Data złożenia: 19.06.2015

Skład zespołu:
- Jakub Durzyński
- Krzysztof Przywarty
- Jakub Podgórski
- Marcin Kwaśnik

I21-Z4

Sprawozdanie z laboratorium sztucznej inteligencji

Informatyka, sem. VI

Teeko

Star3 **systems®**

1. Opis zadania

Projekt zakłada zaimplementowanie dwuosobowej gry *Teeko*, gdzie jednym z graczy będzie zaprogramowana w *Prologu* maszyna grająca.
Teeko jest to popularna w USA w latach 50. i 60. gra logiczna. Celem gracza jest ustawienie swoich czterech pionków w jednej ze zwycięskich konfiguracji: w linii poziomej, linii pionowej, linii przekątnej lub w kwadracie przylegających pól (2x2), zanim dokona tego przeciwnik.
Gra podzielona jest na dwa etapy. W pierwszym gracze ustawiają na zmianę po jednym pionku na planszy w dowolnych miejscach. W drugim etapie gracze przesuwają na zmianę po jednym swoim pionku na przyległe do niego pola.
W projekcie głównie położono nacisk na moduł sztucznej inteligencji w którym dla uzyskania zadowalających efektów wykorzystano algorytm mini-maks wraz z alfa-beta cięciami. Cały ten moduł zaprogramowany został w języku *Prolog* i połączony z wersją gry zaprogramowaną w języku *Java* przy użyciu biblioteki *JPL*.
Dla łatwej obsługi aplikacji stworzony został, z wykorzystaniem biblioteki *libGDX*, graficzny interfejs użytkownika obsługiwany za pomocą myszy.
Głównym elementem całej aplikacji jest moduł łączący wszystkie elementy który przetwarza otrzymane z GUI dane wejściowe w postaci współrzędnych pionka na tablicy, przesyła je do reszty modułów i po otrzymaniu odpowiedzi przekazuje nowe wartości (dotyczące pionka/ruchu sztucznej inteligencji) z powrotem do interfejsu, który wyświetla użytkownikowi zmiany na ekranie.

2. Założenia realizacyjne

Aplikacja składa się z graficznego interfejsu użytkownika, dwóch modułów głównych (modułu obsługi GUI, a zarazem obsługi gry, napisanego w języku *Java* i modułu sztucznej inteligencji napisanego w języku *Prolog*) oraz dwóch modułów stanowiących połączenie dla modułów powyższych. Dodatkowo zaimplementowane zostały nieduże klasy zawierające informacje o pionkach i stanach gry.

Moduł główny gry (*Star3Teeko.java*)
Moduł, którego głównym zadaniem jest obsługa samej gry i czuwanie nad jej prawidłowym przebiegiem. Podstawowym elementem jest tu metoda *Update* w której odbywają się wszelkie akcje związane z grą, a także wyświetlanie stanu gry w GUI.
Dane wejściowe: brak, metoda uruchamiana po inicjalizacji wszystkich innych niezbędnych elementów zawartych w pliku.
Dane wyjściowe: stan gry zobrazowany w graficznym interfejsie użytkownika.
Działanie:

1) Inicjalizacja niezbędnych elementów.
2) Uruchomienie metody *Update*.
3) Sprawdzenie stanu gry (ruch gracza lub ruch SI).
4) Jeśli ruch gracza:
    a) sprawdzenie ilości ruchów,
    b) jeśli mniej niż 4:
        i) pobranie współrzędnych,
        ii) konwersja współrzędnych,
        iii) ustawienie pionka na tablicy,

iv) zmiana stanu gry,
c) jeśli więcej niż 4:
i) pobranie danych o pionku i jego współrzędnych,
ii) konwersja współrzędnych,
iii) ruch pionka na nowe współrzędne,
iv) zmiana stanu gry,
d) sprawdzenie wygranej,
5) Jeśli ruch SI:
a) sprawdzenie ilości ruchów,
b) jeśli mniej niż 4:
i) wywołanie metody ustawiania pionka dla SI,
ii) zmiana stanu gry,
c) jeśli więcej niż 4:
i) wywołanie metody przesunięcia pionka dla SI,
ii) zmiana stanu gry,
d) sprawdzenie wygranej.

Moduł sztucznej inteligencji (*teeko.pl*)
Moduł symulujący sztuczną inteligencję będącą przeciwnikiem w grze, a także przechowujący dane na tematu ruchów człowieka, które niezbędne są do wykonania kolejnych kroków podejmowanych przez komputer. Podzielony na trzy części: zawierającą zasady działania algorytmu sztucznej inteligencji, odpowiedzialną za początkowe rozstawienie pionków na planszy oraz przemieszczającą pionki po ich rozstawieniu, gdy w poprzednim etapie żadne z graczy nie został zwycięzcą. Zawiera podprogramy implementujące działanie algorytmu mini-maksowego wraz z alfa-beta cięciami, a także metody samej gry pozwalające na przekazywanie stanu rozgrywki tak, aby całość była zrozumiała dla SI.
Dane wejściowe: różne w zależności od wywoływanego podprogramu.
Dane wyjściowe: wywołanie podprogramów odpowiedzialnych za moduł sztucznej inteligencji, rozstawienie pionków, przesuwanie pionków człowieka oraz komputera i wykonanie przez nie odpowiednich działań mających na celu zobrazować stan rozgrywki w Prologu oraz mających na celu zwrócić niezbędne wartości potrzebne do zobrazowania gry w interfejsie graficznym.
Działanie:
1) Wywołanie podprogramu znajdującego się w pliku teeko.pl.
2) Wykonanie odpowiedniego podprogramu, na który mogą składać się kolejne zapytania oraz ewentualne zwrócenie poszczególnych wartości potrzebnych do zobrazowania gry po stronie interfejsu graficznego.

Moduł łączący od strony *Javy* (*JavaService.java*)
Moduł stanowiący połączenie między modułem głównym gry, a drugim modułem łączącym. Zawiera metody wywoływane w przypadku ruchów pionków zarówno dla człowieka jak i sztucznej inteligencji.
Dane wejściowe: zmienna typu *HashMap<Integer, String>* obrazująca pole gry, zmienna typu *PrologService* będąca instancją drugiego modułu łączącego, zmienne liczbowe odzwierciedlające współrzędne ustawienia pionka lub jego ruchu.
Dane wyjściowe: uruchomienie metody z drugiego modułu łączącego.
Działanie:
1) Konwersja zmiennej liczbowej (w przypadku metod dotyczących ruchu człowieka).

2) Wywołanie metody z drugiego modułu łączącego.
3) Zwrócenie danych o ruchu (w przypadku metod dotyczących ruchu sztucznej inteligencji).

Moduł łączący od strony *Prologu* (*PrologService.java*)
Moduł stanowiący połączenie między pierwszym modułem łączącym, a modułem sztucznej inteligencji napisanej w języku *Prolog*. Zawiera metody wywoływane w przypadku ruchów pionków zarówno dla człowieka jak i sztucznej inteligencji. Wywołuje metody z pliku *teeko.pl* w celu uzyskania ruchu dla sztucznej inteligencji. Najważniejszą zawartą tu metodą jest *start* która wczytuje plik *teeko.pl* i pozwala wywoływać jego metody z poziomu aplikacji.
Dane wejściowe: różne w zależności od wywoływanej metody.
Dane wyjściowe: wywołanie metod z pliku *teeko.pl*.
Działanie:
1) Utworzenie zapytania w języku *Prolog*.
2) Wywołanie utworzonego zapytania.
3) Zwrócenie uzyskanego wyniku.
Ważniejsze użyte technologie i algorytmy:

- ***Prolog* w wersji 7.2.0** – jeden z najpopularniejszych języków programowania logicznego, który powstał jako język programowania służący do automatycznej analizy języków naturalnych, jest jednak językiem ogólnego zastosowania, szczególnie dobrze sprawdzającym się w programach związanych ze sztuczną inteligencją,
- ***Java* w wersji 1.8.0_45** – obiektowy język programowania, służący do tworzenia programów źródłowych kompilowanych do kodu bajtowego, czyli postaci wykonywanej przez maszynę wirtualną,
- ***Eclipse*** – rozbudowane środowisko programistyczne stworzone przez firmę IBM i przekazane następnie społeczności Open Source, którego główna zaletą jest obsługa wtyczek rozszerzających jego możliwości o obsługę wielu języków tworzenia kodu,
- ***libGDX* w wersji 1.6.2** – wieloplatformowe narzędzie do tworzenia gier i wizualizacji, bazuje na języku Java,
- ***JPL* w wersji 2.0.2** – biblioteka dla programów w języku *Java* zawierająca klasy umożliwiające prostą komunikację pomiędzy *Javą* a *Prologiem*,
- Algorytm mini-maks – metoda minimalizowania maksymalnych możliwych strat (alternatywnie można go traktować jako maksymalizację minimalnego zysku), która wywodzi się z teorii gry o sumie zerowej, obejmującej oba przypadki, zarówno ten, gdzie gracze wykonują ruchy naprzemiennie, jak i ten, gdzie wykonują ruchy jednocześnie,
- Algorytm alfa-beta cięć – algorytm przeszukujący, redukujący liczbę węzłów, które muszą być rozwiązywane w drzewach przeszukujących przez algorytm mini-maks, często wykorzystywany w grach dwuosobowych,

3. Podział prac

Jakub Durzyński	Zapoznanie się z algorytmami mini-maks oraz algorytmem alfa-beta cięć. Zapoznanie się z interfejsem *GNU Prologu* w *Javie*. Implementacja algorytmu mini-maxowego. Implementacja funkcji heurystycznej. Implementacja funkcji cięć drzewa do wybranej głębokości. Łączenie części projektu napisanego w *Prologu* z *Javą* przy wykorzystaniu biblioteki *JPL*. Naprawianie błędów wykrytych podczas testowania aplikacji.
Jakub Podgórski	Zapoznanie się z algorytmami mini-maks oraz algorytmem alfa-beta cięć. Zapoznanie się z interfejsem *GNU Prologu* w *Javie*. Implementacja algorytmu mini-maxowego. Implementacja funkcji heurystycznej. Implementacja funkcji cięć drzewa do wybranej głębokości. Łączenie części projektu napisanego w *Prologu* z *Javą* przy wykorzystaniu biblioteki *JPL*. Naprawianie błędów wykrytych podczas testowania aplikacji.
Krzysztof Przywarty	Zapoznanie się z *libGDX*. Odwzorowanie planszy gry wraz z pionkami. Implementacja graficznego interfejsu użytkownika. Implementacja modułu głównego gry. Naprawianie błędów wykrytych podczas testowania aplikacji.
Marcin Kwaśnik	Zapoznanie się z *libGDX*. Implementacja graficznego interfejsu użytkownika. Implementacja modułu głównego gry. Łączenie części projektu napisanego w *Prologu* z *Javą* przy wykorzystaniu biblioteki *JPL*. Naprawianie błędów wykrytych podczas testowania aplikacji.

4. Opis implementacji

Moduł sztucznej inteligencji (*teeko.pl*)

- `przesunPionek(PosX, PosY, PosXO, PosYO, TOX, TOY, M)`

  Podprogram dotyczący przesunięcia pionka przez sztuczną inteligencję w zadane miejsce.
  Dane wejściowe: zmienne PosX, PosY, PosXO, PosYO służące do zwrócenia informacji na temat zmiany położenia wybranego pionka przez sztuczną inteligencję, zmienne TOX, TOY przechowujące informację o nowym położeniu pionka oraz zmienna M zawierająca dane przesuwanego pionka.
  Dane wyjściowe: zwrócenie informacji na temat przesunięcia pionka oraz wykonanie przesunięcia na zadaną pozycję.
  Działanie:
  1) Wywołanie podprogramu odpowiedzialnego za obliczenie ruchu dla pionka.
  2) Przypisanie nowych współrzędnych dla pionka w programie.
  3) Usunięcie starych współrzędnych dla pionka w programie.

4) Przypisanie zmiennym PosX, PosY, PosXO, PosYO obecnych oraz starych koordynatów pionka w celu zwrócenia informacji wykorzystywanej przez interfejs graficzny.

- `ustawPionek(PosX, PosY, X, Y, M)`

  Podprogram dotyczący ustawienia pionka przez sztuczną inteligencję w zadane miejsce.
  Dane wejściowe: zmienne PosX, PosY służące do zwrócenia informacji na temat położenia wybranego pionka przez sztuczną inteligencję, zmienne X, Y przechowujące informację o rozstawieniu pionka na planszy oraz zmienna M zawierająca dane rozstawianego pionka.
  Dane wyjściowe: zwrócenie informacji na temat rozstawienia pionka oraz wykonanie przesunięcia na zadaną pozycję.
  Działanie:
  1) Wywołanie podprogramu odpowiedzialnego za obliczenie położenia dla pionka.
  2) Przypisanie nowych współrzędnych dla pionka w programie.
  3) Przypisanie zmiennym PosX, PosY obecnych koordynatów pionka w celu zwrócenia informacji wykorzystywanej przez interfejs graficzny.

- `rozstawienieCzlowiek(Pos, Gracz)`

  Podprogram wywoływany przez moduł łączący od strony Prologu służący do ustawienia pionka człowieka.
  Dane wejściowe: zmienna Pos przekazująca informację na temat pionka ustawianego przez człowieka oraz zmienna Gracz przechowująca informację o graczu, dla którego pionek jest ustawiany.
  Dane wyjściowe: brak, wywołanie kolejnych podprogramów odpowiedzialnych za przypisanie lokacji pionka.
  Działanie:
  1) Wywołanie podprogramu ostatniPionek informującego o ostatnim nieustawionym pionku.
  2) Wywołanie podprogramu wprowadzPozycje ustanawiającego pozycję pionka na podstawie zmiennej Pos.

- `rozstawienieAI(PosX, PosY, Gracz)`

  Podprogram wywoływany przez moduł łączący od strony Prologu służący do ustawienia pionka sztucznej inteligencji.
  Dane wejściowe: zmienne PosX, PosY potrzebne do zwrócenia późniejszych informacji na temat położenia pionka sztucznej inteligencji oraz zmienna Gracz przechowująca informację o graczu, dla którego pionek jest ustawiany.
  Dane wyjściowe: brak, wywołanie kolejnych podprogramów odpowiedzialnych za przypisanie lokacji pionka.
  Działanie:
  1) Wywołanie podprogramu najkorzystniejszyRuch ustalającego najkorzystniejszą pozycję ustawienia pionka sztucznej inteligencji.

2) Wywołanie podprogramu ustawPionek ustanawiającego pozycję pionka sztucznej inteligencji.

- `wykonajRuchCzlowiek(Gracz, Move)`

  Podprogram wywoływany przez moduł łączący od strony Prologu służący do przesunięcia pionka człowieka.
  Dane wejściowe: zmienna Gracz przechowująca informację o graczu, dla którego pionek jest ustawiany oraz zmienna Move zawierająca informację przesunięciu pionka w zadaną pozycję.
  Dane wyjściowe: brak, wywołanie kolejnego podprogramu odpowiedzialnego za zmianę lokacji pionka.
  Działanie:
  1) Wywołanie podprogramu sprawdzPoprawnoscRuchu sprawdzającego poprawność wykonywanego ruchu i w dalszych etapach zmieniającego położenie wskazanego pionka.

- `wykonajRuchAI(PosX, PosY, PosXO, PosYO)`

  Podprogram wywoływany przez moduł łączący od strony Prologu służący do przesunięcia pionka sztucznej inteligencji.
  Dane wejściowe: zmienne PosX, PosY, PosXO, PosYO służące w późniejszych etapach do zwrócenia starych koordynatów pionka oraz koordynatów, w które został on przesunięty po wykonaniu ruchu przez sztuczną inteligencję.
  Dane wyjściowe: brak, wywołanie kolejnego podprogramu odpowiedzialnego za zmianę lokacji pionka.
  Działanie:
  1) Wywołanie podprogramu najkorzystniejszyRuch ustalającego najkorzystniejszą pozycję przesunięcia pionka sztucznej inteligencji.
  2) Wywołanie podprogramu przesunPionek ustanawiającego nową pozycję pionka sztucznej inteligencji.

Moduł łączący od strony *Javy* (*JavaService.java*)

- `public void putPawnHuman(HashMap<Integer, String> playboard, PrologService ps, int number, int xy)`

  Metoda dotycząca ustawienia pionka przez człowieka.
  Dane wejściowe: zmienna typu *HashMap<Integer, String>* obrazująca pole gry, zmienna typu *PrologService* będąca instancją drugiego modułu łączącego, zmienne liczbowe odzwierciedlające współrzędne ustawienia pionka i numer pionka.
  Dane wyjściowe: brak, wywoływana jest metoda z drugiego modułu łączącego.
  Działanie:
  1) Konwersja zmiennej liczbowej współrzędnych na dane zrozumiałe dla metody *Prologowej*.
  2) Wywołanie metody `setCoordHuman` z klasy *PrologService* z uzyskanymi w wyniku konwersji argumentami.

3) Wywołanie metody `put` dla planszy gry z argumentami współrzędnych i numerem pionka.

- `public int putPawnAI(HashMap<Integer, String> playboard, PrologService ps, int number)`

  Metoda dotycząca ustawienia pionka przez sztuczną inteligencję.
  Dane wejściowe: zmienna typu *HashMap<Integer, String>* obrazująca pole gry, zmienna typu *PrologService* będąca instancją drugiego modułu łączącego, zmienna liczbowa odzwierciedlająca numer pionka.
  Dane wyjściowe: zmienna liczbowa zawierająca współrzędne ustawienia pionka SI.
  Działanie:
  1) Wywołanie metody `setCoordAI` i przypisanie uzyskanej wartości współrzędnych do zmiennej.
  2) Wywołanie metody `put` dla planszy gry z argumentami współrzędnych i numerem pionka.
  3) Zwrócenie zmiennej ze współrzędnymi ustawienia pionka SI.

- `public void movePawnHuman(HashMap<Integer, String> playboard, PrologService ps, int xyold, int xynew)`

  Metoda dotycząca ruchu pionka przez człowieka.
  Dane wejściowe: zmienna typu *HashMap<Integer, String>* obrazująca pole gry, zmienna typu *PrologService* będąca instancją drugiego modułu łączącego, zmienne liczbowe odzwierciedlające stare i nowe współrzędne wybranego pionka.
  Dane wyjściowe: brak, wywoływana jest metoda z drugiego modułu łączącego.
  Działanie:
  1) Konwersja zmiennej liczbowej współrzędnych na dane zrozumiałe dla metody *Prologowej*.
  2) Ustalenie kierunku ruchu na podstawie współrzędnych starych i nowych.
  3) Wywołanie metody `get` dla planszy gry w celu uzyskania numeru wybranego pionka.
  4) Wywołanie metody `put` dla planszy gry z argumentami nowych współrzędnych i numerem pionka w celu przesunięcia pionka.
  5) Wywołanie metody `moveHuman` z klasy *PrologService* z numerem pionka i kierunkiem ruchu.
  6) Wywołanie metody `put` dla planszy gry z argumentami starych współrzędnych i pustą zmienną tekstową w celu wyczyszczenia pola o starych współrzędnych.

- `public int[] movePawnAI(HashMap<Integer, String> playboard, PrologService ps)`

  Metoda dotycząca ruchu pionka przez sztuczną inteligencję.
  Dane wejściowe: zmienna typu *HashMap<Integer, String>* obrazująca pole gry, zmienna typu *PrologService* będąca instancją drugiego modułu łączącego.
  Dane wyjściowe: tablica zmiennych liczbowych zawierająca stare i nowe współrzędne ustawienia pionka SI.
  Działanie:

1) Stworzenie tablicy dwuargumentowej zmiennych liczbowych.
2) Wywołanie metody `moveAI` i przypisanie uzyskanych wartości współrzędnych do utworzonej wcześniej tablicy.
3) Zwrócenie tablicy zmiennych ze współrzędnymi ustawienia pionka SI.

Moduł łączący od strony *Prologu* (*PrologService.java*)

- `public void start()`

  Metoda wczytująca plik *Prologu*.
  Dane wejściowe: brak, metoda wywoływana przy starcie aplikacji.
  Dane wyjściowe: brak, metoda wczytuje plik *Prologu* i wypisuje na konsoli informację o wykonaniu czynności.
  Działanie:
  1) Wczytanie pliku *Prologu*.
  2) Wypisanie na konsoli informacji o wykonaniu czynności.

- `public void setCoord(String player)`

  Metoda zapisująca współrzędne pionka.
  Dane wejściowe: zmienna typu *String* zawierająca informację o typie gracza.
  Dane wyjściowe: brak, wywoływana jest metoda zapisująca współrzędne pionka.
  Działanie:
  1) Utworzenie zapytania *prologowego*.
  2) Wywołanie utworzonego zapytania.

- `public void setCoordHuman(int position)`

  Metoda zapisująca współrzędne pionka człowieka.
  Dane wejściowe: zmienna liczbowa zawierająca informację o współrzędnych.
  Dane wyjściowe: brak, wywoływana jest metoda zapisująca współrzędne pionka.
  Działanie:
  1) Utworzenie zapytania *prologowego*.
  2) Wywołanie utworzonego zapytania.

- `public void setCoordLast()`

  Metoda pomocnicza zapisująca ostatnie współrzędne pierwszej fazy gry.
  Dane wejściowe: brak.
  Dane wyjściowe: brak, wywoływana jest metoda zapisująca współrzędne pionka.
  Działanie:
  1) Utworzenie zapytania *prologowego*.
  2) Wywołanie utworzonego zapytania.

- `public int setCoordAI()`

  Metoda ustawiająca współrzędne pionka SI.
  Dane wejściowe: brak.

Dane wyjściowe: zmienna liczbowa zawierająca współrzędne pionka.
Działanie:
1) Utworzenie zmiennych liczbowych do przechowywania współrzędnych.
2) Utworzenie zapytania *prologowego*.
3) Wywołanie utworzonego zapytania.
4) Zwrócenie uzyskanego wyniku.

- `public void move(String player)`

Metoda zapisująca ruch pionka.
Dane wejściowe: zmienna typu string zawierająca informację o typie gracza.
Dane wyjściowe: brak, wywoływana jest metoda zapisująca ruch pionka.
Działanie:
Utworzenie zapytania *prologowego*.
1) Wywołanie utworzonego zapytania.

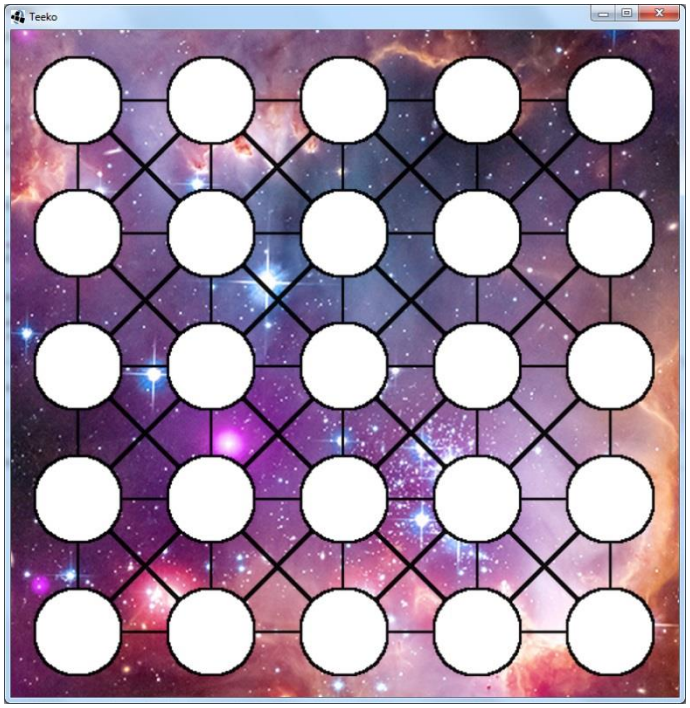
- `public void moveHuman(String move)`

Metoda zapisująca ruch pionka człowieka.
Dane wejściowe: zmienna typu string zawierająca informację kierunku ruchu.
Dane wyjściowe: brak, wywoływana jest metoda zapisująca ruch pionka.
Działanie:
1) Utworzenie zapytania *prologowego*.
2) Wywołanie utworzonego zapytania.

- `public int[] moveAI(HashMap<java.lang.Integer, String> playboard)`

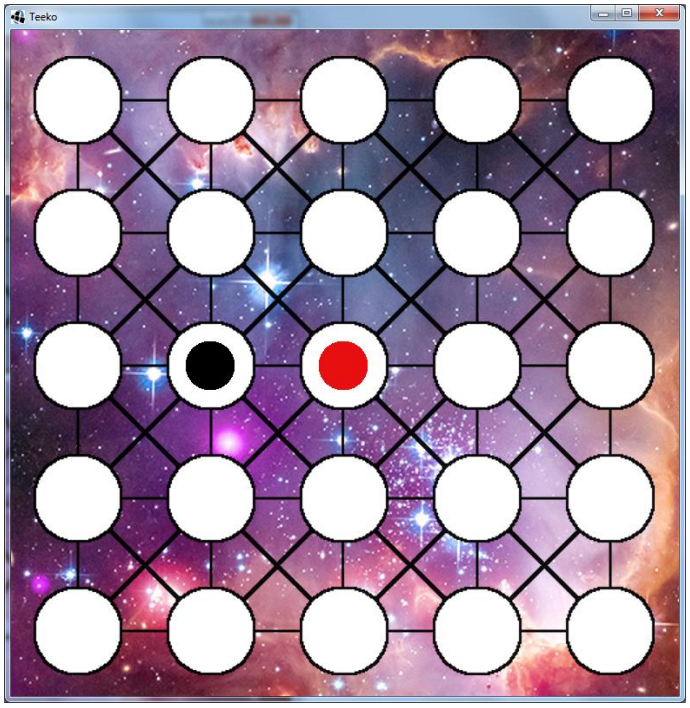
Metoda ustawiająca ruch pionka SI.
Dane wejściowe: zmienna typu *HashMap<Integer, String>* obrazująca pole gry.
Dane wyjściowe: tablica zmiennych liczbowych zawierająca stare i nowe współrzędne ustawienia pionka SI.
Działanie:
1) Utworzenie zmiennych liczbowych do przechowywania współrzędnych.
2) Utworzenie zmiennej tekstowej do przechowywania numeru pionka.
3) Utworzenie zapytania *prologowego*.
4) Wywołanie utworzonego zapytania.
5) Pobranie numeru pionka.
6) Wywołanie metody `put` dla planszy gry z argumentami nowych współrzędnych i numerem pionka w celu przesunięcia pionka.
7) Wywołanie metody `put` dla planszy gry z argumentami starych współrzędnych i pustą zmienną tekstową w celu wyczyszczenia pola o starych współrzędnych.
8) Utworzenie dwuargumentowej tablicy zmiennych liczbowych.
9) Przypisanie wartości zmiennych ze współrzędnymi do tablicy.
10) Zwrócenie tablicy.

5. Użytkowanie i testowanie systemu



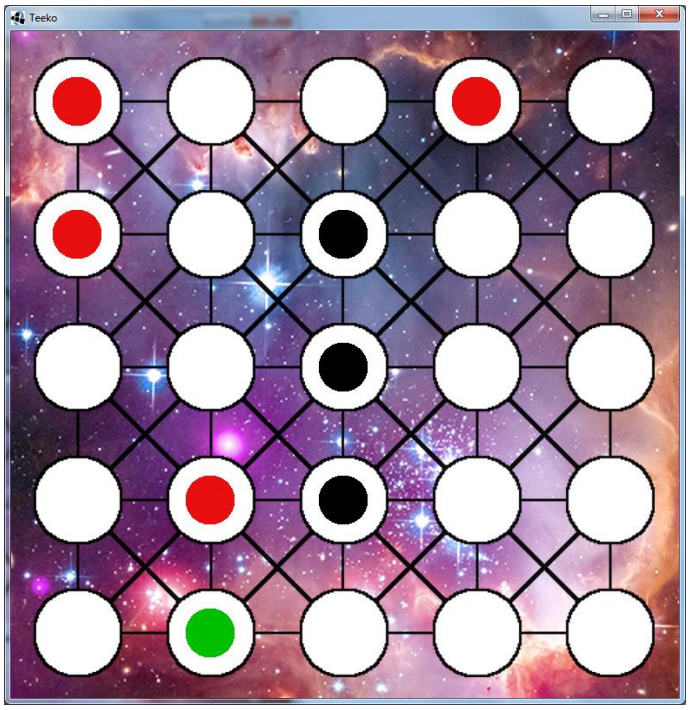
Rys. 5.1 Pusta plansza rozgrywki

Po uruchomieniu aplikacji wyświetlone zostaje od razu okno gry. Gracz może wykonać swój ruch. Pionek można umieścić klikając w dowolnie wybranym białym polu lewym przyciskiem myszy.



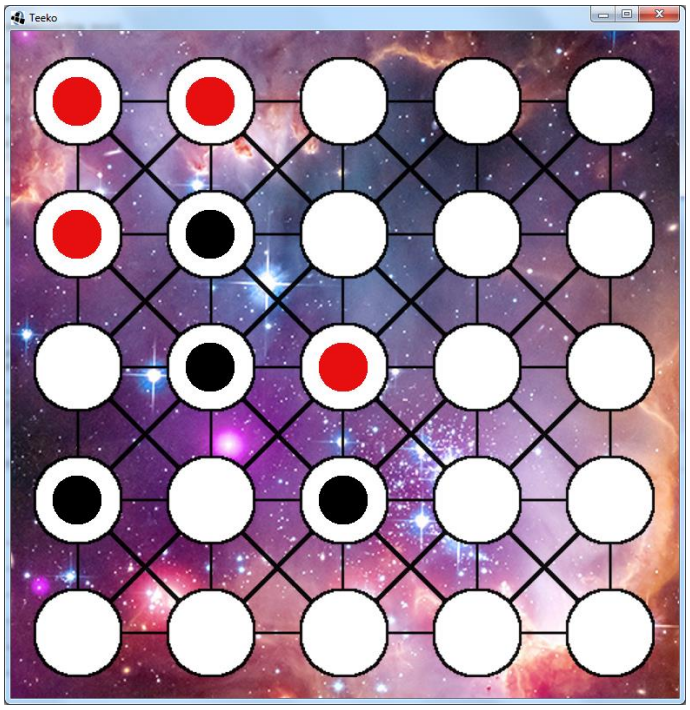
Rys. 5.2 Plansza z ustawionymi dwoma pionkami

Plansza gry po wykonaniu ruchu przez człowieka (pionek czarny) oraz sztuczną inteligencję (pionek czerwony). Gracze mają do dyspozycji 4 pionki które mogą ustawić w dowolnym białym obszarze na planszy.



Rys. 5.3 Plansza ze wszystkimi pionkami, zielony jako czarny pionek zaznaczony przez człowieka

Plansza gry po rozmieszczeniu przez obu graczy czterech pionków. Gracze mogą teraz jedynie przesuwać jeden z wybranych pionków na dowolne puste przylegające pole zaznaczając swój postawiony pionek lewym przyciskiem myszy i klikając tym samym przyciskiem na dowolne przylegające białe pole. Na obrazie widać pionek zielony czyli obecnie wybrany przez człowieka. Ten pionek może zostać przesunięty lub odznaczony jeśli gracz zdecyduje się jednak przesunąć inny pionek.

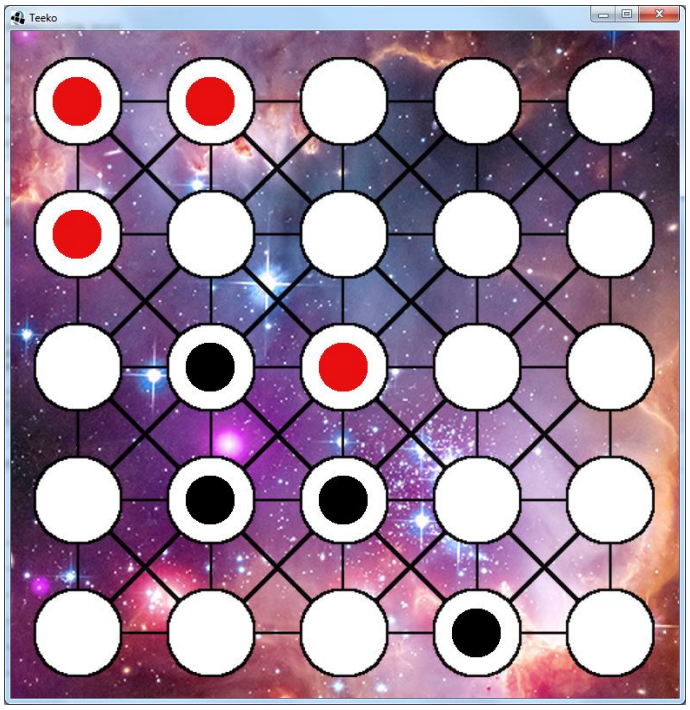


Rys. 5.4 Plansza z początkowym ustawieniem dla wygranej przez człowieka partii

Dla powyższego ustawienia wejściowego jesteśmy w stanie odpowiednio przesunąć pionki tak aby uzyskać ułożenie dające nam zwycięstwo (rys. 5.3 ukazuje tą sytuację jeden ruch przed wygraną człowieka).

Rys. 5.5 Ekran końcowy gry przy partii wygranej przez człowieka

Ekran wyświetlany w przypadku zwycięstwa gracza z czarnymi pionkami (tutaj człowieka). Wynik uzyskany z kombinacji wejściowej pokazanej na rys. 5.4.



Rys. 5.6 Plansza z początkowym ustawieniem dla przegranej przez człowieka partii

W przypadku wejściowego ustawienia jak na powyższym obrazku człowiek nie jest już w stanie wygrać gdyż sztucznej inteligencji brakuje jednego ruchu do zakończenia gry.

Rys. 5.7 Ekran końcowy gry przy partii przegranej przez człowieka

Po wykonaniu przez człowieka dowolnego ruchu sztuczna inteligencja wykonała zwycięski ruch, kończąc tym samym grę. Pokazuje się ekran informujący o zwycięstwie gracza z czerwonymi pionkami (w tym przypadku SI).

6. Tekst programu

DesktopLauncher.java

```
package star3.project.desktop;

import com.badlogic.gdx.backends.lwjgl.LwjglApplication;
import com.badlogic.gdx.backends.lwjgl.LwjglApplicationConfiguration;
import star3.project.Star3Teeko;

public class DesktopLauncher {
	public static void main (String[] arg) {
		LwjglApplicationConfiguration config = new
LwjglApplicationConfiguration();

		config.title = "Teeko";
		config.width = 768;
		config.height = 768;

		new LwjglApplication(new Star3Teeko(), config);
	}
}
```

JavaService.java

```
package star3.project.service;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.HashMap;
import java.util.Map.Entry;

public class JavaService
{
	public JavaService()
	{

	}

	public void putPawnHuman(HashMap<Integer, String> playboard,
PrologService ps, int number, int xy) throws IOException
	{
		String position = Integer.toString(xy);

		try
		{
			String[] positionArray = position.split("");
			int[] xyArray = new int[positionArray.length];

			for (int i = 0; i < positionArray.length; i++)
			{
				xyArray[i] = Integer.parseInt(positionArray[i]);
			}
			String[] playerTypeArray = playboard.get(xyArray[0] +
(xyArray[1])).split("");

				//X + (Y-1)*5
```

```
                        ps.setCoordHuman(Integer.parseInt(position));
                        playboard.put((xyArray[0] + ((xyArray[1]-1)*5)),
"B" + number);

                }
                catch (NumberFormatException e)
                {
                        System.out.println("Wprowadzona wartoďż˝ďż˝ nie jest
liczbďż˝. Sprďż˝buj ponownie wprowadziďż˝ poprawnďż˝ wartoďż˝ďż˝.");
                        putPawnHuman(playboard, ps, number, xy);
                }
        }

        public int putPawnAI(HashMap<Integer, String> playboard,
PrologService ps, int number)
        {
                int xy = ps.setCoordAI();
                playboard.put(xy, "R" + number);
                return xy;
        }

        public void movePawnHuman(HashMap<Integer, String> playboard,
PrologService ps, int xyold, int xynew) throws IOException
        {
                String positionold = Integer.toString(xyold);

                String[] positionoldArray = positionold.split("");
                int[] xyoldArray = new int[positionoldArray.length];

                for (int i = 0; i < positionoldArray.length; i++)
                {
                        xyoldArray[i] = Integer.parseInt(positionoldArray[i]);
                }

                String positionnew = Integer.toString(xynew);

                String[] positionnewArray = positionnew.split("");
                int[] xynewArray = new int[positionnewArray.length];

                for (int i = 0; i < positionnewArray.length; i++)
                {
                        xynewArray[i] = Integer.parseInt(positionnewArray[i]);
                }

                int n = xynewArray[0] + ((xynewArray[1]-1)*5);
                int o = xyoldArray[0] + ((xyoldArray[1]-1)*5);

                String direction = "";

                if(xynew == xyold-10)
                {
                        direction = "l";
                }
                else if(xynew == xyold+10)
                {
                        direction = "p";
                }
                else if(xynew == xyold-1)
                {
```

```
                        direction = "g";
                }
                else if(xynew == xyold+1)
                {
                        direction = "d";
                }
                else if(xynew == xyold-11)
                {
                        direction = "gl";
                }
                else if(xynew == xyold-9)
                {
                        direction = "gp";
                }
                else if(xynew == xyold+11)
                {
                        direction = "dp";
                }
                else if(xynew == xyold+9)
                {
                        direction = "dl";
                }



                String field=playboard.get(o);


                        playboard.put(n, field);
                        ps.moveHuman(field.toLowerCase() +
direction.toLowerCase());
                        playboard.put(o, "");

        }

        public int[] movePawnAI(HashMap<Integer, String> playboard,
PrologService ps)
        {
                int[] xy = new int[2];
                xy = ps.moveAI(playboard);
                return xy;
        }

}
```

PrologService.java

```
package star3.project.service;

import java.util.HashMap;
import java.util.Hashtable;

import jpl.*;

public class PrologService
{
        public PrologService()
        {
```

```
	}

	public void start()
	{
	    Query q1 = new Query("consult('teeko.pl')");
	       System.out.println(q1.hasSolution() ? "Plik wczytano
pomyślnie." : "Wystąpił błąd podczas wczytywania pliku.");
	}

	public void setCoord(String player)
	{
		String t4 = "rozstawienie('" + player + "')";
		Query q4 = new Query(t4);
		q4.oneSolution();
	}

	public void setCoordHuman(int position)
	{
		String t4 = "rozstawienieCzlowiek(" + position + ", 'B')";
		Query q4 = new Query(t4);
		q4.oneSolution();
	}

	public void setCoordLast()
	{
		String t4 = "rozstawienieOstatni('B')";
		Query q4 = new Query(t4);
		q4.oneSolution();
	}

	public int setCoordAI()
	{
		 int x=0, y=0;
		 Hashtable solution;

		 Query q5 = new Query(new Compound("rozstawienieAI", new Term[]
{ new Variable("PosX"), new Variable("PosY"), new Atom("R")}));

		 while (q5.hasMoreSolutions())
		 {
				solution = q5.nextSolution();
				x =
java.lang.Integer.parseInt(solution.get("PosX").toString());
				y =
java.lang.Integer.parseInt(solution.get("PosY").toString());
		 }

		return x + (y-1)*5;
	}

	public void move(String player)
	{
		String t4 = "wykonajRuch('" + player + "')";
		Query q4 = new Query(t4);
		q4.oneSolution();
	}

	public void moveHuman(String move)
	{
```

```
            String t4 = "wykonajRuchCzlowiek('B', '" + move + "')";
            Query q4 = new Query(t4);
            q4.oneSolution();
      }

      public int[] moveAI(HashMap<java.lang.Integer, String> playboard)
      {
            int x=0, y=0, xOld=0, yOld=0;
            String pawnID;
             Hashtable solution;

             Query q5 = new Query(new Compound("wykonajRuchAI", new Term[]
{ new Variable("PosX"), new Variable("PosY"), new Variable("PosXO"), new
Variable("PosYO")}));

             while (q5.hasMoreSolutions())
             {
                         solution = q5.nextSolution();
                         x =
java.lang.Integer.parseInt(solution.get("PosX").toString());
                         y =
java.lang.Integer.parseInt(solution.get("PosY").toString());
                         xOld =
java.lang.Integer.parseInt(solution.get("PosXO").toString());
                         yOld =
java.lang.Integer.parseInt(solution.get("PosYO").toString());
             }

             pawnID = playboard.get(xOld + (yOld-1)*5);
             playboard.put(x + (y-1)*5, pawnID);
             playboard.put(xOld + (yOld-1)*5, "");

             int[] tab = new int[2];
             tab[0] = xOld + (yOld-1)*5;
             tab[1] = x + (y-1)*5;

             return tab;
      }
}
```

Star3Teeko.java

```
package star3.project;

import java.io.IOException;
import java.util.HashMap;

import star3.project.service.JavaService;
import star3.project.service.PrologService;

import com.badlogic.gdx.ApplicationAdapter;
import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.graphics.GL20;
import com.badlogic.gdx.graphics.OrthographicCamera;
import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.graphics.g2d.SpriteBatch;

class TemporaryPawn {
      public int x;
```

```
	public int y;
	Field tmpOwner = null;

	TemporaryPawn() {
		reset();
	}

	void reset() {
		x = -1;
		y = -1;
		tmpOwner = Field.EMPTY;
	}

	boolean isInClosestNeighbourhood(int x, int y) {
		boolean state = false;


		if (this.y == y && (this.x - 1 == x || this.x + 1 == x)) {
			state = true;
		}

		else if (this.x == x && (this.y - 1 == y || this.y + 1 == y)) {
			state = true;
		}

		else if (this.x + 1 == x && (this.y - 1 == y || this.y + 1 ==
y)) {
			state = true;
		}

		else if (this.x - 1 == x && (this.y - 1 == y || this.y + 1 ==
y)) {
			state = true;
		}

		return state;
	}

	boolean isTemporary(int x, int y) {
		if (this.equals(x) && this.equals(y))
			return true;
		else
			return false;
	}

	boolean isAvailable() {

		if (tmpOwner.equals(Field.EMPTY))
			return true;
		else
			return false;
	}

	Field getOwner() {
		return tmpOwner;
	}
}

public class Star3Teeko extends ApplicationAdapter {
```

```
	SpriteBatch batch;
	Texture img;
	private OrthographicCamera camera;
	private TemporaryPawn tmpPawn = new TemporaryPawn();
	private Texture red, black, texT, gameDeck, win, lose;
	private int countPlayerAPawns = 0;
	private int countPlayerBPawns = 0;
	private GameStatus gameStatus;
	private Field fields[][] = new Field[5][5];
	JavaService js = new JavaService();
	PrologService ps = new PrologService();
	HashMap<Integer, String> playboard = new HashMap<Integer, String>();

	@Override
	public void create() {
		batch = new SpriteBatch();
		camera = new OrthographicCamera(Gdx.graphics.getWidth(),
				Gdx.graphics.getHeight());
		win = new Texture("Win.png");
		lose = new Texture("Lose.png");
		red = new Texture("A.png");
		black = new Texture("B.png");
		texT = new Texture("T.png");
		gameDeck = new Texture("GameDeck.png");
		newGame();
	}

	private void newGame() {
		ps.start();
		for(int i=0;i<25;i++)
		{
			playboard.put(i+1, "");
		}
		emptyFields();
		gameStatus = GameStatus.BLACKTURN;

	}

	public boolean checkWinningPositions(int x, int y, Field player) {
		int leftLowerIncline = 0;
		int rightLowerIncline = 0;
		int square = 0;
		int downStroke = 0;
		int rightStroke = 0;

		if (x - 3 >= 0 && y + 3 <= 4) {
			for (int i = 0; i < 4; i++) {

				if (fields[x - i][y + i].equals(player))
					leftLowerIncline++;
			}
			if (leftLowerIncline == 4)
				return true;
		}

		if (x + 3 <= 4 && y + 3 <= 4) {
			for (int i = 0; i < 4; i++) {

				if (fields[x + i][y + i].equals(player))
```

```
                        rightLowerIncline++;
                }
                if (rightLowerIncline == 4)
                        return true;
        }

        if (y + 3 <= 4) {
                for (int i = 0; i < 4; i++) {

                        if (fields[x][y + i].equals(player))
                                downStroke++;
                }
                if (downStroke == 4)
                        return true;
        }

        if (x + 3 <= 4) {
                for (int i = 0; i < 4; i++) {

                        if (fields[x + i][y].equals(player))
                                rightStroke++;
                }
                if (rightStroke == 4)
                        return true;
        }

        if (x + 1 <= 4 && y + 1 <= 4) {
                if (fields[x + 1][y].equals(player)
                                && fields[x + 1][y + 1].equals(player)
                                && fields[x][y + 1].equals(player)
                                && fields[x][y].equals(player)) {
                        return true;
                }

        }

        return false;
    }

    public void emptyFields() {
        for (int x = 0; x < 5; x++)
                for (int y = 0; y < 5; y++)
                        fields[x][y] = Field.EMPTY;
        countPlayerAPawns = 0;
        countPlayerBPawns = 0;
        tmpPawn.reset();
    }

    public boolean checkGameStatus(Field playah) {
        for (int x = 0; x < 5; x++) {
                for (int y = 0; y < 5; y++) {
                        if (checkWinningPositions(x, y, playah))
                                return true;

                }

        }
        return false;
    }
```

```
	@Override
	public void render() {

		try
		{
		Update();
		}
		catch(IOException e){}

		Gdx.gl.glClearColor(0.0f, 0.0f, 0.0f, 1.0f);

		Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);

		batch.setProjectionMatrix(camera.combined);
		batch.begin();
		if (gameStatus == GameStatus.BLACKTURN
				|| gameStatus == GameStatus.REDTURN) {
			batch.draw(gameDeck, 0, 0);
			for (int x = 0; x < 5; x++)
				for (int y = 0; y < 5; y++) {
					if (fields[x][y] == Field.X) {
						batch.draw(red, 153 * 5 - x * 153
								- (red.getWidth() / 2 -
153 / 2) - 153, 153 * 5
								- y * 153 -
(red.getWidth() / 2 - 153 / 2)
								- 153);

					}
					if (fields[x][y] == Field.O) {
						batch.draw(black, 153 * 5 - x * 153
								- (red.getWidth() / 2 -
153 / 2) - 153, 153 * 5
								- y * 153 -
(red.getWidth() / 2 - 153 / 2)
								- 153);
					}
					if (fields[x][y] == Field.T) {
						batch.draw(texT, 153 * 5 - x * 153
								- (red.getWidth() / 2 -
153 / 2) - 153, 153 * 5
								- y * 153 -
(texT.getWidth() / 2 - 153 / 2)
								- 153);
					}
				}
		} else if (gameStatus == GameStatus.REDWIN) {
			batch.draw(lose, 0, 0);
		} else if (gameStatus == GameStatus.BLACKWIN) {
			batch.draw(win, 0, 0);
		}
		batch.end();
	}

	public void dispose() {
		batch.dispose();
		black.dispose();
		red.dispose();
```

```
			gameDeck.dispose();

		}

		public void resize(int width, int height) {
			camera.setToOrtho(false, width, height);
		}

		private void Update() throws IOException{

			if (Gdx.input.justTouched()
					&& (gameStatus != GameStatus.REDTURN && gameStatus
!= GameStatus.BLACKTURN))
			{
				dispose();
				System.exit(0);
			}
			else

			if (gameStatus == GameStatus.REDTURN) {

				int fieldX = 0, fieldY = 0;
				if (countPlayerBPawns < 4) {
					countPlayerBPawns++;
					ps.setCoord("R");
					int xy = js.putPawnAI(playboard, ps,
countPlayerBPawns);
					if(xy%5 != 0)
					{
						fieldX = (xy%5)-1;
						fieldY = (xy/5+1)-1;
					}
					else
					{
						fieldX = ((xy-1)%5);
						fieldY = (xy/5)-1;
					}
					if(fieldX == 0)
					{
						fieldX=4;
					}
					else if(fieldX == 1)
					{
						fieldX=3;
					}
					else if(fieldX == 3)
					{
						fieldX=1;
					}
					else if(fieldX == 4)
					{
						fieldX=0;
					}
					fields[fieldX][fieldY] = Field.X;
					gameStatus = GameStatus.BLACKTURN;

					if (countPlayerBPawns == 4)
					{
						ps.setCoordLast();
```

```
                        }


                } else

                if (countPlayerBPawns >= 4) {
                        int[] xy = new int[2];
                        ps.move("R");
                        xy = js.movePawnAI(playboard, ps);
                        tmpPawn.tmpOwner = Field.X;
                        if(xy[0]%5 != 0)
                        {
                                tmpPawn.x = (xy[0]%5)-1;
                                tmpPawn.y = (xy[0]/5+1)-1;
                        }
                        else
                        {
                                tmpPawn.x = ((xy[0]-1)%5);
                                tmpPawn.y = (xy[0]/5)-1;
                        }
                        if(tmpPawn.x == 0)
                        {
                                tmpPawn.x=4;
                        }
                        else if(tmpPawn.x == 1)
                        {
                                tmpPawn.x=3;
                        }
                        else if(tmpPawn.x == 3)
                        {
                                tmpPawn.x=1;
                        }
                        else if(tmpPawn.x == 4)
                        {
                                tmpPawn.x=0;
                        }
                        fields[tmpPawn.x][tmpPawn.y] = Field.T;

                        fields[tmpPawn.x][tmpPawn.y] = Field.EMPTY;
                        tmpPawn.reset();
                        if(xy[1]%5 != 0)
                        {
                                fieldX = (xy[1]%5)-1;
                                fieldY = (xy[1]/5+1)-1;
                        }
                        else
                        {
                                fieldX = ((xy[1]-1)%5);
                                fieldY = (xy[1]/5)-1;
                        }
                        if(fieldX == 0)
                        {
                                fieldX=4;
                        }
                        else if(fieldX == 1)
                        {
                                fieldX=3;
                        }
                        else if(fieldX == 3)
```

```
                        {
                                fieldX=1;
                        }
                        else if(fieldX == 4)
                        {
                                fieldX=0;
                        }
                        fields[fieldX][fieldY] = Field.X;
                        gameStatus = GameStatus.BLACKTURN;

                }

                if (checkGameStatus(Field.X))
                {
                        gameStatus = GameStatus.REDWIN;
                }


        }

        else if (Gdx.input.justTouched() && gameStatus ==
GameStatus.BLACKTURN) {
                float posX = -((float) Gdx.input.getX() /
Gdx.graphics.getWidth() * 5) + 5;
                float posY = (float) Gdx.input.getY() /
Gdx.graphics.getHeight() * 5;

                int fieldX = 0, fieldY = 0;
                if (posX >= 0 && posX <= 1)
                        fieldX = 0;
                if (posX > 1 && posX <= 2)
                        fieldX = 1;
                if (posX > 2 && posX <= 3)
                        fieldX = 2;
                if (posX > 3 && posX <= 4)
                        fieldX = 3;
                if (posX > 4 && posX <= 5)
                        fieldX = 4;

                if (posY >= 0 && posY <= 1)
                        fieldY = 0;
                if (posY > 1 && posY <= 2)
                        fieldY = 1;
                if (posY > 2 && posY <= 3)
                        fieldY = 2;
                if (posY > 3 && posY <= 4)
                        fieldY = 3;
                if (posY > 4 && posY <= 5)
                        fieldY = 4;

                if (fields[fieldX][fieldY] == Field.EMPTY &&
countPlayerAPawns < 4) {
                        countPlayerAPawns++;
                        fields[fieldX][fieldY] = Field.O;
                        if(fieldX == 0)
                        {
                                fieldX=4;
                        }
                        else if(fieldX == 1)
```

```
                        {
                                fieldX=3;
                        }
                        else if(fieldX == 3)
                        {
                                fieldX=1;
                        }
                        else if(fieldX == 4)
                        {
                                fieldX=0;
                        }
                        int xy =
Integer.parseInt(Integer.toString(fieldX+1) + Integer.toString(fieldY+1));
                        ps.setCoord("B");
                        js.putPawnHuman(playboard, ps, countPlayerAPawns,
xy);
                        gameStatus = GameStatus.REDTURN;

                } else

                if (fields[fieldX][fieldY] == Field.O &&
countPlayerAPawns >= 4
                                && tmpPawn.isAvailable()) {
                        tmpPawn.tmpOwner = Field.O;
                        tmpPawn.x = fieldX;
                        tmpPawn.y = fieldY;
                        fields[fieldX][fieldY] = Field.T;

                } else

                if (fields[fieldX][fieldY] == Field.T &&
countPlayerAPawns >= 4
                                && !tmpPawn.isAvailable()) {
                        tmpPawn.reset();
                        fields[fieldX][fieldY] = Field.O;

                } else

                if (fields[fieldX][fieldY] == Field.EMPTY &&
countPlayerBPawns >= 4
                                && !tmpPawn.isAvailable()
                                && tmpPawn.isInClosestNeighbourhood(fieldX,
fieldY)) {
                        fields[tmpPawn.x][tmpPawn.y] = Field.EMPTY;
                        fields[fieldX][fieldY] = Field.O;

                        if(fieldX == 0)
                        {
                                fieldX=4;
                        }
                        else if(fieldX == 1)
                        {
                                fieldX=3;
                        }
                        else if(fieldX == 3)
                        {
                                fieldX=1;
                        }
                        else if(fieldX == 4)
```

```
                        {
                                fieldX=0;
                        }

                        if(tmpPawn.x == 0)
                        {
                                tmpPawn.x=4;
                        }
                        else if(tmpPawn.x == 1)
                        {
                                tmpPawn.x=3;
                        }
                        else if(tmpPawn.x == 3)
                        {
                                tmpPawn.x=1;
                        }
                        else if(tmpPawn.x == 4)
                        {
                                tmpPawn.x=0;
                        }

                        int xyold =
Integer.parseInt(Integer.toString(tmpPawn.x+1) +
Integer.toString(tmpPawn.y+1));
                        int xynew =
Integer.parseInt(Integer.toString(fieldX+1) + Integer.toString(fieldY+1));



                                ps.move("B");
                                js.movePawnHuman(playboard, ps, xyold, xynew);
                                tmpPawn.reset();
                                gameStatus = GameStatus.REDTURN;
                        }

                        if (checkGameStatus(Field.O))
                        {
                                gameStatus = GameStatus.BLACKWIN;
                        }

                }

        }

}
```

Field.java

```
package star3.project;

public enum Field {
        X,
        O,
        T,
        EMPTY

}
```

GameStatus.java

```
package star3.project;

public enum GameStatus {
	REDTURN,
	BLACKTURN,
	REDWIN,
	BLACKWIN
}
```

teeko.pl

```
%%%%%%%%%%  SZTUCZNA INTELIGENCJA  %%%%%%%%%%

:- dynamic pozycja/3.

%Sprawdza, czy A jest rÄ‡Ľne od B i C jest rÄ‡Ľne od D.
czyRozne(A,B,_,_) :- dif(A,B), !.
czyRozne(_,_,C,D) :- dif(C,D), !.

%Zwraca wszystkie pozycje pionkÄ‡w gracza
wszystkiePozycje('B', X1, Y1, X2, Y2, X3, Y3, X4, Y4) :- pozycja(X1, Y1,
'B1'), pozycja(X2, Y2, 'B2'), pozycja(X3, Y3, 'B3'), pozycja(X4, Y4, 'B4'),
!.
wszystkiePozycje('R', X1, Y1, X2, Y2, X3, Y3, X4, Y4) :- pozycja(X1, Y1,
'R1'), pozycja(X2, Y2, 'R2'), pozycja(X3, Y3, 'R3'), pozycja(X4, Y4, 'R4'),
!.

%Sprawdza, czy wszystkie elementy od pierwszego elementu na liĹ›cie sÄ…
pomiÄ™dzy MIN a MAX.
czyPomiedzy([T|R], MIN, MAX) :- between(MIN, MAX, T), czyPomiedzy(R, MIN,
MAX).
czyPomiedzy([], _, _).

%Zwraca wszystkie pionki zaleĹĽnie od koloru.
zwrocPionki('B', ['B1', 'B2', 'B3', 'B4']).
zwrocPionki('R', ['R1', 'R2', 'R3', 'R4']).

%Zwraca nazwÄ™ przeciwnika.
zamienGracza('R', 'B').
zamienGracza('B', 'R').

%ZwrÄ‡cenie poczÄ…tkowych elementÄ‡w spoĹ›rÄ‡d wszystkich elementÄ‡w
pierwszego argumentu listy.
zwrocPierwsze([], []) :- !.
zwrocPierwsze([[T|_]|R2], [T|R3]) :- zwrocPierwsze(R2, R3).

%Zwraca min lub max listy w zaleĹĽnoĹ›ci od pierwszego argumentu.
minLubMax(1, L, V) :- max_list(L, V).
minLubMax(-1, L, V) :- min_list(L, V).

%Ostatni nieustawiony pionek.
ostatniPionek(Gracz, Pionek) :- zwrocPionki(Gracz, PioneksList),
zwrocNieustawionyPionek(PioneksList, Pionek).

%Zwraca pierwszy nieustawiony pionek.
zwrocNieustawionyPionek([M|_], M) :- not(pozycja(_,_,M)), !.
zwrocNieustawionyPionek([T|R], M) :- pozycja(_,_,T),
```

```
zwrocNieustawionyPionek(R, M).

%Sprawdza, czy gracz grajÄ…cy czarnymi pionkami wygrywa.
zwyciezca('B') :-
    pozycja(B1X, B1Y, 'B1'),
    pozycja(B2X, B2Y, 'B2'),
    pozycja(B3X, B3Y, 'B3'),
    pozycja(B4X, B4Y, 'B4'),
    zwyciezca(B1X, B1Y, B2X, B2Y, B3X, B3Y, B4X, B4Y), !.

%Sprawdza, czy gracz grajÄ…cy czerwonymi pionkami wygrywa.
zwyciezca('R') :-
    pozycja(R1X, R1Y, 'R1'),
    pozycja(R2X, R2Y, 'R2'),
    pozycja(R3X, R3Y, 'R3'),
    pozycja(R4X, R4Y, 'R4'),
    zwyciezca(R1X, R1Y, R2X, R2Y, R3X, R3Y, R4X, R4Y), !.

%Wykrywa ustawienie pionkÃłw dla wygranej w poziomie.
zwyciezca(M1X, M1Y, M2X, M2Y, M3X, M3Y, M4X, M4Y) :-
    permutation([[X1, Y], [X2, Y], [X3, Y], [X4, Y]], [[M1X, M1Y], [M2X,
M2Y], [M3X, M3Y], [M4X, M4Y]]),
    czyPomiedzy([X1, X2, X3, X4, Y], 1, 5),
    X2 is X1 + 1, X3 is X2 + 1, X4 is X3 + 1.

%Wykrywa ustawienie pionkÃłw dla wygranej w pionie.
zwyciezca(M1X, M1Y, M2X, M2Y, M3X, M3Y, M4X, M4Y) :-
    permutation([[X, Y1], [X, Y2], [X, Y3], [X, Y4]], [[M1X, M1Y], [M2X,
M2Y], [M3X, M3Y], [M4X, M4Y]]),
    czyPomiedzy([X, Y1, Y2, Y3, Y4], 1, 5),
    Y2 is Y1 + 1, Y3 is Y2 + 1, Y4 is Y3 + 1.

%Wykrywa ustawienie pionkÃłw dla wygranej po przekÄ…tnej.
zwyciezca(M1X, M1Y, M2X, M2Y, M3X, M3Y, M4X, M4Y) :-
    permutation([[X1, Y1], [X2, Y2], [X3, Y3], [X4, Y4]], [[M1X, M1Y],
[M2X, M2Y], [M3X, M3Y], [M4X, M4Y]]),
    czyPomiedzy([X1, X2, X3, X4, Y1, Y2, Y3, Y3], 1, 5),
    Y2 is Y1 + 1, X2 is X1 + 1, Y3 is Y2 + 1, X3 is X2 + 1, Y4 is Y3 + 1,
X4 is X3 + 1.

%Wykrywa ustawienie pionkÃłw dla wygranej po przeciwnej przekÄ…tnej.
zwyciezca(M1X, M1Y, M2X, M2Y, M3X, M3Y, M4X, M4Y) :-
    permutation([[X1, Y1], [X2, Y2], [X3, Y3], [X4, Y4]], [[M1X, M1Y],
[M2X, M2Y], [M3X, M3Y], [M4X, M4Y]]),
    czyPomiedzy([X1, X2, X3, X4, Y1, Y2, Y3, Y3], 1, 5),
    Y2 is Y1 + 1, X2 is X1 - 1, Y3 is Y2 + 1, X3 is X2 - 1, Y4 is Y3 + 1,
X4 is X3 - 1.

%Wykrywa ustawienie pionkÃłw dla wygranej w kwadracie.
zwyciezca(M1X, M1Y, M2X, M2Y, M3X, M3Y, M4X, M4Y) :-
    permutation([[X1, Y1], [X2, Y1], [X2, Y2], [X1, Y2]], [[M1X, M1Y],
[M2X, M2Y], [M3X, M3Y], [M4X, M4Y]]),
    czyPomiedzy([X1, X2, Y1, Y2], 1, 5),
    X2 is X1 + 1, Y2 is Y1 + 1.

%Obliczenie poĹ,oĹĽenia pionka.
obliczPolozenie(X, Y, M) :-
    czyPomiedzy([X, Y], 1, 5),
    not(pozycja(X, Y, _)),
```

```
    not(pozycja(_, _, M)).

%Ustawienie pionka na planszy.
ustawPionek(X, Y, M) :- nonvar(X), nonvar(Y), nonvar(M), obliczPolozenie(X,
Y, M), assert(pozycja(X, Y, M)), !.
ustawPionek(PosX, PosY, X, Y, M) :- nonvar(X), nonvar(Y), nonvar(M),
obliczPolozenie(X, Y, M), assert(pozycja(X, Y, M)), PosX = X, PosY = Y, !.

%UsuniÄ™cie pionka z planszy.
usunPionek(M) :- nonvar(M), pozycja(X, Y, M), retract(pozycja(X, Y, M)), !.

%Obliczenie wszystkich moĹĽliwych pozycji do ustawienia dla pionka i
sprawdzenie, czy pionek jest pierwszym, ktÄłry moĹĽe byÄ‡ ustawiony.
obliczWszystkiePozycje(C, M, X, Y) :-
    ostatniPionek(C, M),
    obliczPolozenie(X, Y, 'NE').

%Wykonanie ruchu pionka, przesuniÄ™cie go do innej pozycji z poprzedniej.
przesunPionek(TOX, TOY, M) :- pozycja(FROMX, FROMY, M), obliczRuch(TOX,
TOY, M), assert(pozycja(TOX, TOY, M)), retract(pozycja(FROMX, FROMY, M)),
!.
przesunPionek(PosX, PosY, PosXO, PosYO, TOX, TOY, M) :- pozycja(FROMX,
FROMY, M), obliczRuch(TOX, TOY, M), assert(pozycja(TOX, TOY, M)),
retract(pozycja(FROMX, FROMY, M)), PosX is TOX, PosY is TOY, PosXO is
FROMX, PosYO is FROMY, !.

%Obliczenie wszystkich moĹĽliwych ruchÄłw dla wszystkich pionkÄłw danego
koloru.
obliczWszystkieMozliweRuchy(C, M, TOX, TOY) :-
    zwrocPionki(C, LM),
    member(M, LM),
    obliczRuch(TOX, TOY, M).

%Obliczenie ruchu dla pionka.
obliczRuch(TOX, TOY, M) :-
    pozycja(FROMX, FROMY, M),
    czyPomiedzy([TOX, TOY], 1, 5),
    MAXX is FROMX + 1, MINX is FROMX - 1, czyPomiedzy([TOX], MINX, MAXX),
    MAXY is FROMY + 1, MINY is FROMY - 1, czyPomiedzy([TOY], MINY, MAXY),
    czyRozne(TOX, FROMX, TOY, FROMY),
    not(pozycja(TOX, TOY, _)).

%Obliczenie zbioru wszystkich ruchÄłw w zaleĹĽnoĹ›ci od fazy rozgrywki.
obliczWszystkieRuchy(C, M, X, Y) :- ostatniPionek(C, M),
obliczWszystkiePozycje(C, M, X, Y).
obliczWszystkieRuchy(C, M, X, Y) :- not(ostatniPionek(C, M)),
obliczWszystkieMozliweRuchy(C, M, X, Y).

%Predykat dla algorytmu sztucznej inteligencji.
najkorzystniejszyRuch(C, _, _, M,3,3):- obliczLiczbeRuchow(C,0, _),
not(pozycja(3,3,_)), ostatniPionek(C, M),!.
najkorzystniejszyRuch(C, _, _, M,X,Y):- obliczLiczbeRuchow(C,0, _),
ostatniPionek(C, M), random(RndX), random(RndY), X is 2 + 2 * round(RndX),
Y is 2 + 2 * round(RndY),!.
najkorzystniejszyRuch(C, L, IAL, M, X, Y) :- najkorzystniejszyRuch(C, L,
IAL, _, 1, -200, 200, [M, X, Y]).

%Zwraca najlepsze moĹĽliwe przesuniÄ™cie pionka dla sztucznej inteligencji.
szansa(3).
```

```
szansaAI(3) :- random(X), X < 0.8.
najkorzystniejszyRuch(C, 0, L, X, _, _, _, _) :- zamienGracza(C, C1),
ocen(L, C1, X).
najkorzystniejszyRuch(C, _, L, X, MINMAX, _, _, _) :- zwyciezca(C),
szansaAI(L), X is MINMAX*100.
najkorzystniejszyRuch(C, _, L, X, MINMAX, _, _, _) :- zamienGracza(C, C2),
zwyciezca(C2), szansa(L), X is -1*MINMAX*90.
najkorzystniejszyRuch(C, V, IAL, RE, MINMAX, ALFA, BETA, ADD) :- dif(V, 0),
findall([C, M, X, Y], obliczWszystkieRuchy(C, M, X, Y), L), VAL is -1 *
MINMAX * 200, zmianaPolozenia(L, V, IAL, LR, MINMAX, ALFA, BETA, VAL),
zwrocPierwsze(LR, LV), minLubMax(MINMAX, LV, RE), nth1(_, LR, [RE|ADD]), !.

%GĹ‚Ăłwna czÄ™Ĺ›Ä‡ algorytmu zaleĹĽna od fazy gry (rozstawienie pionkĂłw
lub ich przemieszczanie po rozstawieniu).
zmianaPolozenia([[C, M, X, Y]|R], V, IAL, [TE|RE], MINMAX, ALFA, BETA, VAL)
:-
    ostatniPionek(C, M),
    ustawPionek(X, Y, M), zamienGracza(C, C2), V1 is V - 1, INVMINMAX is -1
* MINMAX,
    najkorzystniejszyRuch(C2, V1, IAL, T, INVMINMAX, ALFA, BETA, _), TE =
[T, M, X, Y], usunPionek(M),
    minLubMax(MINMAX, [VAL, T], NEWVAL), alfaBetaCiecie(MINMAX, NEWVAL,
ALFA, BETA, IAL, R, V, RE).
zmianaPolozenia([[C, M, X, Y]|R], V, IAL, [TE|RE], MINMAX, ALFA, BETA, VAL)
:-
    not(ostatniPionek(C, M)),
    pozycja(FROMX, FROMY, M), przesunPionek(X, Y, M), zamienGracza(C, C2),
V1 is V - 1, INVMINMAX is -1 * MINMAX,
    najkorzystniejszyRuch(C2, V1, IAL, T, INVMINMAX, ALFA, BETA, _), TE =
[T, M, X, Y], przesunPionek(FROMX, FROMY, M),
    minLubMax(MINMAX, [VAL, T], NEWVAL), alfaBetaCiecie(MINMAX, NEWVAL,
ALFA, BETA, IAL, R, V, RE).
zmianaPolozenia([], _, _, [], _, _, _, _).

%Koniec algorytmu dla ruchu sztucznej inteligencji, sprawdza, czy alfa-beta
ciÄ™cia sÄ… moĹĽliwe.
alfaBetaCiecie(-1, VAL, ALFA, _, _, _, _, []) :- ALFA >= VAL.
alfaBetaCiecie(-1, VAL, ALFA, BETA, IAL, R, V, RE) :- min_list([BETA, VAL],
BETA2), zmianaPolozenia(R, V, IAL, RE, -1, ALFA, BETA2, VAL).
alfaBetaCiecie(1, VAL, _, BETA, _, _, _, []) :- VAL >= BETA.
alfaBetaCiecie(1, VAL, ALFA, BETA, IAL, R, V, RE) :- max_list([ALFA, VAL],
ALFA2), zmianaPolozenia(R, V, IAL, RE, 1, ALFA2, BETA, VAL).

%Funkcja oceniajÄ…ca.
ocen(_,C,100):- zwyciezca(C),!.
ocen(D,C,-100):- D > 1, zamienGracza(C,C1), zwyciezca(C1),!.
ocen(3,C,N):- obliczLiczbeRuchow(C, NbMC, LMC), NbMC<4,
zamienGracza(C,C1),obliczLiczbeRuchow(C1, NbMC1, LMC1), NbMC1<4, !,
obliczZwyciestwo(C,NWS, NbMC, LMC), obliczZwyciestwo(C1,NWS1, NbMC1, LMC1),
N is (NbMC*NWS - 1.5*NbMC1*NWS1).
ocen(3,C,N):- obliczLiczbeRuchow(C, 4, _),
zamienGracza(C,C1),obliczLiczbeRuchow(C1, 4, _), !,obliczUstawienie(C,NA),
obliczUstawienie(C1,NA2), N is (4*NA - 7*NA2).
ocen(3,C,N):- obliczUstawienie(C,NA), zamienGracza(C,C1),
obliczUstawienie(C1,NA2), obliczLiczbeRuchow(C, NbMC,
LMC),obliczLiczbeRuchow(C1, NbMC1, LMC1),obliczZwyciestwo(C,NWS, NbMC,
LMC), obliczZwyciestwo(C1,NWS1, NbMC1, LMC1), N is (4*NA - 7*NA2 + NbMC*NWS
- NbMC1*NWS1).
```

```
%Oblicza liczbÄ™ pionkÃłw ustawionych dla danego koloru (sprawdza, czy
wszystkie pola wymagane do wygranej nie sÄ… zajÄ™te przez pionki
przeciwnika).
obliczUstawienie(C,N):- wszystkiePozycje(C, _, _, X2, Y2, X3, Y3, X4, Y4),
zwyciezca(X, Y, X2, Y2, X3, Y3, X4, Y4), czyPomiedzy([X,Y], 1, 5),
sprawdzPustePole(X,Y,EC),!, N is 3*EC.
obliczUstawienie(C,N):- wszystkiePozycje(C, X1, Y1, _, _, X3, Y3, X4, Y4),
zwyciezca(X1, Y1, X, Y, X3, Y3, X4, Y4), czyPomiedzy([X,Y], 1, 5),
sprawdzPustePole(X,Y,EC),!, N is 3*EC.
obliczUstawienie(C,N):- wszystkiePozycje(C, X1, Y1, X2, Y2, _, _, X4, Y4),
zwyciezca(X1, Y1, X2, Y2, X, Y, X4, Y4), czyPomiedzy([X,Y], 1, 5),
sprawdzPustePole(X,Y,EC),!, N is 3*EC.
obliczUstawienie(C,N):- wszystkiePozycje(C, X1, Y1, X2, Y2, X3, Y3, _, _),
zwyciezca(X1, Y1, X2, Y2, X3, Y3, X, Y), czyPomiedzy([X,Y], 1, 5),
sprawdzPustePole(X,Y,EC),!, N is 3*EC.
obliczUstawienie(_,0).

%Sprawdza, czy pole o koordynatach X, Y jest puste.
sprawdzPustePole(X,Y,1):- not(pozycja(X,Y,_)),!.
sprawdzPustePole(_,_,0.2).

%Oblicza liczbÄ™ wygrywajÄ…cych pozycji moĹĽliwych dla pionkÃłw w
pozostaĹ‚ych dla nich miejscach.
obliczZwyciestwo(C,T, N, L):- N<4, zamienGracza(C1,C),
obliczPozycjeN(4,L,[X1, Y1, X2, Y2, X3, Y3, X4, Y4]), setof([[X1, Y1], [X2,
Y2], [X3, Y3], [X4, Y4]], (zwyciezca(X1, Y1, X2, Y2, X3, Y3, X4, Y4),
czyPomiedzy([X1, Y1, X2, Y2, X3, Y3, X4, Y4], 1, 5),
pionkiPrzeciwnika(C1,[X1, Y1, X2, Y2, X3, Y3, X4, Y4])),
L2),usunDuplikat(L2,L3), length(L3, T),!.
obliczZwyciestwo(_,0,_,_).

%Oblicza liczbÄ™ postawieĹ„ pionka danego koloru. Zwraca pozycje pionkÃłw
sÄ…siadujÄ…cych do niego.
obliczLiczbeRuchow(C,N, L2):- findall([X,Y], (zwrocPionki(C, L),
member(P,L), pozycja(X,Y,P)),L2), length(L2, N).

%Tworzy listÄ™ zawierajÄ…cÄ… N pozycji wzbogaconÄ… o koordynaty pionkÃłw
przekazywanych jako parametr.
obliczPozycjeN(0, _, []):-!.
obliczPozycjeN(N, [], [_,_|R]):- N1 is N-1, obliczPozycjeN(N1, [], R),!.
obliczPozycjeN(N, [[X,Y]|R], [X,Y|R2]):-  N1 is N-1, obliczPozycjeN(N1, R,
R2).

%Sprawdza, czy pozycje zawarte na liĹ›cie nie sÄ… zajmowane przez pionki
gracza.
pionkiPrzeciwnika(_,[]):- !.
pionkiPrzeciwnika(C,[X,Y|R]):-
zwrocPionki(C,[C1,C2,C3,C4]),not(pozycja(X,Y,C1)),not(pozycja(X,Y,C2)),not(
pozycja(X,Y,C3)),not(pozycja(X,Y,C4)), pionkiPrzeciwnika(C,R).

%Usuwanie duplikatÃłw
usunDuplikat([],[]):-!.
usunDuplikat([T|R],R2):- member(T2,R), obliczIdentyczne(T,T2),
usunDuplikat(R,R2),!.
usunDuplikat([T|R],[T|R2]):-usunDuplikat(R,R2).

%Sprwadza, czy dwie pozycje sÄ… identyczne.
obliczIdentyczne([],_):-!.
obliczIdentyczne([T|R],L):-member(T,L),obliczIdentyczne(R,L).
```

```
%%%%%%%%%%%  ETAP 0 - WSTÄ�PNA KONFIGURACJA GRY  %%%%%%%%%%%

%Definicje predykatÄłw zmieniajÄ…cych siÄ™ w trakcie dziaĹ‚ania programu.
:- dynamic sztucznaInteligencja/1.

%PoczÄ…tkowe przypisanie zmiennych.
start:-
        assert(sztucznaInteligencja(3)).

%%%%%%%%%%%  ETAP 1 - ROZSTAWIENIE PIONKÄ“W  %%%%%%%%%%%

rozstawienie(Gracz):-
        not(uruchomPrzemieszczanie),
        zamienGracza(Gracz, Przeciwnik), not(zwyciezca(Przeciwnik)), !.

%Po rozstawieniu wszystkich pionkÄłw sprawdza, czy nie ma zwyciÄ™zcy.
rozstawienieOstatni(Gracz):-
        zamienGracza(Gracz, Przeciwnik), zwyciezca(Przeciwnik), !.

%Rozstawienie pionka przez czĹ‚owieka.
rozstawienieCzlowiek(Pos, Gracz):-
        ostatniPionek(Gracz, Pionek), wprowadzPozycje(Pos, Pionek), !.

%Rozstawienie pionka przez sztucznÄ… inteligencjÄ™.
rozstawienieAI(PosX, PosY, Gracz):-
        sztucznaInteligencja(L),
             najkorzystniejszyRuch(Gracz, 3, L, M, X, Y),
             ustawPionek(PosX, PosY, X, Y, M), !.

uruchomPrzemieszczanie:-
        not(ostatniPionek('B', _)), not(ostatniPionek('R', _)), !,
        not(zwyciezca('B')), not(zwyciezca('R')).

%Zapisanie pionka gracza dla wprowadzonej pozycji.
wprowadzPozycje(Pos, Pionek):-
        ostatniPionek('B', Pionek),
        name(Pos, ListaKoordynatow), uzyskajKoordynaty(Pionek,
ListaKoordynatow, X, Y),
        not(pozycja(X, Y, _)), !, ustawPionek(X, Y, Pionek).

%Uzyskanie koordynatÄłw pionka postawionego przez gracza.
uzyskajKoordynaty(_, [], _, _).
uzyskajKoordynaty(_, [X, Y], X1, Y1):-
        X1 is X - 48, Y1 is Y - 48,
        czyPomiedzy([X1, Y1], 1, 5), !.

%%%%%%%%%%%  ETAP 2 - PRZESTAWIANIE PIONKÄ“W  %%%%%%%%%%%

wykonajRuch(Gracz):-
        zamienGracza(Gracz, Przeciwnik), not(zwyciezca(Przeciwnik)), !.

%PrzesuniÄ™cie wybranego pionka na wybranÄ… pozycjÄ™ przez gracza.

wykonajRuchCzlowiek(Gracz, Move):-
        name(Move, ListaPrzemieszczenia), sprawdzPoprawnoscRuchu(Gracz,
ListaPrzemieszczenia), !.

%Wykonanie przemieszczenia przez sztucznÄ… inteligencjÄ™.
```

```
wykonajRuchAI(PosX, PosY, PosXO, PosYO):-
        sztucznaInteligencja(L), najkorzystniejszyRuch('R', 3, L, Pionek,
X, Y), przesunPionek(PosX, PosY, PosXO, PosYO, X, Y, Pionek), !.

%Sprawdzanie poprawnoĹ›ci komendy przemieszczenia pionka w liniach
prostych.
sprawdzPoprawnoscRuchu(Gracz, [_, Wartosc, ProstyKierunekWartosc]):-
        NumerPionka is Wartosc - 48,
        between(1, 4, NumerPionka), !,
        concat(Gracz, NumerPionka, Pionek),
        name(ProstyKierunek, [ProstyKierunekWartosc]),
        ruchProsty(Pionek, ProstyKierunek).

%Sprawdzanie poprawnoĹ›ci komendy przemieszczenia pionka po skosie.
sprawdzPoprawnoscRuchu(Gracz, [_, Wartosc, PierwszyKierunekWartosc,
DrugiKierunekWartosc]):-
        NumerPionka is Wartosc - 48,
        between(1, 4, NumerPionka), !,
        concat(Gracz, NumerPionka, Pionek),
        name(FirstMove, [PierwszyKierunekWartosc]),
        name(SecondMove, [DrugiKierunekWartosc]),
        ruchZlozony(Pionek, FirstMove, SecondMove).

%Przemieszczenie pionka do zadanej pozycji przez gracza w liniach prostych.
ruchProsty(Pionek, ProstyKierunek):-
        pozycja(X, Y, Pionek),
        mozliweRuchy(X, Y, ProstyKierunek, NewX, NewY), not(pozycja(NewX,
NewY, _)), !,
        przesunPionek(NewX, NewY, Pionek).

%Przemieszczenie pionka do zadanej pozycji przez gracza po skosie.
ruchZlozony(Pionek, FirstMove, SecondMove):-
        pozycja(X, Y, Pionek),
        mozliweRuchy(X, Y, FirstMove, NewX1, NewY1),
        mozliweRuchy(NewX1, NewY1, SecondMove, NewX2, NewY2),
        not(pozycja(NewX2, NewY2, _)), !,
        przesunPionek(NewX2, NewY2, Pionek).

%MoĹĽliwe ruchy proste dla pionka.
mozliweRuchy(X, Y, 'g', X, NewY):- NewY is Y - 1, czyPomiedzy([NewY], 1,
5), !.
mozliweRuchy(X, Y, 'd', X, NewY):- NewY is Y + 1, czyPomiedzy([NewY], 1,
5), !.
mozliweRuchy(X, Y, 'p', NewX, Y):- NewX is X + 1, czyPomiedzy([NewX], 1,
5), !.
mozliweRuchy(X, Y, 'l', NewX, Y):- NewX is X - 1, czyPomiedzy([NewX], 1,
5), !.

%Uruchomiane podczas wczytania pliku.
:- start.
```