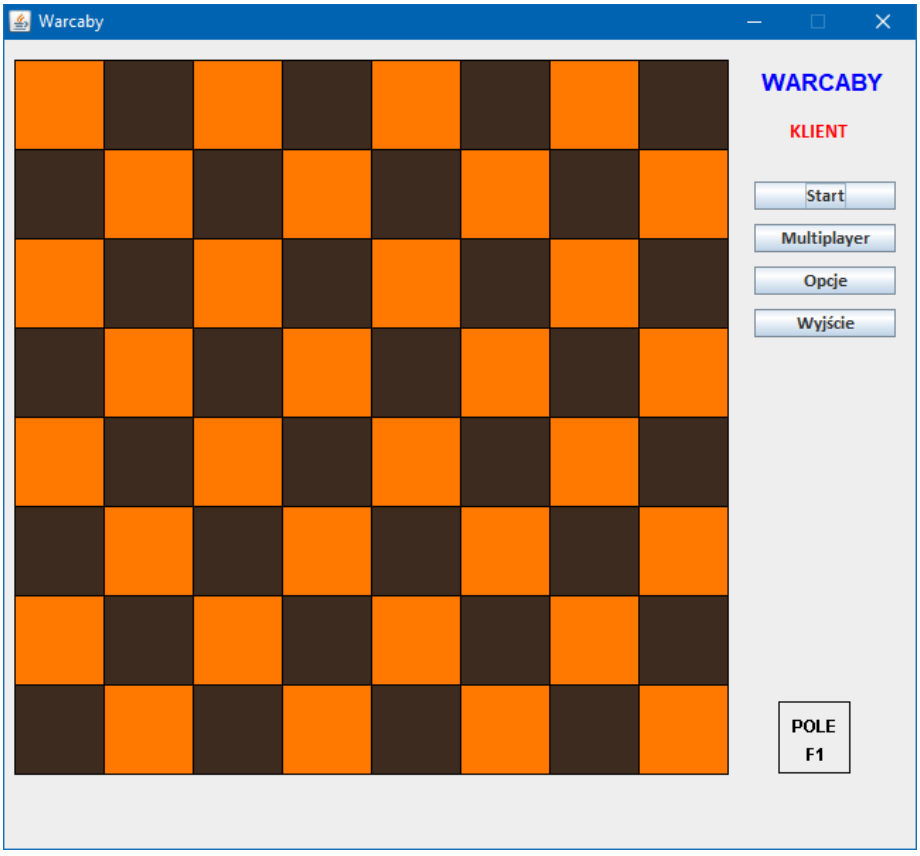
W grze zaimplementowaliśmy również opcje. Dostęp do nich jest realizowany poprzez kliknięcie na przycisk Opcje w głównym oknie gry. Pojawi się zatem okno:



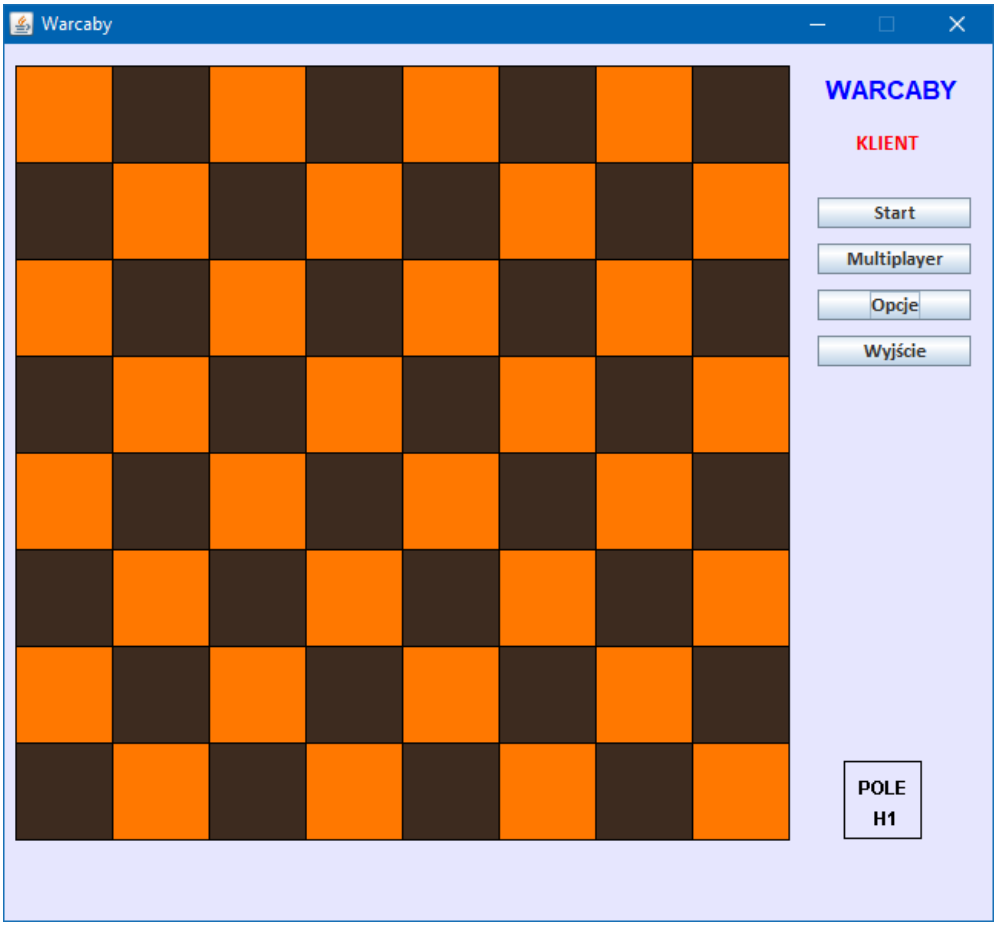

Gracz ma trzy możliwości personalizacji gry. Warto wspomnieć, iż wygląd może być personalizowany podczas gry lokalnej jak i sieciowej – wygląd aplikacji nie jest wysyłany przez sieć, więc stanowi indywidualny wybór dla każdego z graczy. Oto metody personalizacji:

- **Styl aplikacji** – zmienia kolor okna głównego gry. Do wyboru są 4 różne kolory o nazwach: Domyślny, Metal, Motif oraz Nimbus.

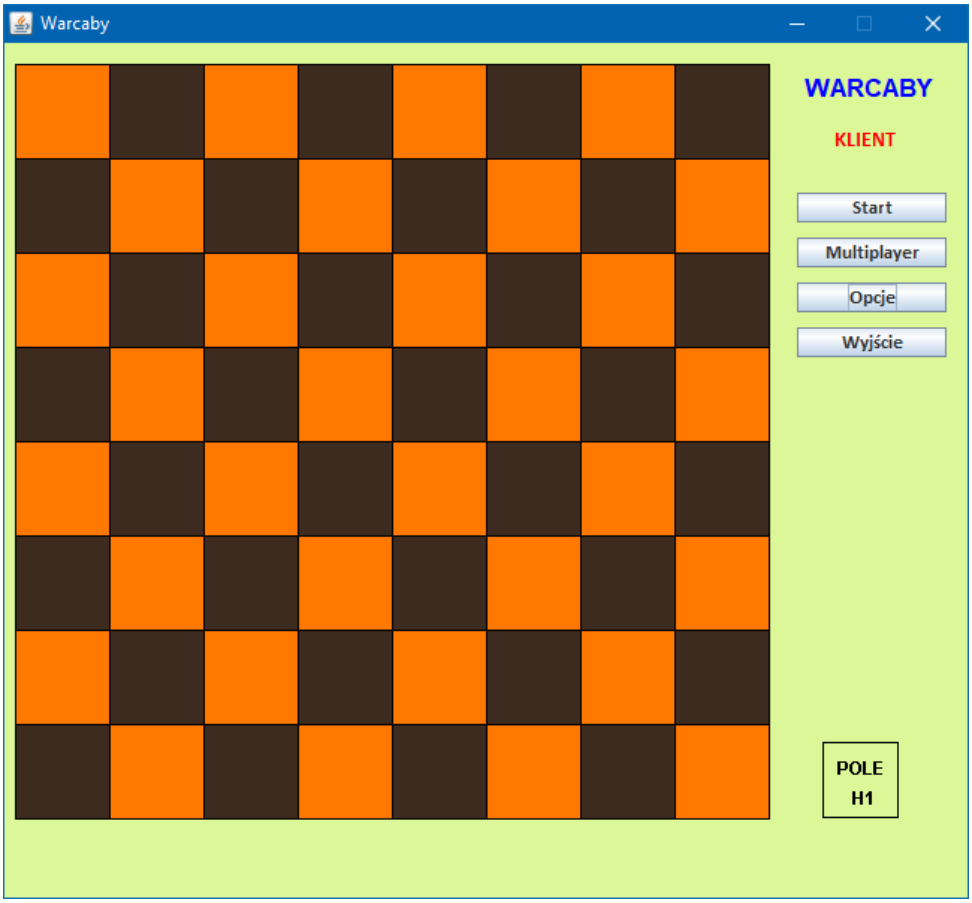
  **Wygląd domyślny:**



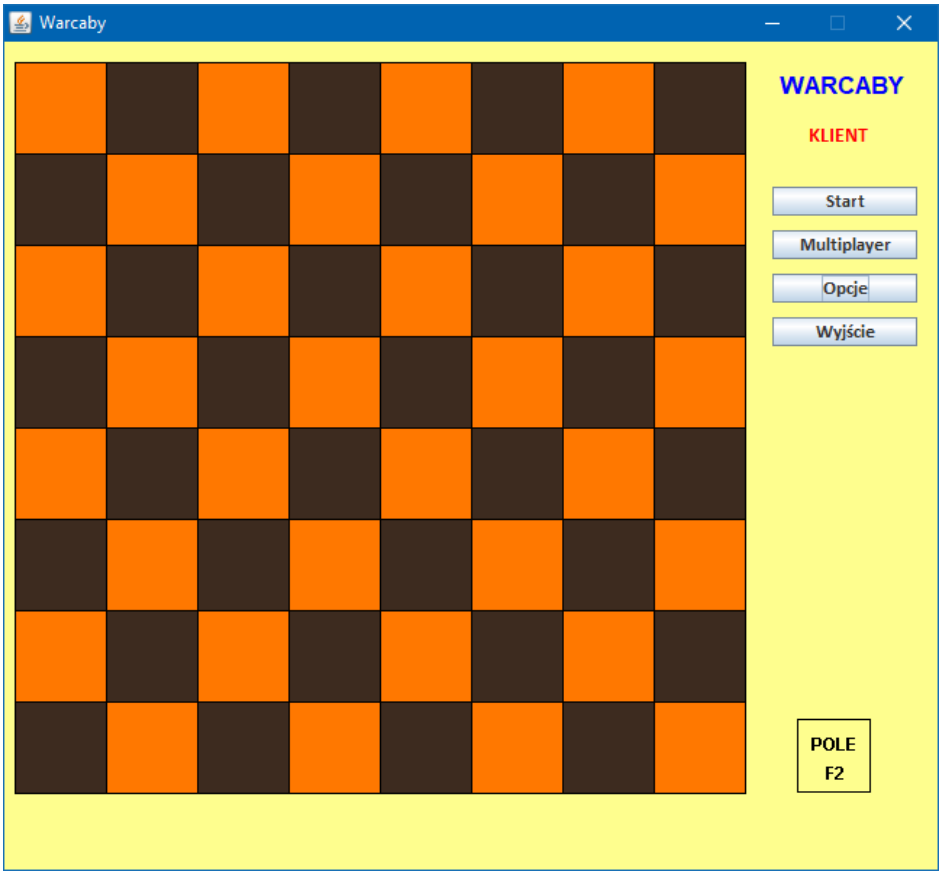
**Wygląd „Metal”:**



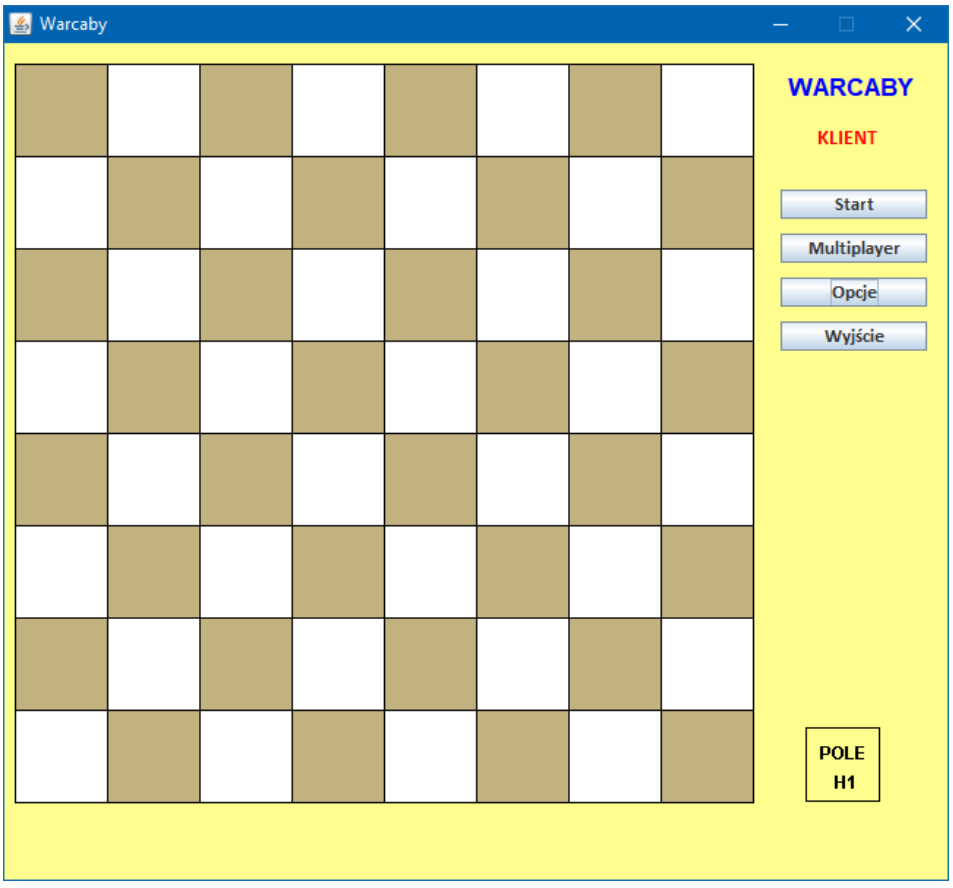

**Wygląd „Motif”:**

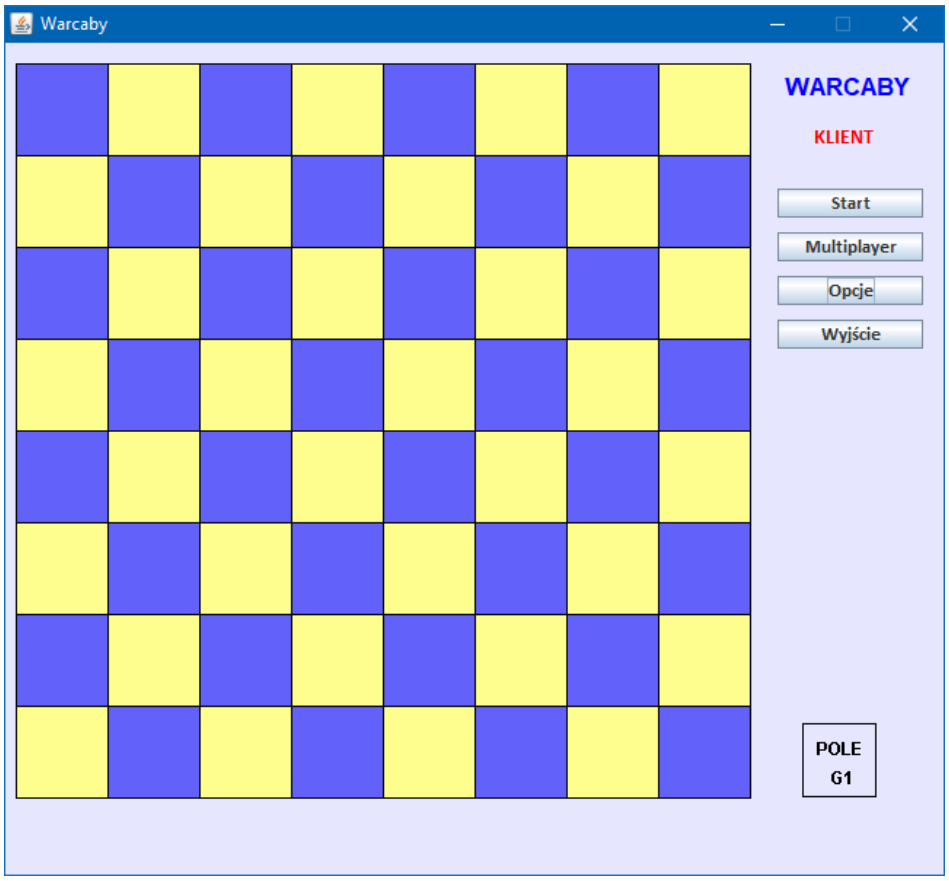


**Wygląd „Nimbus”:**



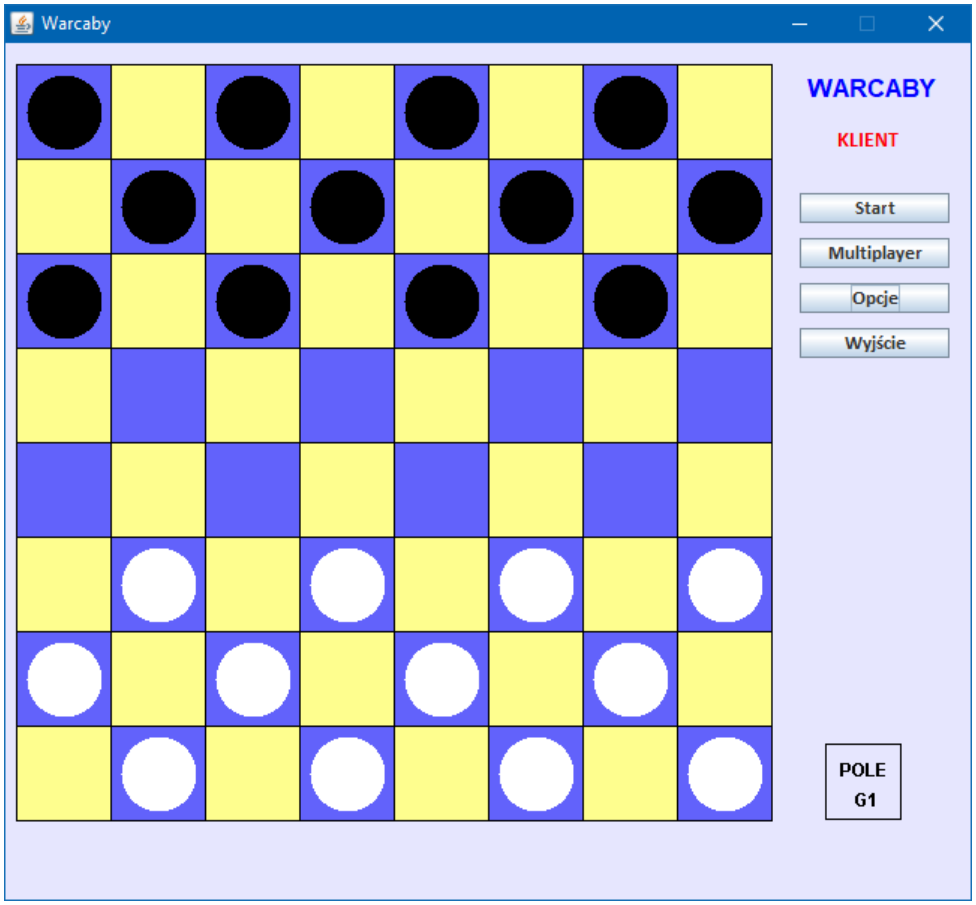

- **Wygląd planszy** – zmienia kolor planszy do gry. Do wyboru trzy kolory – domyślny prezentowany powyżej oraz dwa dodatkowe – zmiana powoduje wyświetlanie się okna dialogowego potwierdzającego zmianę koloru.



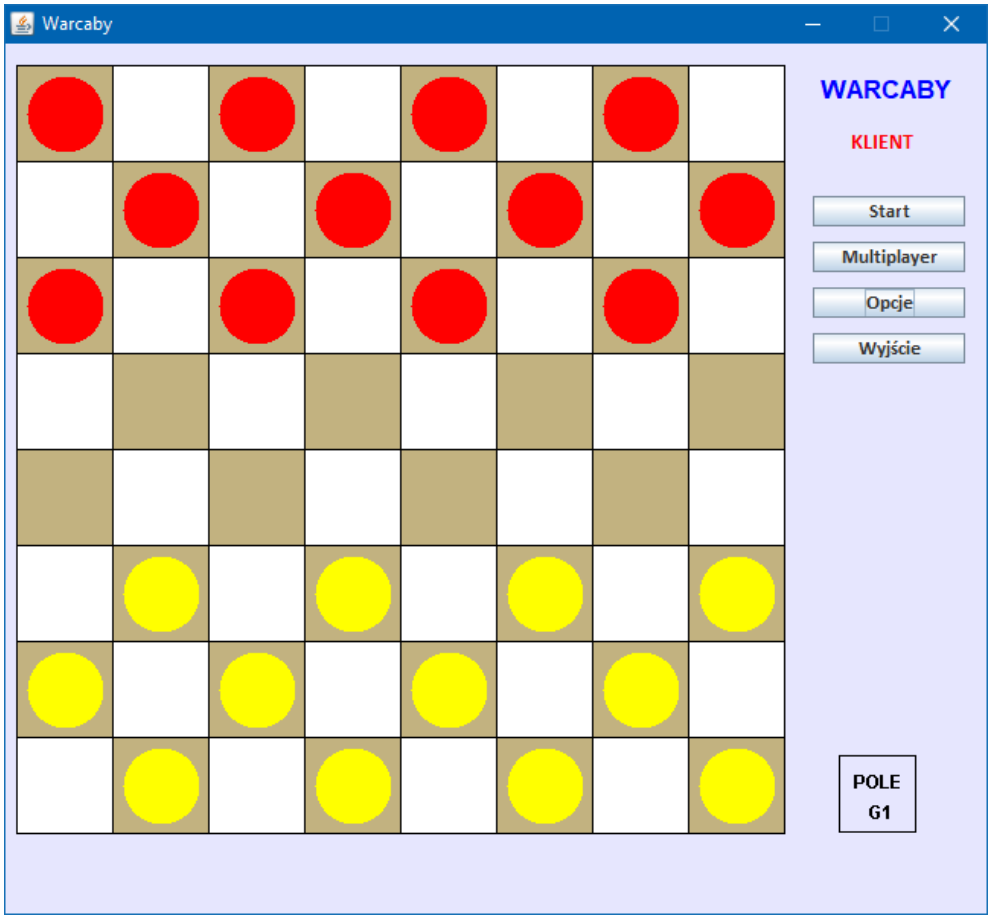



- **Wygląd pionków** – zmienia kolor pionków do gry. Do wyboru trzy kolory – domyślny oraz dwa dodatkowe - zmiana powoduje wyświetlanie się okna dialogowego potwierdzającego zmianę koloru:

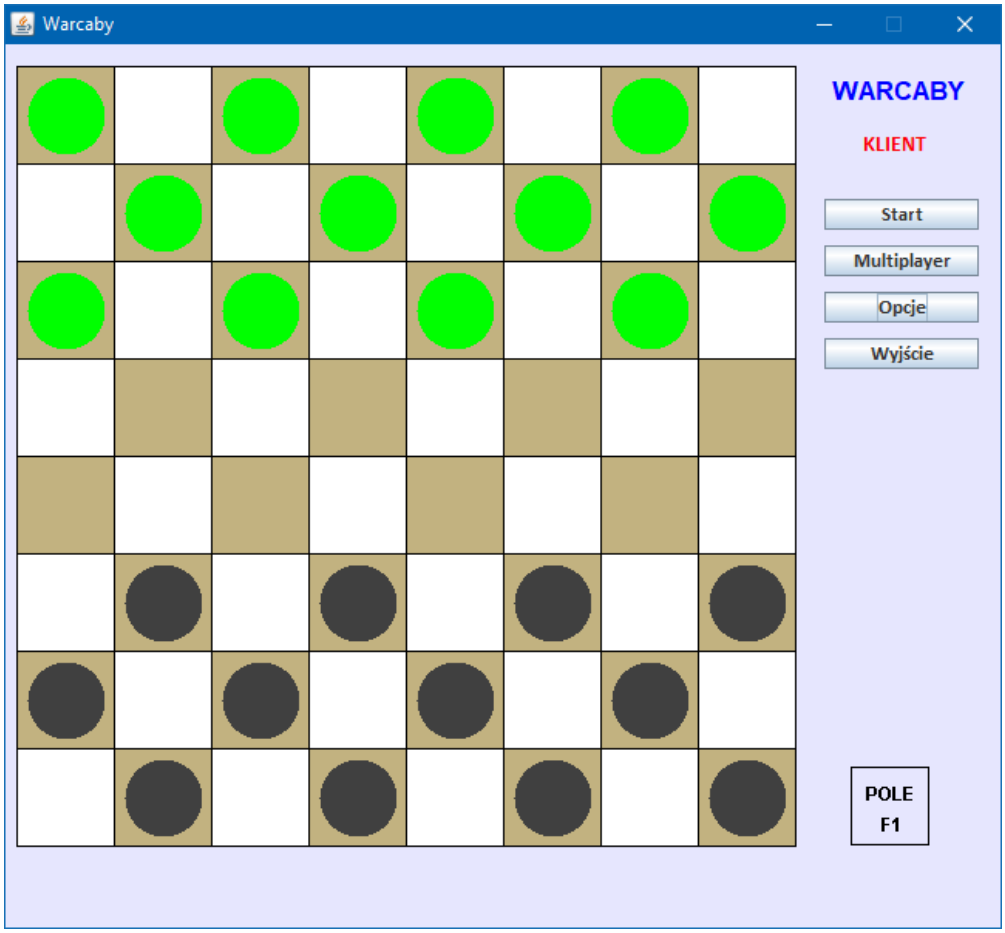
**Wygląd domyślny:**



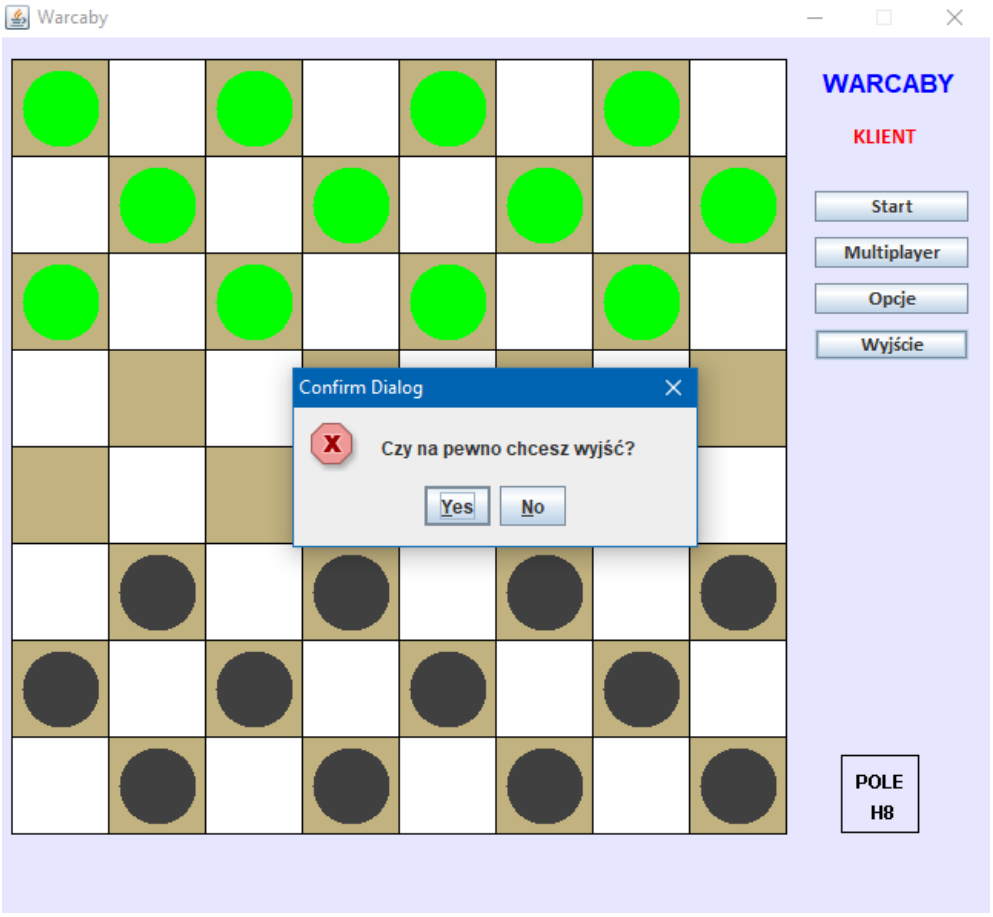
**Wygląd dodatkowy pierwszy – czerwono-żółte:**




**Wygląd dodatkowy drugi – zielono-szare:**



Ostatnia opcja dostępna na pasku obsługi gry w oknie głównym to wyjście z aplikacji. W przypadku wybrania tej opcji wyświetli się okno z prośbą o potwierdzenie:




Oczywiście w przypadku wybrania opcji „Yes” aplikacja zamknie się i połączenie w przypadku gry sieciowej zostanie przerwane poprzez zamknięcie gniazd sieciowych. W przeciwnym przypadku okno dialogowe zniknie i użytkownik będzie mógł powrócić do głównego okna aplikacji. Istnieje również drugi sposób na wyjście klikając krzyżyk w prawym górnym rogu głównego okna gry.

Wszystkie opisane komponenty zostały napisane za pośrednictwem biblioteki Swing oraz AWT. Mechanizm gry został napisany w czystym języku Java w sposób, który zostanie opisany w następnym punkcie sprawozdania projektowego. Względem klasycznych rozgrywek warcabowych dodaliśmy kafelek informacyjny, który może pomóc w nawigowaniu po planszy. Ponadto opcje gry tworzą możliwość personalizacji wyglądu aplikacji. Ponadto rozgrywka sieciowa może okazać się ciekawą formą gry w porównaniu z formą lokalną.

# 5. OPIS KOMPONENTÓW GRY

Projekt został podzielony na dwa pakiety:

- warcaby.serwer
- warcaby.klient

W powyżej przytoczonych pakietach przechowywane są klasy związane z prawidłową obsługą gry po stronie serwera oraz klienta. Są to klasy (równocześnie dla Klienta jak i Serwera):

- Kafelek
- Opcje
- Plansza
- Warcaby

Opis klas oraz ich metod dla pakietu **Serwer,** który zawiera wszystkie klasy oraz mechanizmy wymagane do poprawnej kompilacji oraz uruchomienia gry po stronie gracza – gospodarza.

**Klasa warcaby.serwer.Kafelek:**

**Opis:**

Klasa Kafelek zawiera podstawowe zmienne oraz metody obsługi ruchu pionków po szachownicy oraz określa również bicia. Klasa posiada zmienne oraz metody odpowiedzialne za określenie pojedynczego kafelka występującego na szachownicy, dzięki temu można łatwo określić możliwe ruchy dla pionka oraz możliwości bicia. Ponadto w klasie zdefiniowano wystąpienie królówek zgodnie z zasadami gry - warcaby. Zdefiniowano również działanie kafelka informacyjnego, który pokazuje współrzędne kursora użytkownika względem szachownicy - kafelek znajduje się w prawym dolnym rogu głównego okna aplikacji. Klasa Kafelek dziedziczy po klasie JComponent celem stworzenia obiektu kafelka informacyjnego. Ponadto rozszerzona jest ona poprzez interfejs MouseListener celem zdefiniowania akcji myszą - czyli podstawowego kontrolera rozgrywki - poprzez klikanie myszą na odpowiednie pionki możemy je przemieszczać oraz wykonywać bicia.

Metody:

- `public Kafelek(JFrame f, int i, int j)`

  Opis:
  Konstruktor klasy Kafelek inicjalizuje zmienne typu JFrame - odwołanie do okna głównego aplikacji, zmienne określające współrzędne.

  Parametry:
  **f** - parametr JFrame - główne okno aplikacji

  **i** - współrzędna x napisu w kafelku informacyjnym

  **j** - współrzędna y napisu w kafelku informacyjnym

- **protected void ustawNazwe**(int i, int j)

  Opis:
  Metoda ustawNazwe ma za zadanie poprawne wyświetlanie informacji na kafelku informacyjnym. Przyjmuje dwa parametry wywołania. W zależności od współrzędnych informacja jest wyświetlana poprzez dodanie do zmiennej StringBuilder nowych treści poprzez metodę append().

  Parametry:
  **i** - współrzędna x pojedynczego kafelka na szachownicy.

  **j** - współrzędna y pojedynczego kafelka na szachownicy.

- **public void mouseEntered(MouseEvent e)**

  Opis:
  Metoda przesłonięta pochodząca z interfejsu MouseListener. Określa ona zachowanie programu podczas poruszania się kursorem myszy po oknie aplikacji, czyli gdy kursor myszy najeżdża na dany komponent - w naszym przypadku komponentem jest okno główne aplikacji - w szczególności szachownica. W naszym przypadku wykorzystywana jest do aktualizacji współrzędnych wyświetlanych na kafelku informacyjnym.

  Parametry: domyślny MouseEvent

- **public void mouseClicked(MouseEvent e)**

  Opis:
  Metoda przesłonięta pochodząca z interfejsu MouseListener. Określa ona zachowanie programu podczas, gdy użytkownik kliknie myszką na komponent. Funkcja wykorzystywana do opracowania logiki ruchu pionków oraz bić pionków. Ważne oznaczenia w celu zrozumienia algorytmu ruchu oraz bić pionów:
  *0* - oznaczenie pola pustego
  *1* - oznaczenie pionka czarnego
  *2* - oznaczenie pionka białego
  *3* - oznaczenie wybranego pionka czarnego (wybrany, czyli kliknięty myszką przez gracza)
  *4* - oznaczenie wybranego pionka białego (wybrany, czyli kliknięty myszką przez gracza)
  *5* - oznaczenie pola pustego, na które można wykonać ruch (przesunąć pionka)
  *6* - oznaczenie białej królówki
  *7* - oznaczenie czarnej królówki
  *8* - oznaczenie wybranej białej królówki
  *9* - oznaczenie wybranej czarnej królówki
  Takie rozgraniczenie pozwala na dość czytelne odczytanie algorytmu ruchu oraz bić. Ponadto w metodzie wysyłany jest obiekt tablicy pionków poprzez sieć do drugiego gracza. W metodzie zdefiniowano również warunek zakończenia gry - zakończenie gry sygnalizowane jest specjalnym komunikatem wyświetlanym w oknie dialogowym.

  Parametry: domyślny MouseEvent

- **`public String toString()`**

  Opis:
  Przeciążona metoda toString(), która ma na celu prawidłowe wyświetlanie informacji na kafelku informacyjnym.

  Parametry: brak

- **`public void mouseExited(MouseEvent e)`**

  Opis:
  Metoda przesłonięta pochodząca z interfejsu MouseListener - niewykorzystywana. Jej obecność jest obowiązkowa w celu poprawnego zaimplementowania np. metody mouseClicked(MouseEvent).

  Parametry: domyślny MouseEvent

- **`public void mousePressed(MouseEvent e)`**

  Opis:
  Metoda przesłonięta pochodząca z interfejsu MouseListener - niewykorzystywana. Jej obecność jest obowiązkowa w celu poprawnego zaimplementowania np. metody mouseClicked(MouseEvent).

  Parametry: domyślny MouseEvent

- **`public void mouseReleased(MouseEvent e)`**

  Opis:
  Metoda przesłonięta pochodząca z interfejsu MouseListener - niewykorzystywana. Jej obecność jest obowiązkowa w celu poprawnego zaimplementowania np. metody mouseClicked(MouseEvent).

  Parametry: domyślny MouseEvent

## Klasa warcaby.serwer.Plansza:

**Opis:**

Klasa Plansza zawiera deklaracje oraz definicje wszystkich komponentów tworzących okno główne aplikacji. W klasie zdefiniowano wygląd ogólny okna gry, wygląd planszy, której wygląd można zmienić poprzez opcje gry - Opcje. Ponadto w klasie zadeklarowano zmienne potrzebne do nawiązania połączenia sieciowego z drugim graczem - tu z klientem. W klasie ustawiane są również początkowe współrzędne tablicy pionków na szachownicy. Ponadto określono działanie przycisków - w szczególności przycisk 'Multiplayer' (zdefiniowano wątek, który odczytuje obiekt ze strumienia umożliwiającego zapis danych do gniazda (w tym przypadku tablica pionków - tablicaPionkow. Klasa implementuje interfejs ActionListener odpowiadający za określenie akcji po wciśnięciu np. danego przycisku. Ponadto implementowany jest interfejs Serializable w celu serializacji obiektu wysyłanego poprzez połączenie sieciowe TCP. Klasa dziedziczy po JFrame - klasie tworzącej główne okno aplikacji.

Metody:

- **`public Plansza()`**

  Opis:
  Konstruktor bezparametrowy klasy Plansza odpowiedzialny za inicjalizację komponentów z biblioteki Swing, zmiennych, tablic. Komponenty definiowane: JButton, JLabel, JFrame - dla tych komponentów ustawiane są wymiary, fonty, kolory. Po zdefiniowaniu każdego z użytych komponentów dodano go do okna z opcjami głównymi metodą Container.add(java.awt.Component). Ponadto definiowane są tablice obiektów tj. tablica obiektów klasy Kafelek, tablica współrzędnych pionków na szachownicy. Zdefiniowano również zmienne typu Color, które określają kolor planszy lub kolor pionków.

  Parametry: brak

- **`public void ustaw()`**

  Opis:
  Metoda bezparametrowa odpowiedzialna za ustawianie pionków na szachownicy. Operacja przeprowadzana jest poprzez użycie pętli for oraz rozgraniczenie dwóch kolorów pionów od siebie - jeden kolor oznaczany jest poprzez 1, a drugi poprzez 2. Pole puste poprzez 0. Po poprawnym ustawieniu zmienna logiczna 'gra' jest ustawiana na wartość TRUE. Metoda jest typu void - nie zwraca żadnej wartości.

  Parametry: brak

- **`public void paint(Graphics g)`**

  Opis:
  Metoda przesłonięta o nazwie paint. Przyjmuje jeden stały parametr typu Graphics. Metoda wykorzystywana jest do rysowania szachownicy oraz pionków. Ponadto w klasie zaadaptowano również rysowanie podświetleń konturów aktywnych kafelków, na które możemy przemieścić gracz swój pionek - są to tak zwane pola aktywne do ruchu dla pionka. W metodzie rysowany jest również kafelek informacyjny umieszczony w prawym dolnym rogu okna głównego aplikacji, który informuje użytkownika na jakim polu szachownicy obecnie znajduje się jego kursor myszy. Rysowanie odbywa się za pomocą komponentu Graphics2D. Wykorzystywane są funkcje takie jak: Graphics.drawRect(int, int, int, int),Graphics.setColor(Color), Graphics.fillRect(int, int, int, int),Graphics.fillOval(int, int, int, int) itp. Metoda jest typu void - nie zwraca żadnej wartości.

  Parametry: domyślny Graphics

- **`public void actionPerformed(ActionEvent e)`**

  Opis:
  Przesłonięta metoda służąca do określania zachowania aplikacji po kliknięciu na dany komponent przez użytkownika. W metodzie tej określono działanie dla poszczególnych przycisków znajdujących się po prawej stronie głównego interfejsu gry. W przypadku kliknięcia na przycisk 'Start' wywoływana jest metoda ustaw(), która ustawia pionki na początkowe pozycje. Następnie wywoływana jest zdefiniowana metoda Component.repaint(), celem wyświetlenia zmian na interfejsie gry. W przypadku

kliknięcia na przycisk 'Opcje' tworzona jest nowa instancja klasy Opcje, poprzez wywołanie konstruktora tej klasy. Otworzy się wtedy drugie okno z opcjami gry do wyboru. W przypadku kliknięcia na przycsik 'Multiplayer' tworzone jest gniazdo serwerowe, powiązane z podanym portem. Następnie jeśli wiązanie przebiegło poprawnie wyświetlany jest komunikat za pomocą okna dialogowego klasy JOptionPane ze stosowną informacją o oczekiwaniu serwera na zgłoszenie się klienta go gry. Dalej za pomocą funkcji ServerSocket.accept() nasłuchiwane są połączenia przychodzące od aplikacji klienckich na danym porcie i są akceptowane. Następnie inicjalizowane są strumienie zapisu do gniazda lub odczytu z gniazda. Dalej zmienna logiczna o nazwie 'multi' jest ustawiana na wartość 'true'. Następnie wywoływana jest metoda ustaw(), która ustawia pionki na początkowe pozycje oraz zdefiniowana metoda Component.repaint(), celem wyświetlenia zmian na interfejsie gry. Po funkcji Component.repaint()tworzona jest nowa instancja klasy Thread, która tworzy nowy wątek. Wątek ma za zadanie odczytywać dane ze strumienia odbierającego dane z gniazda - jest to tablica pionków. W przypadku niepowodzenia w łączeniu się z serwerem wypisywany jest stosowny komunikat za pomocą komponentu okna dialogowego klasy JOptionPane. W przypadku kliknięcia na przycisk 'Wyjscie' pojawia się komunikat w postaci okna dialogowego z prośbą potwierdzenia operacji. Gdy użytkownik potwierdzi wolę zamknięcia aplikacji to zamykane są strumienie zapisu oraz odczytu z gniazda sieciowego, zamykane jest gniazdo połączone oraz gniazdo serwerowe.

**Parametry:** domyślny ActionEvent

**Klasa warcaby.serwer.Opcje:**

**Opis:**

Klasa Opcje zawiera zmienne oraz metody odpowiedzialne za prawidłowe wyświetlanie okienka opcji gry oraz ich działanie. W klasie istnieją zmienne będące obiektami klas z pakietu Swing: JFrame, JLabel, JButton oraz JComboBox. Ponadto występuje konstruktor oraz przesłonięta metoda actionPerformed odpowiedzialna za prawidłowe akcje po wciśnięciu danego przycisku lub innego komponentu. Klasa implementuje interfejs ActionListener odpowiadający za określenie akcji po wciśnięciu np. danego przycisku. Ponadto dziedziczy ona po klasie JFrame - odpowiedzialnej za tworzenie okna opcji gry.

**Metody:**

- `public Opcje(JFrame oknoF)`

  **Opis:**
  Konstruktor klasy Opcje przyjmujący jeden argument. W konstruktorze definiowane są: rozmiar okna z opcjami gry, jego pozostałe właściwości. Ponadto zdefiniowano właściwości napisów poszczególnych sekcji na komponencie JLabel tj. font, rozmiar, wymiary, kolor. Zdefiniowano listę wysuwaną umożlwiającą wybór stylu aplikacji. Określono rozmiar, styl. Ponadto zdefiniowano właściwości poszczególnych przycisków na komponencie JButton - ustalono font, napis oraz wymiar. Po zdefiniowaniu każdego z użytych komponentów dodano go do okna z opcjami głównymi metodą Container.add(java.awt.Component).

  **Parametry:**
  **f** - parametr JFrame - główne okno aplikacji

- **`public void actionPerformed(ActionEvent e)`**

  Opis:
  Przesłonięta metoda służąca do określania zachowania aplikacji po kliknięciu na dany komponent przez użytkownika. W metodzie określone są działania dla przycisków oraz listy wysuwanej. Ponadto po wciśnięciu niektórych z przycisków wyświetlane są okna dialogowe za pomocą JOptionPane. W przypadku zmiany wyglądu aplikacji lub pionków lub planszy wywoływana jest metoda klasy Component.repaint(), w celu nałożenia zmian na wygląd poszczególnych komponentów. Metoda jest typu void - nie zwraca żadnej wartości.

  Parametry: domyślny ActionEvent

**Klasa warcaby.serwer.Warcaby:**

**Opis:**

Główna klasa aplikacji, która wywołuje konstruktory pozostałych klas, odpowiedzialna za uruchomienie rozgrywki po stronie Serwera. Klasa ta zawiera w swoim konstruktorze wywołanie konstruktora klasy Plansza, która zawiera całą logikę aplikacji - ruchy, bicia itd.

Metody:

- **`public Warcaby()`**

  Opis:
  Konstruktor klasy Warcaby bezargumentowy, w którym wywoływany jest konstruktor klasy Plansza.Plansza(). Dzięki temu możliwa jest rozgrywka, ponieważ w klasie Plansza zawarta jest cała logika aplikacji - ruchy, bicia itd, po stronie Serwera.

  Parametry: brak

- **`public static void main(String[] args)`**

  Opis:
  Metoda odpowiedzialna za wywołanie konstruktora klasy Warcaby (Warcaby()). Dzięki wywołaniu konstruktora w tej metodzie możliwe jest prawidłowe uruchomienie gry poprzez jej kompilację lub bezpośrednio z poziomu archiwum .jar dla użytkownika Serwera.

  Parametry: domyślna tablica Stringów

## Klasa warcaby.klient.Kafelek:

**Opis:**

Klasa Kafelek zawiera podstawowe zmienne oraz metody obsługi ruchu pionków po szachownicy oraz określa również bicia. Klasa posiada zmienne oraz metody odpowiedzialne za określenie pojedynczego kafelka występującego na szachownicy, dzięki temu można łatwo określić możliwe ruchy dla pionka oraz możliwości bicia. Ponadto w klasie zdefiniowano wystąpienie królówek zgodnie z zasadami gry - warcaby. Zdefiniowano również działanie kafelka informacyjnego, który pokazuje współrzędne kursora użytkownika względem szachownicy - kafelek znajduje się w prawym dolnym rogu głównego okna aplikacji. Klasa Kafelek dziedziczy po klasie JComponent celem stworzenia obiektu kafelka informacyjnego. Ponadto rozszerzona jest ona poprzez interfejs MouseListener celem zdefiniowania akcji myszą - czyli podstawowego kontrolera rozgrywki - poprzez klikanie myszą na odpowiednie pionki możemy je przemieszczać oraz wykonywać bicia.

Metody:

- `public Kafelek(JFrame f, int i, int j)`

  Opis:
  Konstruktor klasy Kafelek inicjalizuje zmienne typu JFrame - odwołanie do okna głównego aplikacji, zmienne określające współrzędne.

  Parametry:
  **f** - parametr JFrame - główne okno aplikacji

  **i** - współrzędna x napisu w kafelku informacyjnym

  **j** - współrzędna y napisu w kafelku informacyjnym

- `protected void ustawNazwe(int i, int j)`

  Opis:
  Metoda ustawNazwe ma za zadanie poprawne wyświetlanie informacji na kafelku informacyjnym. Przyjmuje dwa parametry wywołania. W zależności od współrzędnych informacja jest wyświetlana poprzez dodanie do zmiennej StringBuilder nowych treści poprzez metodę append().

  Parametry:
  **i** - współrzędna x pojedynczego kafelka na szachownicy.

  **j** - współrzędna y pojedynczego kafelka na szachownicy.

- **`public void mouseEntered(MouseEvent e)`**

  Opis:
  Metoda przesłonięta pochodząca z interfejsu MouseListener. Określa ona zachowanie programu podczas poruszania się kursorem myszy po oknie aplikacji, czyli gdy kursor myszy najeżdża na dany komponent - w naszym przypadku komponentem jest okno główne aplikacji - w szczególności szachownica. W naszym przypadku wykorzystywana jest do aktualizacji współrzędnych wyświetlanych na kafelku informacyjnym.
  Parametry: domyślny MouseEvent

- **`public void mouseClicked(MouseEvent e)`**

  Opis:
  Metoda przesłonięta pochodząca z interfejsu MouseListener. Określa ona zachowanie programu podczas, gdy użytkownik kliknie myszką na komponent. Funkcja wykorzystywana do opracowania logiki ruchu pionków oraz bić pionków. Ważne oznaczenia w celu zrozumienia algorytmu ruchu oraz bić pionów:
  *0* - oznaczenie pola pustego
  *1* - oznaczenie pionka czarnego
  *2* - oznaczenie pionka białego
  *3* - oznaczenie wybranego pionka czarnego (wybrany, czyli kliknięty myszką przez gracza)
  *4* - oznaczenie wybranego pionka białego (wybrany, czyli kliknięty myszką przez gracza)
  *5* - oznaczenie pola pustego, na które można wykonać ruch (przesunąć pionka)
  *6* - oznaczenie białej królówki
  *7* - oznaczenie czarnej królówki
  *8* - oznaczenie wybranej białej królówki
  *9* - oznaczenie wybranej czarnej królówki
  Takie rozgraniczenie pozwala na dość czytelne odczytanie algorytmu ruchu oraz bić. Ponadto w metodzie wysyłany jest obiekt tablicy pionków poprzez sieć do drugiego gracza. W metodzie zdefiniowano również warunek zakończenia gry - zakończenie gry sygnalizowane jest specjalnym komunikatem wyświetlanym w oknie dialogowym.

  Parametry: domyślny MouseEvent

- **`public String toString()`**

  Opis:
  Przeciążona metoda toString(), która ma na celu prawidłowe wyświetlanie informacji na kafelku informacyjnym.

  Parametry: brak

- **`public void mouseExited(MouseEvent e)`**

  Opis:
  Metoda przesłonięta pochodząca z interfejsu MouseListener - niewykorzystywana. Jej obecność jest obowiązkowa w celu poprawnego zaimplementowania np. metody mouseClicked(MouseEvent).

  Parametry: domyślny MouseEvent

- **`public void mousePressed(MouseEvent e)`**

  Opis:
  Metoda przesłonięta pochodząca z interfejsu MouseListener - niewykorzystywana. Jej obecność jest obowiązkowa w celu poprawnego zaimplementowania np. metody mouseClicked(MouseEvent).

  Parametry: domyślny MouseEvent

- **`public void mouseReleased(MouseEvent e)`**

  Opis:
  Metoda przesłonięta pochodząca z interfejsu MouseListener - niewykorzystywana. Jej obecność jest obowiązkowa w celu poprawnego zaimplementowania np. metody mouseClicked(MouseEvent).

  Parametry: domyślny MouseEvent

**Klasa warcaby.klient.Plansza:**

**Opis:**

Klasa Plansza zawiera deklaracje oraz definicje wszystkich komponentów tworzących okno główne aplikacji. W klasie zdefiniowano wygląd ogólny okna gry, wygląd planszy, której wygląd można zmienić poprzez opcje gry - Opcje. Ponadto w klasie zadeklarowano zmienne potrzebne do nawiązania połączenia sieciowego z drugim graczem - tu z serwerem. W klasie ustawiane są również początkowe współrzędne tablicy pionków na szachownicy. Ponadto określono działanie przycisków - w szczególności przycisk 'Multiplayer' (zdefiniowano wątek, który odczytuje obiekt ze strumienia umożliwiającego zapis danych do gniazda (w tym przypadku tablica pionków - tablicaPionkow. Klasa implementuje interfejs ActionListener odpowiadający za określenie akcji po wciśnięciu np. danego przycisku. Ponadto implementowany jest interfejs Serializable w celu serializacji obiektu wysyłanego poprzez połączenie sieciowe TCP. Klasa dziedziczy po JFrame - klasie tworzącej główne okno aplikacji.

Metody:

- **`public Plansza()`**

  Opis:
  Konstruktor bezparametrowy klasy Plansza odpowiedzialny za inicjalizację komponentów z biblioteki Swing, zmiennych, tablic. Komponenty definiowane: JButton, JLabel, JFrame - dla tych komponentów ustawiane są wymiary, fonty, kolory. Po zdefiniowaniu każdego z użytych komponentów dodano go do okna z opcjami głównymi metodą Container.add(java.awt.Component). Ponadto definiowane są tablice obiektów tj. tablica obiektów klasy Kafelek, tablica współrzędnych pionków na szachownicy. Zdefiniowano również zmienne typu Color, które określają kolor planszy lub kolor pionków.

  Parametry: brak

- **`public void ustaw()`**

  Opis:
  Metoda bezparametrowa odpowiedzialna za ustawianie pionków na szachownicy. Operacja przeprowadzana jest poprzez użycie pętli for oraz rozgraniczenie dwóch kolorów pionów od siebie - jeden kolor oznaczany jest poprzez 1, a drugi poprzez 2. Pole puste poprzez 0. Po poprawnym ustawieniu zmienna logiczna 'gra' jest ustawiana na wartość TRUE. Metoda jest typu void - nie zwraca żadnej wartości.

  Parametry: brak

- **`public void paint(Graphics g)`**

  Opis:
  Metoda przesłonięta o nazwie paint. Przyjmuje jeden stały parametr typu Graphics. Metoda wykorzystywana jest do rysowania szachownicy oraz pionków. Ponadto w klasie zaadaptowano również rysowanie podświetleń konturów aktywnych kafelków, na które możemy przemieścić gracz swój pionek - są to tak zwane pola aktywne do ruchu dla pionka. W metodzie rysowany jest również kafelek informacyjny umieszczony w prawym dolnym rogu okna głównego aplikacji, który informuje użytkownika na jakim polu szachownicy obecnie znajduje się jego kursor myszy. Rysowanie odbywa się za pomocą komponentu Graphics2D. Wykorzystywane są funkcje takie jak: Graphics.drawRect(int, int, int, int),Graphics.setColor(Color), Graphics.fillRect(int, int, int, int),Graphics.fillOval(int, int, int, int) itp. Metoda jest typu void - nie zwraca żadnej wartości.

  Parametry: domyślny Graphics

- **`public void actionPerformed(ActionEvent e)`**

  Opis:
  Przesłonięta metoda służąca do określania zachowania aplikacji po kliknięciu na dany komponent przez użytkownika. W metodzie tej określono działanie dla poszczególnych przycisków znajdujących się po prawej stronie głównego interfejsu gry. W przypadku kliknięcia na przycisk 'Start' wywoływana jest metoda ustaw(), która ustawia pionki na początkowe pozycje. Następnie wywoływana jest zdefiniowana metoda Component.repaint(), celem wyświetlenia zmian na interfejsie gry. W przypadku kliknięcia na przycisk 'Opcje' tworzona jest nowa instancja klasy Opcje, poprzez wywołanie konstruktora tej klasy. Otworzy się wtedy drugie okno z opcjami gry do wyboru. W przypadku kliknięcia na przycisk 'Multiplayer' tworzone jest gniazdo, które jest automatycznie łączone z podanym portem i adresem. Następnie jeśli łączenie przebiegło poprawnie wyświetlany jest komunikat za pomocą okna dialogowego klasy JOptionPane ze stosowną informacją. Następnie inicjalizowane są strumienie zapisu do gniazda lub odczytu z gniazda. Dalej zmienna logiczna o nazwie 'multi' jest ustawiana na wartość 'true'. Następnie wywoływana jest metoda ustaw(), która ustawia pionki na początkowe pozycje oraz zdefiniowana metoda Component.repaint(), celem wyświetlenia zmian na interfejsie gry. Po funkcji Component.repaint()tworzona jest nowa instancja klasy Thread, która tworzy nowy wątek. Wątek ma za zadanie odczytywać dane ze strumienia odbierającego dane z gniazda - jest to tablica pionków. W przypadku niepowodzenia w łączeniu się z serwerem wypisywany jest stosowny komunikat za pomocą komponentu okna dialogowego klasy JOptionPane. W przypadku kliknięcia na przycisk 'Wyjscie' pojawia się komunikat w postaci okna dialogowego z prośbą potwierdzenia operacji. Gdy użytkownik

potwierdzi wolę zamknięcia aplikacji to zamykane są strumienie zapisu oraz odczytu z gniazda sieciowego, zamykane jest gniazdo połączone oraz gniazdo serwerowe.

Parametry: domyślny ActionEvent

**Klasa warcaby.klient.Opcje:**

**Opis:**

Klasa Opcje zawiera zmienne oraz metody odpowiedzialne za prawidłowe wyświetlanie okienka opcji gry oraz ich działanie. W klasie istnieją zmienne będące obiektami klas z pakietu Swing: JFrame, JLabel, JButton oraz JComboBox. Ponadto występuje konstruktor oraz przesłonięta metoda actionPerformed odpowiedzialna za prawidłowe akcje po wciśnięciu danego przycisku lub innego komponentu. Klasa implementuje interfejs ActionListener odpowiadający za określenie akcji po wciśnięciu np. danego przycisku. Ponadto dziedziczy ona po klasie JFrame - odpowiedzialnej za tworzenie okna opcji gry.

Metody:

- `public Opcje(JFrame oknoF)`

  Opis:
  Konstruktor klasy Opcje przyjmujący jeden argument. W konstruktorze definiowane są: rozmiar okna z opcjami gry, jego pozostałe właściwości. Ponadto zdefiniowano właściwości napisów poszczególnych sekcji na komponencie JLabel tj. font, rozmiar, wymiary, kolor. Zdefiniowano listę wysuwaną umożlwiającą wybór stylu aplikacji. Określono rozmiar, styl. Ponadto zdefiniowano właściwości poszczególnych przycisków na komponencie JButton - ustalono font, napis oraz wymiar. Po zdefiniowaniu każdego z użytych komponentów dodano go do okna z opcjami głównymi metodą Container.add(java.awt.Component).

  Parametry:
  **f** - parametr JFrame - główne okno aplikacji

- `public void actionPerformed(ActionEvent e)`

  Opis:
  Przesłonięta metoda służąca do określania zachowania aplikacji po kliknięciu na dany komponent przez użytkownika. W metodzie określone są działania dla przycisków oraz listy wysuwanej. Ponadto po wciśnięciu niektórych z przycisków wyświetlane są okna dialogowe za pomocą JOptionPane. W przypadku zmiany wyglądu aplikacji lub pionków lub planszy wywoływana jest metoda klasy Component.repaint(), w celu nałożenia zmian na wygląd poszczególnych komponentów. Metoda jest typu void - nie zwraca żadnej wartości.

  Parametry: domyślny ActionEvent

**Klasa warcaby.klient.Warcaby:**

**Opis:**

Główna klasa aplikacji, która wywołuje konstruktory pozostałych klas, odpowiedzialna za uruchomienie rozgrywki po stronie Klienta. Klasa ta zawiera w swoim konstruktorze wywołanie konstruktora klasy Plansza, która zawiera całą logikę aplikacji - ruchy, bicia itd.

Metody:

- `public Warcaby()`

  Opis:
  Konstruktor klasy Warcaby bezargumentowy, w którym wywoływany jest konstruktor klasy Plansza.Plansza(). Dzięki temu możliwa jest rozgrywka, ponieważ w klasie Plansza zawarta jest cała logika aplikacji - ruchy, bicia itd, po stronie Klienta.

  Parametry: brak

- `public static void main(String[] args)`

  Opis:
  Metoda odpowiedzialna za wywołanie konstruktora klasy Warcaby (Warcaby()). Dzięki wywołaniu konstruktora w tej metodzie możliwe jest prawidłowe uruchomienie gry poprzez jej kompilację lub bezpośrednio z poziomu archiwum .jar dla użytkownika Klienta.

  Parametry: domyślna tablica Stringów

# 6. PODSUMOWANIE

## a) Realizacja celów projektu:

Celem projektu było napisanie gry Warcaby w języku Java. Zdecydowaliśmy się na bibliotekę graficzną Swing celem napisania graficznego interfejsu gry. Ponadto naszym głównym celem było napisanie gry sieciowej, co również nam się udało zrealizować. Zastosowaliśmy mechanizm wątków wykorzystywany podczas odczytywania danych wysyłanych przez sieć. Cały projekt zgodnie z zamysłem został napisany zgodnie z zasadami paradygmatu obiektowego. Ponadto mieliśmy zrealizować projekt z zastosowaniem narzędzia Gradle – pomogło nam to w szybkim imporcie projektu do różnych środowisk takich jak: Eclipse, IntelliJ czy NetBeans. Dzięki wymaganiom projektowym mogliśmy zaznajomić się z obsługą zdalnego repozytorium kodu. Zdecydowaliśmy się na usługę GitHub ze względu na jej największe rozpowszechnienie. Dokumentacja została wykonana narzędziem JavaDoc zgodnie z wymogiem. Projekt został napisany przez cały zespół, każdy z członków zespołu ma równy wkład w stworzenie projektu. Reasumując wszystkie postawione cele zostały zrealizowane.

### b) Napotkanie problemy:

Jednym z napotkanych problemów był mechanizm bicia dla pionków, lecz po dłuższym debugowaniu doszliśmy do rozwiązania. Ponadto system tworzenia damek również pochłonął większą ilość czasu, lecz tak samo jak w przypadku pionków doszliśmy do rozwiązania.

### c) Kierunki rozwoju:

Kierunkiem rozwoju jest adaptacja aplikacji do nowszej biblioteki graficznej – JavaFX. Jest to wygodna biblioteka przez tworzenie plików FXML, które zawierają wszelkie dane na temat wyglądu aplikacji co czyni kod jeszcze czytelniejszym. W niektórych miejscach naszego projektu można użyć mechanizmu lambd celem uzyskania czytelnego kodu. Ponadto kierunkiem rozwoju może być przeniesienie gry do zaawansowanej biblioteki graficznej tj. LibGDX.

# 7. BIBLIOGRAFIA, ŹRÓDŁA

## a) Bibliografia:


1. Cay S.Horstmann, *Java Podstawy*, Wyd. X, Gliwice, Wydawnictwo Helion, 2016.
2. Cay S.Horstmann, *Java 8. Przewodnik doświadczonego programisty*, Wyd. X, Gliwice, Wydawnictwo Helion, 2016.


## b) Źródła:


1. http://docs.oracle.com/javase/8/docs/api/index.html
2. https://javastart.pl
3. https://docs.oracle.com/javase/7/docs/api/javax/swing/package-summary.html
4. http://naukajavy.pl/