18 czerwca 2013r.

Rafał Nowicki
Marcin Gmyz
Łukasz Szabelski
I12-2

Sprawozdanie z laboratorium sztucznej inteligencji

Informatyka, sem. VI

I. Opis zadania

Zadanie jakie należało wykonać to realizacja strategicznej gry planszowej z rodziny *N in a row* o nazwie ***Freedom***. Jest to gra dla dwóch graczy przypominająca nieco: *Connect four* lub *Kółko i Krzyżyk*.

Rozgrywka toczy się na planszy o wymiarach 8x8 pól, początkowo pustej. Każdy z graczy (w każdym ruchu) umieszcza jeden pionek na planszy (rozpoczynający kładzie go w dowolnym miejscu). W każdym kolejnym ruchu gracze mogą położyć pionek jedynie na polach pustych - sąsiadujących z pionkiem, który ostatnio został położony. Jeżeli dojdzie do sytuacji, w której gracz nie może wykonać ruchu ze względu na zajęte wszystkie przyległe pola, otrzymuje przywilej *Freedom* (*Wolność*). Dzięki temu może postawić pionek na dowolnym, pustym polu.

Celem gry jest posiadanie większej niż przeciwnik ilości (pionowych, poziomych, lub ukośnych) *czwórek* o tym samym kolorze, po zakończeniu gry.

II. Założenia realizacyjne

1. Założenia dodatkowe

Gra zrealizowana jest zgodnie z punktem I. Żadne dodatkowe założenia nie zostały wykonane.

2. Algorytmy używane do rozwiązania zadania:

Algorytm zliczający wystąpienia *czwórek* i ilość zdobytych przez graczy punktów:

Dane: *pointsArr* – tablica obiektów zawierająca współrzędne i typ pola, *boardWidth* – szerokość tablicy (planszy), *boardHeight* – wysokość tablicy (planszy), *tempArr* – tablica pomocnicza.

Wyjście: *scorePlayer1*, *scorePlayer2* – punkty gracza pierwszego i drugiego.

Metoda: Niech tablica *tempArr* początkowo będzie pusta.

1. Sprawdź zapełnienie tablicy *pointsArr:*
    a. Jeśli tablica posiada chociaż jedno pole typu 0, rozpocznij algorytm od początku,
    b. Jeśli tablica nie posiada żadnego pola typu 0, kontynuuj algorytm,
2. Sprawdź rezultat – dla każdego elementu tablicy *pointsArr* o indeksach *i, j*:
    a. Sprawdź czy element jest typu 1 lub 2, jeśli tak – kontynuuj algorytm, jeśli nie – przejdź do następnego elementu,

b. Sprawdź czy element znajduje się w tablicy tymczasowej *tempArr*, jeśli tak – przejdź do kolejnego elementu, jeśli nie - sprawdź:
   - Czy element sąsiaduje z innymi trzema elementami tego samego typu, w jednej linii: po skosie w prawo, po skosie w lewo, pionowo lub poziomo,
   - Jeśli tak, dodaj 4 sąsiadujące elementy tego samego typu do tablicy tymczasowej *tempArr*,
   - Jeśli typ pola == 1, *scorePlayer1++*,
   - Jeśli typ pola == 2, *scorePlayer2++*,

3. Wyświetl rezultat.

Algorytm obciętego mini-maksu:

Do utworzenia drzewa gry i wybrania na jego podstawie kolejnego ruchu komputera zastosowaliśmy algorytm obciętego mini-maksu.

Dane: *s* – stan, *g* – gracz, *k* – poziom

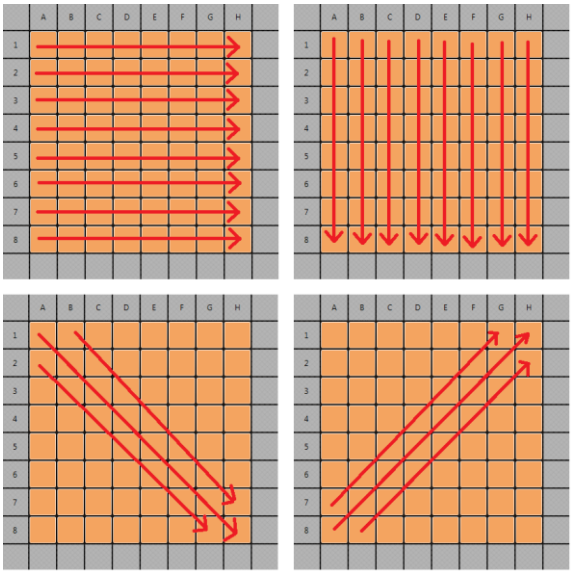
Wyjście: maksimum uzyskanych wartości, minimum uzyskanych wartości

Metoda: minimax(stan *s*, gracz *g*, poziom *k*): ocena stanu *s* z punktu widzenia gracza *g*

1. jeśli *s* jest stanem końcowym, zwróć użyteczność stanu *s*;
2. jeśli *k* przekracza maksymalną głębokość przeszukiwania, zwróć heurystyczną ocenę stanu *s*;
3. oblicz minimax(*s'*, *g*, *k*+1) dla wszystkich możliwych stanów następnych *s'*;
4. jeśli w stanie *s* ruch wykonuje gracz *g*, zwróć maksimum uzyskanych wartości;
5. jeśli w stanie *s* ruch wykonuje przeciwnik gracza *g*, zwróć minimum uzyskanych wartości.

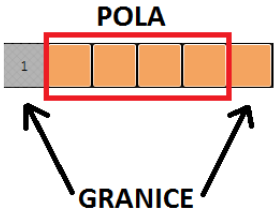
Algorytm heurystycznej oceny stanu gry:

Algorytm ten polega na przeglądaniu tablicy reprezentującej stan gry w 4 kierunkach:
- poziomo
- pionowo
- ukośnie, z góry na dół (ku prawej stronie)
- ukośnie, z dołu do góry (ku prawej stronie)



Przeglądając tablicę w każdy z wymienionych sposobów, brane pod uwagę są wszelkie możliwe „szóstki", składające się z kolejnych sześciu pól na planszy (z uwzględnieniem obramowania). Szóstki mogą nachodzić na siebie oraz się krzyżować.

Każda z utworzonych szóstek, składa się z 4 pól oraz obramowania (granic).



Granicami nazywamy dwa skrajne pola (pole pierwsze oraz szóste).

Wszystkie możliwe do utworzenia szóstki poddawane są ocenie. Ocena ta przyjmuje wartości z zakresu od 0 do 100, i uzależniona jest od ilości pionków gracza oraz ilości wypełnionych granic w szóstce.

Przykładowa szóstka z trzema pionkami gracza oraz z nie wypełnioną żadną granicą:

Przykładowa szóstka z trzema pionkami gracza oraz z nie wypełnioną jedną granicą:

Wartości liczbowe oceny, nadawane są szóstkom na podstawie tabeli:

L.Pionków	L.Granic	Ocena
0	0	**0**
0	1	**1**
0	2	**2**
1	0	**2**
1	1	**3**
1	2	**5**
2	0	**15**
2	1	**18**
2	2	**20**
3	0	**40**
3	1	**45**
3	2	**50**
4	0	**80**
4	1	**90**
4	2	**100**

gdzie *liczba pionków*, to ilość występujących na polach pionków gracza, a *liczba granic*, to ilość wypełnionych (przez obramowanie planszy lub pionki przeciwnika) granic.

Niezależnie od ustawień oraz od ilości ustawionych pionków gracza, stan otrzymuje ocenę równą 0, w przypadku gdy:

- na którymkolwiek z pól znajduje się pionek przeciwnika
- na którejkolwiek z granic znajduje się pionek gracza (potencjalne wystąpienie pięciu pionków pod rząd)

Po dokonaniu oceny wszystkich szóstek, ich wartości liczbowe są sumowane. Daje to wartość *OcenyGracza*. W taki sam sposób, oceniana jest użyteczność stanu dla przeciwnika (*OcenaPrzeciwnika*).

Ocenę użyteczności z punktu widzenia gracza, obliczamy wg wzoru:

OcenaKoncowa = OcenaGracza - OcenaPrzeciwnika

3. Języki programowania, narzędzia, środowisko implementacji

Aplikacja została wykonana w języku C#/.NET Framework 4.0, w środowisku Visual Studio 2012 z wykorzystaniem interfejsu programowania aplikacji jakim jest Windows Presentation Foundation (WPF). WPF dostarcza język opisu interfejsu – XAML, dzięki któremu można tworzyć rozbudowane graficznie interfejsy użytkownika.

Sztuczna inteligencja aplikacji stworzona została w języku programowania logicznego Prolog, z wykorzystaniem środowiska i interfejsu SWI-Prolog.

Całość połączona została za pomocą biblioteki dostępnej w środowisku Visual Studio – Swi-cs-pl (*SbsSW.SwiPlCs*).

III. Podział prac

Autor	**Podzadanie**
Marcin Gmyz	Wykonanie interfejsu graficznego aplikacji, opracowanie i implementacja algorytmu odpowiedzialnego za zliczanie ilości zdobytych przez graczy punktów oraz algorytmu kontrolującego przebieg rozgrywki (niedozwolone ruchy) w języku C#, testy aplikacji
Rafał Nowicki	Wykonanie prologowej części aplikacji (implementacja algorytmu odciętego mini-maksu, opracowanie i implementacja funkcji heurystycznej odpowiedzialnej za zwracanie użyteczności określonego stanu gry), testy aplikacji
Łukasz Szabelski	Połączenie prologowej części aplikacji oraz interfejsu graficznego w języku C# z wykorzystaniem biblioteki Swi-cs-pl, nadzór i pomoc w wykonywaniu prologowej części aplikacji (w szczególności przy funkcji oceny), testy aplikacji

IV. Opis implementacji

1. Struktury danych

Podstawową strukturą danych używaną przy implementacji jest dwuwymiarowa tablica obiektów *pointsArr*, zawierająca dane dotyczące rozmieszczenia pionków na planszy. Tablica ta składa się z obiektów *Point* zawierających pola:

- *X* oraz *Y* – współrzędne określające położenie pola na planszy,
- *type* – określające typ pola znajdującego się na planszy, odpowiednio:
  - 0 – pole wolne,
  - 1 – pole zajęte przez gracza pierwszego,
  - 2 – pole zajęte przez gracza drugiego,
  - 3 – pole ograniczające planszę – *ściana.*

Całość polega więc na zmianie typu pola w tablicy, w momencie wykonywania ruchu przez jednego z graczy (lub komputer).
Do rysowania planszy na ekranie używana jest kontrolka *Canvas*, dzięki której na podstawie współrzędnych można wyrysować przyciski w odpowiednich miejscach. W zależności od typu pola, wyświetlany jest odpowiedni przycisk.

Jak wcześniej wspomniano, do utworzenia drzewa gry i wybrania na jego podstawie kolejnego ruchu komputera zastosowaliśmy algorytm obciętego mini-maksu. Podstawową trudnością przy implementacji mini-maksu jest określenie reprezentacji stanu gry. W przypadku gry Freedom, za stan należy rozumieć aktualne rozmieszczenie pionków na planszy, współrzędne ostatniego ruchu, oraz to, który z graczy aktualnie wykonuje ruch. O ile reprezentacja aktualnego gracza i współrzędnych ostatniego ruchu są zadaniem trywialnym, to reprezentacja rozmieszczenia pionków na planszy była problemem nieco bardziej złożonym. Ostatecznie, struktura ta reprezentowana jest w prologu jako lista wierszy. Wiersz natomiast jest listą elementów reprezentujących pola rzeczywistej planszy. Każde z pól może przyjmować wartości od 0 do 3, przy czym 0 oznacza pole puste, 3 pole będące obramowaniem planszy, a 1 i 2 odpowiedniego gracza. Przykład reprezentacji planszy:

[[3,3,3,3,3,3],[3,0,0,0,2,3],[3,0,1,1,0,3],[3,0,0,0,2,3],[3,0,1,0,0,3],[3,3,3,3,3,3]]

2. Predykaty zdefiniowane w prologowej części aplikacji

W nawiasach opisana jest lista argumentów, jakie funkcje przyjmują (oznaczone poprzez znak ,,-"), oraz zwracane wyniki (oznaczone poprzez znak ,,+").

Podczas przedstawiania argumentów wykorzystywane są stwierdzenia takie jak:

- **Tablica punktów** - jest to lista, za pomocą której reprezentowany jest aktualny stan rozmieszczenia pionków na planszy. Lista ta składa się z podlist zawierających elementy. Element jest odpowiednikiem jednego pola na planszy i może przyjmować wartości z zakresu od 0 do 3.

  np. Plansza o następującym ustawieniu pionków

  [3,3,3,3,3,3]
  [3,0,0,0,2,3]
  [3,0,1,1,0,3]
  [3,0,0,0,2,3]
  [3,0,1,0,0,3]
  [3,3,3,3,3,3]

  Reprezentowana jest w postaci

  [[3,3,3,3,3,3],[3,0,0,0,2,3],[3,0,1,1,0,3],[3,0,0,0,2,3],[3,0,1,0,0,3],[3,3,3,3,3,3]]

- **Szóstka** - jest to lista składająca się z 6 kolejnych elementów znajdujących się w jednym wierszu

- **Ocena użyteczności stanu gry** - Jest to wartość liczbowa, reprezentująca ,,atrakcyjność"- z punktu widzenia określonego gracza - rozmieszczenia pionków na planszy (stanu gry). Im wyższa, tym osiągnięcie takiego stanu jest bardziej pożądana.

Predykaty składające się na funkcję heurystyczną odpowiedzialną za zwracanie użyteczności określonego stanu gry

2.1. ilePkt(+Szóstka punktów, +Aktualny gracz, -OCENA STANU).

+ Szóstka punktów - Lista składająca się z 6 elementów, przyjmujących wartości od 0 do 3 (np. [3,0,0,1,1,2])

+ Aktualny gracz - Wartość liczbowa przyjmująca wartości 1 lub 2 (np. 1)

- Ocena stanu - Liczbowa reprezentacja użyteczności stanu. (np. 75)

Jest to predykat, który odpowiada za przeglądanie podanej na wejściu tablicy składającej się z reprezentacji pionków na 6 kolejnych polach planszy. Jego funkcja polega na zliczeniu pionków należących do gracza, sprawdzeniu czy między pionkami gracza nie występują pionki przeciwnika, oraz z ilu stron potencjalna „czwórka" jest otoczona innymi pionkami (czy występują granice opisane w punkcie II.2.). Po wykonaniu zliczenia, wykonana zostaje ocena użyteczności ustawienia na planszy w/w 6 pionków.

Predykat ten, pomocniczo korzysta z predykatów punkty(…), oraz punkty2(…), które przyjmują następujące dodatkowe argumenty:

- **punkty(+Szóstka punktów, +Aktualny gracz, +Przeciwnik, - OCENA STANU)**

  + **Przeciwnik** - Wartość liczbowa przyjmująca wartości 1 lub 2 (np. 1)

- **punkty2(+Szóstka punktów, +Aktualny gracz, +Przeciwnik, +Pkt. za granice, +Pkt. za żetony, - OCENA STANU)**

  + **Pkt. za granice** - Wartość liczbowa przyjmująca wartości od 0 do 2 (np. 2)

  + **Pkt. za żetony** - Wartość liczbowa przyjmująca wartości od 0 do 4 (np. 3)

2.2. ocen(+Pkt. za granice, +Liczba pkt za żetony, -Wynik)

+ **Pkt. za granice** - Wartość liczbowa przyjmująca wartości od 0 do 2 (np. 2)

+ **Pkt. za żetony** - Wartość liczbowa przyjmująca wartości od 0 do 4 (np. 3)

- Wynik - Liczbowa reprezentacja użyteczności stanu. (np. 35)

Predykat ten służy do nadania wartości liczbowej (oceny), na podstawie ilości występujących w ocenianej szóstce pionków gracza, oraz ilości występujących granic.

2.3. budujSzesc(+Tablica punktów, -Szóstka wynikowa)

+ **Tablica punktów** - Lista reprezentująca aktualny stan planszy. Jej struktura opisana jest w punkcie IV.2.

- **Szóstka wynikowa** - Lista składająca się z 6 elementów, przyjmujących wartości od 0 do 3 (np. [3,0,0,1,1,2])

Jest to predykat służący do tworzenia listy kolejnych sześciu elementów planszy, począwszy od pierwszego elementu występującego w przekazanej mu jako argument Tablicy punktów. Lista tworzona jest z elementów pierwszego wiersza w/w Tablicy punktów.

2.4 poziomo(+Tablica punktów, +Zakres X, +Zakres Y, +Aktualny gracz, -Wynik)

+ **Tablica punktów** - Lista reprezentująca aktualny stan planszy. Jej struktura opisana jest w punkcie IV.2.

+ **Zakres X** - Liczba - określa ile wierszy należy przejrzeć przy budowaniu szóstek (np. 8)

+ **Zakres Y** - Liczba - określa ile elementów należy przejrzeć przy budowaniu szóstek (np.4)

+ **Aktualny gracz** - Wartość liczbowa przyjmująca wartości 1 lub 2 (np. 1)

- **Wynik** - Liczbowa reprezentacja składowej poziomej oceny użyteczności stanu gry (np. 24)

Predykat ten służy do oceniania składowej poziomej oceny stanu gry. Przeglądane są wszystkie możliwe szóstki ułożone poziomo na planszy i na podstawie ich indywidualnych ocen wyliczana jest składowa pozioma końcowej oceny stanu gry.

Pomocniczo predykat ten korzysta z predykatu sprawdzWiersz( ), oraz dajWartoscGlowy( ), które przyjmują niektóre z argumentów opisanych powyżej.

2.5. zamienWierKol(+Tablica punktów, -Tablica po transformacji)

+ **Tablica punktów** - Lista reprezentująca aktualny stan planszy. Jej struktura opisana jest w punkcie IV.2.

- **Tablica po transformacji** - Lista, o strukturze identycznej jak Tablica punktów

Jest to predykat, który jest odpowiedzialny na zamianę kolumn i wierszy w tablicy reprezentującej rozmieszczenie pionków na planszy. Jego zastosowanie pozwala na obliczenie składowej pionowej użyteczności stanu gry.

2.6. pionowo(+Tablica punktów, +Zakres X, +ZakresY, +Aktualny gracz, -WYNIK)

+ **Tablica punktów** - Lista reprezentująca aktualny stan planszy. Jej struktura opisana jest w punkcie IV.2.

+ **Zakres X** - Liczba - określa ile wierszy należy przejrzeć przy budowaniu szóstek (np. 8)

+ **Zakres Y** - Liczba - określa ile elementów należy przejrzeć przy budowaniu szóstek (np.4)

+ **Aktualny gracz** - Wartość liczbowa przyjmująca wartości 1 lub 2 (np. 1)

- **Wynik** - Liczbowa reprezentacja składowej poziomej oceny użyteczności stanu gry (np. 24)

Predykat ten oblicza składową pionową końcowej oceny użyteczności stanu gry. Wykorzystuje on transformację tablicy punktów, a następnie poddaje przetransformowaną tablicę ocenie identycznej, jak przy obliczaniu składowej poziomej użyteczności stanu gry.

2.7. skosOdLewej(+Tablica punktów, + Aktualny gracz, - Wynik)

+ Tablica punktów - Lista reprezentująca aktualny stan planszy. Jej struktura opisana jest w punkcie IV.2.

+ Aktualny gracz - Wartość liczbowa przyjmująca wartości 1 lub 2 (np. 1)

- Wynik - Liczbowa reprezentacja składowej ukośnej oceny użyteczności stanu gry (np. 24)

Predykat ten oblicza składową ukośną (z góry na dół, ku prawej stronie) końcowej oceny użyteczności stanu gry.

2.8. skosOdPrawej(+Tablica punktów, + Aktualny gracz, - Wynik)

+ Tablica punktów - Lista reprezentująca aktualny stan planszy. Jej struktura opisana jest w punkcie IV.2.

+ Aktualny gracz - Wartość liczbowa przyjmująca wartości 1 lub 2 (np. 1)

- Wynik - Liczbowa reprezentacja składowej ukośnej oceny użyteczności stanu gry (np. 24)

Predykat ten oblicza składową ukośną (z dołu na górę, ku prawej stronie) końcowej oceny użyteczności stanu gry.

2.9. sprawdzWymiaryTablicy(+Tablica punktów, -Rozmiar tablicy)

+ Tablica punktów - Lista reprezentująca aktualny stan planszy. Jej struktura opisana jest w punkcie IV.2.

- Rozmiar planszy - Liczba - określa rozmiar kwadratowej planszy (np. 5, przy planszy o wymiarach 5x5)

Jest to pomocniczy predykat wykorzystywany podczas procesu tworzenia drzewa gry oraz oceny użyteczności stanu gry.

2.10 obliczStan(+ Tablica punktów, -Użyteczność stanu)

+ Tablica punktów - Lista reprezentująca aktualny stan planszy. Jej struktura opisana jest w punkcie IV.2.

- Użyteczność stanu - Liczbowa reprezentacja końcowej (całkowitej) użyteczności stanu gry (np. 145)

Predykat ten służy do oceny użyteczności stanu gry. Wykorzystuje on omówione wcześniej predykaty służące do obliczania poszczególnych składowych oceny stanu gry.

Predykaty pomocnicze, oraz odpowiedzialne za tworzenie drzewa gry

2.11. czyRownyElTab(+Tablica punktów, +Współrzędna X , +Współrzędna Y, +Element do porównania)

+ **Tablica punktów** - Lista reprezentująca aktualny stan planszy. Jej struktura opisana jest w punkcie IV.2.

+ **Współrzędna X** - Liczba - Numer wiersza, w którym znajduje się element (np. 3)

+ **Współrzędna Y** - Liczba - Element wiersza, który należy porównań (np. 5)

+ **Element do porównania** - Liczba z zakresu od 0 do 3

Predykat zwraca *true*, jeśli element tablicy punktów o określonych współrzędnych jest równy z elementem porównywanym.

2.12. zamienElTab(+Tablica punktów,+Współrzędna X, +Współrzędna Y, +Element, -Tablica wynikowa)

+ **Tablica punktów** - Lista reprezentująca aktualny stan planszy. Jej struktura opisana jest w punkcie IV.2.

+ **Współrzędna X** - Liczba - Numer wiersza, w którym znajduje się element (np. 3)

+ **Współrzędna Y** - Liczba - Element wiersza, który należy porównań (np. 5)

+ **Element** - Liczba z zakresu od 0 do 3 - określa wartość nadawaną elementowi tablicy

- **Tablica wynikowa** - Lista o strukturze identycznej jak Tablica punktów

Predykat ten służy do zamiany określonego elementu tablicy, na inny, podany w jednym z argumentów.

2.13. czyNiePusty(+Tablica punktów, +Współrzędna X, +Współrzędna Y)

+ **Tablica punktów** - Lista reprezentująca aktualny stan planszy. Jej struktura opisana jest w punkcie IV.2.

+ **Współrzędna X** - Liczba - Numer wiersza, w którym znajduje się element (np. 1)

+ **Współrzędna Y** - Liczba - Element wiersza, który należy porównań (np. 7)

Predykat zwraca *true*, jeśli wartość określonego przez współrzędne elementu jest równa 0, co oznacza, że jest to pole puste na planszy.

2.14. czyPelna(+Tablica punktów)

+ **Tablica punktów** - Lista reprezentująca aktualny stan planszy. Jej struktura opisana jest w punkcie IV.2.

Jest to predykat służący do sprawdzania, czy plansza jest całkowicie zapełniona, co oznacza koniec gry. Zwraca *true*, w przypadku, pełnego wypełnienia planszy.

2.15. czyFreedom(+Tablica punktów, +OstatniX, +OstatniY)

+ **Tablica punktów** - Lista reprezentująca aktualny stan planszy. Jej struktura opisana jest w punkcie IV.2.

+ **Ostatni X -** Liczba reprezentująca współrzędną X ostatniego ruchu (np.2)

+ **Ostatni Y** - Liczba reprezentująca współrzędną Y ostatniego ruchu (np.2)

Predykat ten zwraca *true*, w przypadku, gdy na planszy występuje stan Freedom.

2.16. miniMax(+Tablica punktów, +Xnext, +Ynext, +Aktualny gracz, +Głębokość przeszukiwania, -WYNIK)

+ **Tablica punktów** - Lista reprezentująca aktualny stan planszy. Jej struktura opisana jest w punkcie IV.2.

+ **Xnext** - Liczba - Wspólrzędna X analizowanego ruchu (np. 4)

+ **Ynext** - Liczba - Wspólrzędna Y analizowanego ruchu (np. 8)

+ **Aktualny gracz** - Liczba 1 lub 2, reprezentująca aktualnego gracza

+ **Głębokość przeszukiwania** - Liczba - Przedstawia ilość pozostałych do przeszukania poziomów drzewa gry

- **Wynik** - Liczba - Wartość z funkcji heurystycznej przekazywana wg algorytmu mini-maksu

Predykat ten jest odpowiedzialny za tworzenie drzewa gry poddawanie ocenie użyteczności wszystkich możliwych do wykonania ruchów zgodnie z algorytmem mini-maksu.

2.17. przegladajTablice(+Tablica punktów, +Xod, +Yod, +Xdo, +Ydo, +Xdo, +Aktualny gracz, +Głębokość przeszukiwania,- WYNIK)

+ **Tablica punktów** - Lista reprezentująca aktualny stan planszy. Jej struktura opisana jest w punkcie IV.2

+ **Xod** - liczba - Współrzędna X od której rozpoczynam przeglądanie tablicy (różne wartości w zależności od tego, czy aktualnie jest stan *freedom*.
  - W przypadku wystąpienia *freedom* - Xod = 0,
  - W pozostałych przypadkach Xod = Xostatniego ruchu -1

+**Yod** - liczba - Współrzędna Y od której rozpoczynam przeglądanie Tablicy punktów - analogicznie jak Xod

+ **Xdo** - liczba - Współrzędna X do której przeglądam Tablicę punktów. Przyjmuje wartości:
  - w przypadku wystąpienia *freedom* - Xdo = rozmiar tablicy (np.8x8 - Xdo=8)
  - w pozostałych wypadkach Xdo= Xostatniego ruchu +1

+**Ydo** - liczba - Współrzędna Y od której rozpoczynam przeglądanie Tablicy punktów - analogicznie jak Xdo

+ **Aktualny gracz** - Liczba 1 lub 2, reprezentująca aktualnego gracza

+ **Głębokość przeszukiwania** - Liczba - Przedstawia ilość pozostałych do przeszukania poziomów drzewa gry

- **Wynik** - Liczba - Wartość z funkcji heurystycznej przekazywana wg algorytmu mini-maksu

Predykat ten służy do tworzenia drzewa gry zgodnie z obowiązującymi w grze Freedom zasadami. Uwzględnia wszystkie możliwe do wykonania w danym momencie ruchy, w zależności, czy gra rozgrywka jest w stanie *freedom*, czy w nim nie jest.

2.18. dlaJakich(+Tablica punktów, +X od, +Y od, +X do, +Y do, +X do, +Głębokość przeglądania, +MaxWartosc, -Xwybr, -Ywybr)

+ **Tablica punktów** - Lista reprezentująca aktualny stan planszy. Jej struktura opisana jest w punkcie IV.2

+ **Xod** - liczba - Współrzędna X od której rozpoczynam przeglądanie tablicy (różne wartości w zależności od tego, czy aktualnie jest stan *freedom*.
  - W przypadku wystąpienia *freedom* - Xod = 0,
  - W pozostałych przypadkach Xod = Xostatniego ruchu -1

+**Yod** - liczba - Współrzędna Y od której rozpoczynam przeglądanie Tablicy punktów - analogicznie jak Xod

+ **Xdo** - liczba - Współrzędna X do której przeglądam Tablicę punktów. Przyjmuje wartości:
  - w przypadku wystąpienia *freedom* - Xdo = rozmiar tablicy (np.8x8 - Xdo=8)
  - w pozostałych wypadkach Xdo= Xostatniego ruchu +1

+**Ydo** - liczba - Współrzędna Y od której rozpoczynam przeglądanie Tablicy punktów - analogicznie jak Xdo

+ **Głębokość przeszukiwania** - Liczba - Przedstawia ilość pozostałych do przeszukania poziomów drzewa gry

+ **MaxWartosc** - liczba - maksymalna uzyskana z mini-maksu wartość (np. 215)

- **Xwybr** - liczba - Współrzędna X ruchu, dla którego wartość mini-maksu była największa

- **Ywybr** - liczba - Współrzędna Y ruchu, dla którego wartość mini-maksu była największa

Predykat ten wykorzystany jest do ustalenia dla którego z następnych, możliwych do wykonania ruchów wartość zwrócona przez algorytm mini-maksu jest największa.

2.19. wybierzNastepnyX(+Tablica punktów, +X ostatniego ruchu, +Y ostatniego ruchu, +Głębokość przeszukiwania, +WybranyX)

+ **Tablica punktów** - Lista reprezentująca aktualny stan planszy. Jej struktura opisana jest w punkcie IV.2

+ **X ostatniego ruchu** - Współrzędna X ostatniego wykonanego ruchu (np.3)

+**Y ostatniego ruchu** - Współrzędna Y ostatniego wykonanego ruchu (np.5)

+ **Głębokość przeszukiwania** - Liczba - Przedstawia ilość pozostałych do przeszukania poziomów drzewa gry

- **WybranyX** - Liczba reprezentująca współrzędną X ruchu wybranego przez komputer

Jest to predykat zwracający wartość współrzędnej X kolejnego ruchu, który wykonuje komputer.

2.20. wybierzNastepnyY(+Tablica punktów, +X ostatniego ruchu, +Y ostatniego ruchu, +Głębokość przeszukiwania, +WybranyY)

+ **Tablica punktów** - Lista reprezentująca aktualny stan planszy. Jej struktura opisana jest w punkcie IV.2

+ **X ostatniego ruchu** - Współrzędna X ostatniego wykonanego ruchu (np.3)

+**Y ostatniego ruchu** - Współrzędna Y ostatniego wykonanego ruchu (np.5)

+ **Głębokość przeszukiwania** - Liczba - Przedstawia ilość pozostałych do przeszukania poziomów drzewa gry

- **WybranyY** - Liczba reprezentująca współrzędną Y ruchu wybranego przez komputer

Jest to predykat zwracający wartość współrzędnej Y kolejnego ruchu, który wykonuje komputer.

3. Połączenie między językami C# i Prolog

Obie platformy spotykają się ze sobą tylko w jednym miejscu w programie. Możliwe jest to, dzięki bibliotece ***Swi-cs-pl*** *(SbsSW.SwiPlCs)*, która dostarcza interfejs C# dla języka Prolog i daje możliwość całkowitego rozdzielenia interfejsu użytkownika oraz części odpowiedzialnej za ruch gracza, od części odpowiedzialnej za wyznaczenie kolejnego ruchu gracza komputerowego. Instalacja biblioteki polega na pobraniu kilku plików *.dll* i dołączeniu ich jako referencji do projektu w środowisku .NET.

Po wykonaniu ruchu przez rzeczywistego gracza, formułowane jest zapytanie dla interfejsu prologowego, które składa się z: reprezentacji tablicy gry, współrzędnych ostatniego ruchu oraz wyznaczonej głębokości przeszukiwania.

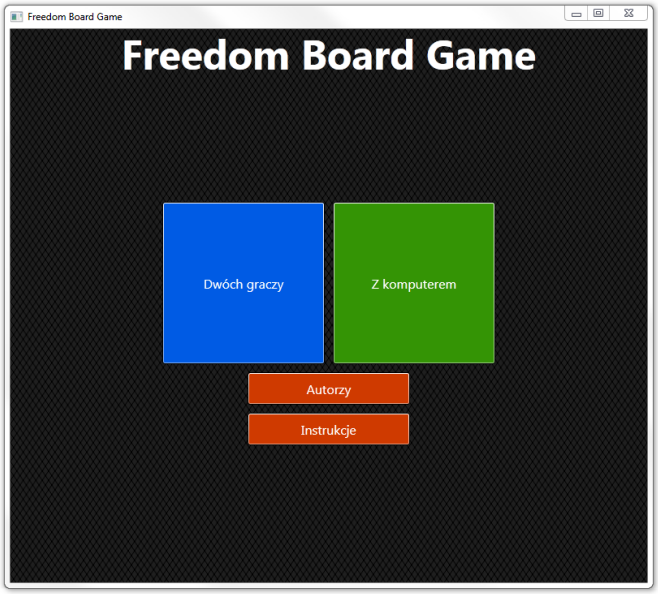
Zapytanie to wywołuje predykaty *wybierzNastepnyX*, oraz *wybierzNastepnyY*, które zwracają 2 wartości liczbowe, reprezentujące współrzędne wybranego przez komputer ruchu. Następnie współrzędne te interpretowane są przez interfejs użytkownika, który umieszcza na planszy pionek gracza komputerowego.

Funkcja w języku C# (*ComputerMoveAi()*), odpowiedzialna za formułowanie zapytania i korzystająca z biblioteki *Swi-cs-pl*, znajduje się w kodzie źródłowym (punkt VI.).

V. Użytkowanie i testowanie systemu

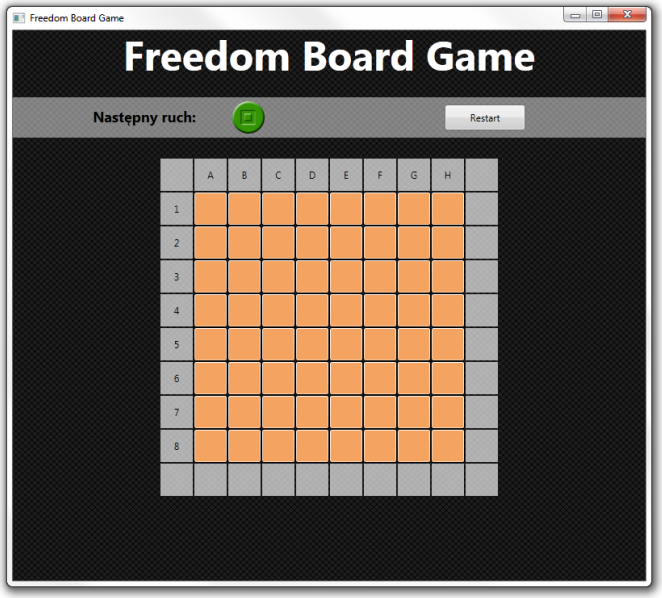
Instalacja gry odbywa się poprzez uruchomienie pliku InstalujFreedom.exe. Stworzony instalator sprawdza czy na komputerze zainstalowane jest środowisko .NET 4.0. W razie jego braku rozpoczyna się jego pobieranie i instalacja. Gdy .NET jest zainstalowany, kreator przeprowadza użytkownika przez proces instalacji. Po jego ukończeniu na pulpicie pojawia się skrót, za pomocą którego można uruchomić grę.

Po załadowaniu gry użytkownikowi ukazuje się menu główne. Z tego miejsca istnieje możliwość wyboru trybu gry. Do dyspozycji są dwa tryby: gra pomiędzy dwoma użytkownikami wykonującymi naprzemiennie swoje ruchy oraz rozgrywka z zaprogramowaną sztuczną inteligencją. Ponadto gracz może dowiedzieć się kto jest pomysłodawcą gry, a kto twórcą aplikacji, czy zapoznać się z regułami gry. Dokonuje się to poprzez kliknięcie odpowiednio podpisanego przycisku. Prostota aplikacji i łatwe zasady gry sprawiają, że obsługa jest bardzo intuicyjna.



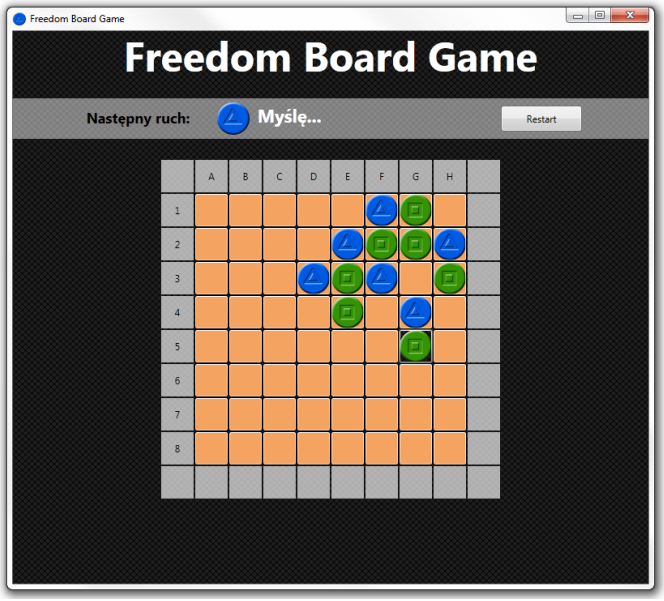
Rys. 1 Menu główne gry.

W oknie rozgrywki widoczna jest plansza oraz informacja o tym, kto wykonuje następny ruch. Domyślnie w trybie gry z komputerem gracz otrzymuje pionki koloru zielonego. Postawienie pionka na planszy dokonuje się poprzez kliknięcie odpowiedniego pola. Gracz ma możliwość ukończenia rozgrywki i powrotu do głównego menu poprzez wciśnięcie przycisku „Restart”.



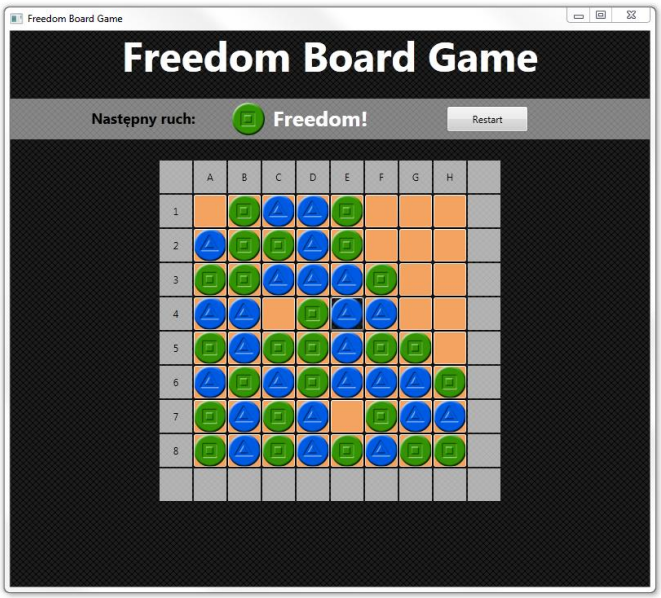
Rys. 2 Okno rozgrywki.

Gdy gracz postawi na planszy swój pionek, następuje oczekiwanie na ruch komputera. Taka sytuacja jest sygnalizowana etykietą „Myślę…”.



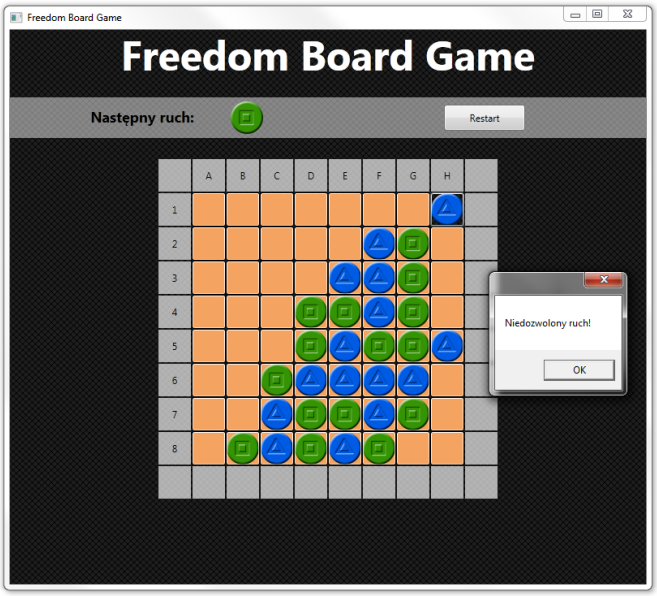
Rys. 3 Informacja o planowaniu ruchu przez komputer.

Podczas gry użytkownik aplikacji jest pilnowany aby stawiał pionki w dozwolonych zasadami miejscach. Informuje go o tym zacienienie pola, na którym przeciwnik postawił ostatni pionek. Zgodnie z zasadami ruch dozwolony jest tylko na polu sąsiadującym z ostatnim ruchem przeciwnika. W szczególnym wypadku, gdy wszystkie pola sąsiadujące z ostatnim ruchem przeciwnika są zajęte, gracz otrzymuje możliwość postawienia pionka na dowolnym wolnym polu na planszy. Jest o tym informowany etykietą „Freedom!”.



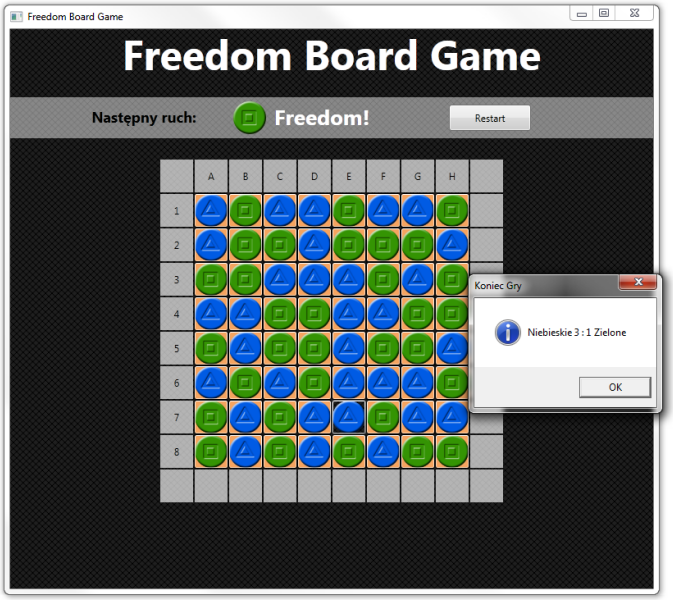
Rys. 3 Rozgrywka w toku, widoczna etykieta „Freedom!”.

Przy próbie wykonania ruchu niedozwolonego w rozumieniu zasad gry wyświetlany jest stosowny komunikat.

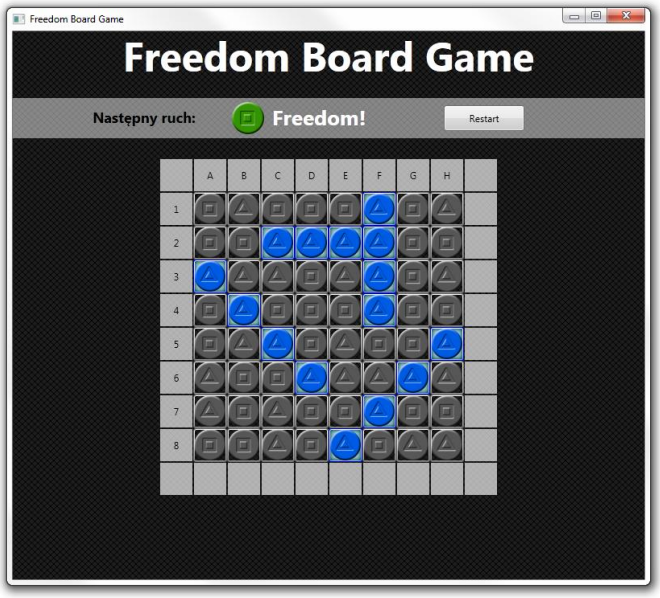


Rys. 4 Próba wykonania niedozwolonego ruchu

Po zakończeniu rozgrywki wyświetlany jest jej wynik, a po jego zatwierdzeniu na planszy pokazane są tylko punktowane pionki.



Rys. 5 Wynik rozgrywki



Rys. 6 Podświetlenie punktowanych pionków

Podczas pracy nad aplikacją była ona testowana głównie pod kątem działania algorytmu wyboru najlepszego możliwego ruchu oraz algorytmu liczenia punktów. Testy były przeprowadzane wielokrotnie przy implementowaniu algorytmu oceny ruchu. Każda nowa wersja była sprawdzana pod kątem prawidłowego działania oraz współpracy ze stworzonym interfejsem graficznym. Po wielu optymalizacjach zastosowanych funkcji uzyskany został końcowy satysfakcjonujący efekt. Zaprogramowana sztuczna inteligencja bardzo dobrze radzi sobie z ogrywaniem człowieka, a punkty liczone są poprawnie.

VI. Tekst programu

Główna klasa ***GameClass*** odpowiadająca za rozgrywkę, zawierająca funkcję *playGame* – odpowiadająca za kontrolowanie wykonywanego ruchu, *doMovement* – odpowiadająca za wykonanie ruchu wybranego przez gracza, *cheackGameOver* – sprawdzająca zapełnienie planszy pionkami (koniec gry), *checkFreedom* – sprawdzająca czy gracz otrzymał przywilej Freedom, *checkResult* – zawierająca algorytm sprawdzający ilość punktów zdobytych przez graczy, *odpal prolog* – odpowiadająca za połączenie Prologa z C#:

```
class GameClass
    {
        public void playGame(int coordX, int coordY, int whichPlayer)
        {
            try
            {
                //w zaleznosci od tego czy gracz ma przywilej 'freedom' - dajacy
mozliwosc postawienia pionka gdziekolwiek,
                //sprawdzany jest odpowiedni warunek. Jesli nie ma 'freedom' - aby
wykonac ruch pole musi byc puste i byc w sasiedztwie ostatniego ruchu.
                //Jesli jest 'freedom' - sprawdzane jest tylko czy pole jest puste
                switch (HelperClass.isFreedom)
                {
                    case false:
                        if (HelperClass.pointsArr[coordX, coordY].type != 0 ||
                            coordX > HelperClass.lastMove.CX + 1 || coordY >
HelperClass.lastMove.CY + 1 ||
                            coordX < HelperClass.lastMove.CX - 1 || coordY <
HelperClass.lastMove.CY - 1)
                        {
                            MessageBox.Show("Niedozwolony ruch!");
                            return;
                        }
                        else
                        {
                            doMovement(coordX, coordY, whichPlayer);
                            // checkResult(whichPlayer);
                        }
                        break;
                    case true:
                        if (HelperClass.pointsArr[coordX, coordY].type != 0)
                        {
                            MessageBox.Show("Niedozwolony ruch!");
                            return;
                        }
                        else
                        {
                            doMovement(coordX, coordY, whichPlayer);
                            // checkResult(whichPlayer);
                        }
                        break;
                    default:
                        break;
                }
            }
            catch
            {
            }
            //sprawdzenie czy nacisnieto przycisk, ktory jest juz 'zajety' lub czy
wcisniety przycisk jest w sasiedztwie 'ostatniego ruchu'
```

```
        }
        private void doMovement(int coordX, int coordY, int whichPlayer)
        {
            //w zaleznosci, ktory gracz wcisnal button, taki typ jest przypisywany
            if (whichPlayer == 1)
            {
                HelperClass.pointsArr[coordX, coordY].type = 1;
                HelperClass.lastMove = new Coordinates(coordX, coordY);
                HelperClass.nowMove = 2;

                HelperClass.isFreedom = checkFreedom(coordX, coordY);
                //MessageBox.Show(isFreedom.ToString());
            }
            else if (whichPlayer == 2)
            {
                switch (HelperClass.gameType)
                {
                    case HelperClass.GameType.twoPlayers:
                        HelperClass.pointsArr[coordX, coordY].type = 2;
                        HelperClass.lastMove = new Coordinates(coordX, coordY);
                        HelperClass.nowMove = 1;

                        HelperClass.isFreedom = checkFreedom(coordX, coordY);
                        break;
                    case HelperClass.GameType.withComputer:
                        HelperClass.pointsArr[coordX, coordY].type = 2;
                        HelperClass.lastMove = new Coordinates(coordX, coordY);
                        HelperClass.nowMove = 1;

                        HelperClass.isFreedom = checkFreedom(coordX, coordY);

                        ComputerMoveAI();//obliczanie ruchu komputera

                        HelperClass.pointsArr[HelperClass.nextMoveX,
HelperClass.nextMoveY].type = 1;
                        HelperClass.lastMove = new Coordinates(HelperClass.nextMoveX,
HelperClass.nextMoveY);
                        HelperClass.nowMove = 2;

                        HelperClass.isFreedom = checkFreedom(HelperClass.nextMoveX,
HelperClass.nextMoveY);
                        break;
                    default:
                        break;
                }


            }
        }
        public bool checkGameOver()
        {
            bool gameOver = true;

            foreach (Point p in HelperClass.pointsArr)
            {
                if (p.type == 0)
                {
                    gameOver = false;
                }
            }

            if (gameOver == true)
            {
                checkResult(1);
```

```
                checkResult(2);
                MessageBox.Show("Niebieskie " + HelperClass.scorePlayer1 + " : " +
HelperClass.scorePlayer2 + " Zielone", "Koniec Gry", MessageBoxButton.OK,
MessageBoxImage.Information);
            }

            return gameOver;
        }
        public bool checkFreedom(int coordX, int coordY)
        {
            try
            {
                if (HelperClass.pointsArr[coordX, coordY - 1].type != 0 &&
HelperClass.pointsArr[coordX + 1, coordY - 1].type != 0 &&
                    HelperClass.pointsArr[coordX + 1, coordY].type != 0 &&
HelperClass.pointsArr[coordX + 1, coordY + 1].type != 0 &&
                    HelperClass.pointsArr[coordX, coordY + 1].type != 0 &&
HelperClass.pointsArr[coordX - 1, coordY + 1].type != 0 &&
                    HelperClass.pointsArr[coordX - 1, coordY].type != 0 &&
HelperClass.pointsArr[coordX - 1, coordY - 1].type != 0)
                    return true;
                else return false;
            }
            catch (Exception ee)
            {
                return false;
            }
        }

        public void checkResult(int player)
        {
            for (int i = 0; i < HelperClass.boardWidth + 2; i++)
            {
                for (int j = 0; j < HelperClass.boardHeight + 2; j++)
                {
                    //prawko skos (od gory w dol) -> \
                    if (HelperClass.pointsArr[i, j].type == player &&
HelperClass.pointsArr[i + 1, j + 1].type == player &&
                        HelperClass.pointsArr[i + 2, j + 2].type == player &&
HelperClass.pointsArr[i + 3, j + 3].type == player &&
                        HelperClass.pointsArr[i - 1, j - 1].type != player &&
HelperClass.pointsArr[i + 4, j + 4].type != player &&
                        HelperClass.tempArr[i, j] != player) //ostatni warunek to
sprawdzenie w tymczasowej tablicy czy dana czworka byla juz sprawdzana, zeby nie
zliczyc dwa razy tej samej czworki (od tego samego pionka)
                    {
                        HelperClass.tempArr[i, j] = player; HelperClass.tempArr[i + 1,
j + 1] = player; HelperClass.tempArr[i + 2, j + 2] = player; HelperClass.tempArr[i +
3, j + 3] = player;
                        if (player == 1)
                            HelperClass.scorePlayer1++;
                        else HelperClass.scorePlayer2++;
                        //  MessageBox.Show(HelperClass.scorePlayer1.ToString());
                    }
                    //lewo skos (od gory w dol) -> /
                    if (HelperClass.pointsArr[i, j].type == player &&
HelperClass.pointsArr[i - 1, j + 1].type == player &&
                       HelperClass.pointsArr[i - 2, j + 2].type == player &&
HelperClass.pointsArr[i - 3, j + 3].type == player &&
                       HelperClass.pointsArr[i + 1, j - 1].type != player &&
HelperClass.pointsArr[i - 4, j + 4].type != player &&
                       HelperClass.tempArr[i, j] != player)
                    {
```

```
                        HelperClass.tempArr[i, j] = player; HelperClass.tempArr[i - 1,
j + 1] = player; HelperClass.tempArr[i - 2, j + 2] = player; HelperClass.tempArr[i -
3, j + 3] = player;
                        if (player == 1)
                            HelperClass.scorePlayer1++;
                        else HelperClass.scorePlayer2++;
                        //  MessageBox.Show(HelperClass.scorePlayer1.ToString());
                    }
                    //poziom -> -
                    if (HelperClass.pointsArr[i, j].type == player &&
HelperClass.pointsArr[i + 1, j].type == player &&
                       HelperClass.pointsArr[i + 2, j].type == player &&
HelperClass.pointsArr[i + 3, j].type == player &&
                       HelperClass.pointsArr[i - 1, j].type != player &&
HelperClass.pointsArr[i + 4, j].type != player &&
                       HelperClass.tempArr[i, j] != player)
                    {
                        HelperClass.tempArr[i, j] = player; HelperClass.tempArr[i + 1,
j] = player; HelperClass.tempArr[i + 2, j] = player; HelperClass.tempArr[i + 3, j] =
player;
                        if (player == 1)
                            HelperClass.scorePlayer1++;
                        else HelperClass.scorePlayer2++;
                        // MessageBox.Show(HelperClass.scorePlayer1.ToString());
                    }
                    //pion -> |     - z dołu do góru
                    if (HelperClass.pointsArr[i, j].type == player &&
HelperClass.pointsArr[i, j - 1].type == player &&
                       HelperClass.pointsArr[i, j - 2].type == player &&
HelperClass.pointsArr[i, j - 3].type == player &&
                       HelperClass.pointsArr[i, j + 1].type != player &&
HelperClass.pointsArr[i, j - 4].type != player &&
                       HelperClass.tempArr[i, j] != player)
                    {
                        HelperClass.tempArr[i, j] = player; HelperClass.tempArr[i, j -
1] = player; HelperClass.tempArr[i, j - 2] = player; HelperClass.tempArr[i, j -3] =
player;
                        if (player == 1)
                            HelperClass.scorePlayer1++;
                        else HelperClass.scorePlayer2++;
                        // MessageBox.Show(HelperClass.scorePlayer1.ToString());
                    }
                    //pion -> |    - z góry do dołu
                    if (HelperClass.pointsArr[i, j].type == player &&
HelperClass.pointsArr[i, j + 1].type == player &&
                       HelperClass.pointsArr[i, j + 2].type == player &&
HelperClass.pointsArr[i, j + 3].type == player &&
                       HelperClass.pointsArr[i, j - 1].type != player &&
HelperClass.pointsArr[i, j + 4].type != player &&
                       HelperClass.tempArr[i, j] != player)
                    {
                        HelperClass.tempArr[i, j] = player; HelperClass.tempArr[i, j +
1] = player; HelperClass.tempArr[i, j + 2] = player; HelperClass.tempArr[i, j + 3] =
player;
                        if (player == 1)
                            HelperClass.scorePlayer1++;
                        else HelperClass.scorePlayer2++;
                        // MessageBox.Show(HelperClass.scorePlayer1.ToString());
                    }
                }
            }
        }
        public static void OdpalProlog()
        {
```

```
            if (!PlEngine.IsInitialized)
            {
              String[] param = { "-q", "-f", @"FreedomProlog.txt" };
              PlEngine.Initialize(param);
            }
        }

}
```

Klasa pomocnicza *HelperClass*, zawierająca potrzebne struktury danych:

```
class HelperClass
    {
        public static int nowMove = 1; //kto teraz rusza: 1 - gracz 1, 2 - gracz 2
        public static List<Point> points = new List<Point>();
        public static int boardWidth = 8;
        public static int boardHeight = 8;
        public static Point[,] pointsArr = new Point[boardWidth + 2, boardHeight + 2];
        public static int[,] tempArr = new int[HelperClass.boardWidth + 2,
HelperClass.boardHeight + 2];
        public static Coordinates lastMove;
        public static bool isFreedom;
        public static int scorePlayer1, scorePlayer2;
        public static int glebokoscPrzeszukiwania = 6;
        public static int nextMoveX, nextMoveY;
        public static bool czyOdpalonyProlog = false;
        public enum GameType { twoPlayers, withComputer };
        public static GameType gameType;
    }

    class Coordinates //klasa do przekazywania wspolrzednych buttona po jego
wcisnieciu
    {
        public int CX;
        public int CY;
        public Coordinates(int x, int y)
        {
            this.CX = x;
            this.CY = y;
        }
    }
    class Point //klasa okreslajaca wspolrzedne i typ buttona przechowywanego w
tablicy pointsArr
    {
        public int X;
        public int Y;
        public int type;
        public Point(int x, int y, int type)
        {
            this.X = x;
            this.Y = y;
            this.type = type;
        }

    }
```

Klasa *MainWindow* zawierająca implementację interfejsu graficznego:

```
public partial class MainWindow : Window
    {
        public int buttonSize;
        private bool firstMove;
        GameClass game = new GameClass();
        public MainWindow()
        {
            InitializeComponent();
            HelperClass.nowMove = 2;
            firstMove = true;

            buttonSize = 42;

            //generowanie tablicy ze wspolrzednymi punktow wg. ktorej rysowane sa
buttony, mozna ustawic dowolne wymiary w klasie pomocniczej HelperClass
            whoMove();
            startOrRestart();
            draw();
        }
        void startOrRestart()
        {
            //generowanie tablicy ze wspolrzednymi punktow wg. ktorej rysowane sa
buttony, mozna ustawic dowolne wymiary w klasie pomocniczej HelperClass
            for (int i = 0; i < HelperClass.boardWidth + 2; i++) // + 2, bo dokladamy
po jednym wierszu i kolumnie z kazdej strony, zeby wiedziec gdzie jest 'sciana/koniec
planszy', do sprawdzania pozniejszego, np. czy wystapilo 'freedom'
            {
                for (int j = 0; j < HelperClass.boardHeight + 2; j++)
                {
                    if (i == HelperClass.boardWidth + 1 || i == 0 || j ==
HelperClass.boardHeight + 1 || j == 0)
                    {
                        HelperClass.pointsArr[i, j] = new Point(i, j, 3);
                    }
                    else HelperClass.pointsArr[i, j] = new Point(i, j, 0);
                }
            }
        }
        private void whoMove()
        {
            if (HelperClass.nowMove == 1)
            {
                whickPlayerTextBlock.Text = "Następny ruch: " ;
                nextMoveImage.Style = Resources["nextMoveBlue"] as Style;
            }
            else if (HelperClass.nowMove == 2)
            {
                whickPlayerTextBlock.Text = "Następny ruch: " ;
                nextMoveImage.Style = Resources["nextMoveGreen"] as Style;
            }
        }
        //metoda sluzaca tylko i wylacznie do rysowania
        private void draw()
        {
            Button b;
            canvas.Children.Clear();
            //lecimy po tablicy punktow i sprawdzamy typ buttona, na podstawie typu
button przyjmuje wyglad
            foreach (Point p in HelperClass.pointsArr)
            {
                if (p.type == 0) //dla buttona nie wcisnietego
                {
```

```
                    b = new Button() { Width = buttonSize, Height = buttonSize };
                    b.Click += b_Click;
                    b.Tag = new Coordinates(p.X, p.Y);
                    b.Background = new
SolidColorBrush(Color.FromArgb(255,244,164,96));
                    b.BorderBrush = new SolidColorBrush(Colors.Black);
                    canvas.Children.Add(b);

                    Canvas.SetLeft(b, p.X * buttonSize + 5);
                    Canvas.SetTop(b, p.Y * buttonSize + 5);
                }
                else if(p.type == 1) //button wcisniety przez gracza 1
                {
                    b = new Button() { Width = buttonSize, Height = buttonSize };
                    b.Click += b_Click;
                    b.Tag = new Coordinates(p.X, p.Y); //taguje button wspolrzednymi,
zeby pozniej wiedziec ktory wcisniety

                    //podpisanie przycisku ktory ostatnio wcisnieto
                    if ((b.Tag as Coordinates).CX == HelperClass.lastMove.CX && (b.Tag
as Coordinates).CY == HelperClass.lastMove.CY)
                    {
                        b.Style = Resources["Player1Last"] as Style;
                    }else b.Style = Resources["Player1"] as Style;

                    canvas.Children.Add(b);

                    Canvas.SetLeft(b, p.X * buttonSize + 5);
                    Canvas.SetTop(b, p.Y * buttonSize + 5);
                }
                else if (p.type == 2) //button wcisniety przez gracza 2
                {
                    b = new Button() { Width = buttonSize, Height = buttonSize };
                    b.Click += b_Click;
                    b.Tag = new Coordinates(p.X, p.Y);

                    if ((b.Tag as Coordinates).CX == HelperClass.lastMove.CX && (b.Tag
as Coordinates).CY == HelperClass.lastMove.CY)
                    {
                        b.Style = Resources["Player2Last"] as Style;
                    }else b.Style = Resources["Player2"] as Style;

                    canvas.Children.Add(b);

                    Canvas.SetLeft(b, p.X * buttonSize + 5);
                    Canvas.SetTop(b, p.Y * buttonSize + 5);
                }
                else if (p.type == 3) //sciana - granice planszy
                {
                    b = new Button() { Width = buttonSize, Height = buttonSize };
                    b.Click += b_Click;
                    b.Tag = new Coordinates(p.X, p.Y);
                    b.IsEnabled = false;
                    b.Style = this.Resources["ButtonWallStyle"] as Style;

                    string letters = "ABCDEFGHIJKLMN";
                    if (p.X == 0 && p.Y > 0 && p.Y < HelperClass.boardHeight + 1)
                        b.Content = p.Y;
                    else if(p.Y == 0 && p.X > 0 && p.X < HelperClass.boardHeight + 1)
                    {
                        b.Content = letters[p.X-1];
                    }

                    b.Background = new SolidColorBrush(Colors.Blue);
```

```
                    canvas.Children.Add(b);

                    Canvas.SetLeft(b, p.X * buttonSize + 5);
                    Canvas.SetTop(b, p.Y * buttonSize + 5);
                }
            }
        }
        private void drawGameOver()
        {
            Button b;
            canvas.Children.Clear();
            //lecimy po tablicy punktow i sprawdzamy typ buttona, na podstawie typu
button przyjmuje wyglad
            foreach (Point p in HelperClass.pointsArr)
            {
                if (p.type == 0) //dla buttona nie wcisnietego
                {
                    b = new Button() { Width = buttonSize, Height = buttonSize };
                    b.Click += b_Click;
                    b.Tag = new Coordinates(p.X, p.Y);
                    b.Background = new SolidColorBrush(Color.FromArgb(255, 244, 164,
96));
                    b.BorderBrush = new SolidColorBrush(Colors.Black);

                    canvas.Children.Add(b);

                    Canvas.SetLeft(b, p.X * buttonSize + 5);
                    Canvas.SetTop(b, p.Y * buttonSize + 5);
                }
                else if (p.type == 1) //button wcisniety przez gracza 1
                {
                    b = new Button() { Width = buttonSize, Height = buttonSize };
                    b.Click += b_Click;
                    b.Tag = new Coordinates(p.X, p.Y); //taguje button wspolrzednymi,
zeby pozniej wiedziec ktory wcisniety

                    //podpisanie przycisku ktory ostatnio wcisnieto
                    if (p.type == HelperClass.tempArr[p.X, p.Y])
                    {
                        b.Style = Resources["Player1Live"] as Style;
                    }
                    else b.Style = Resources["Player1End"] as Style;

                    canvas.Children.Add(b);

                    Canvas.SetLeft(b, p.X * buttonSize + 5);
                    Canvas.SetTop(b, p.Y * buttonSize + 5);
                }
                else if (p.type == 2) //button wcisniety przez gracza 2
                {
                    b = new Button() { Width = buttonSize, Height = buttonSize };
                    b.Click += b_Click;
                    b.Tag = new Coordinates(p.X, p.Y);

                    if (p.type == HelperClass.tempArr[p.X, p.Y])
                    {
                        b.Style = Resources["Player2Live"] as Style;
                    }
                    else b.Style = Resources["Player2End"] as Style;

                    canvas.Children.Add(b);

                    Canvas.SetLeft(b, p.X * buttonSize + 5);
                    Canvas.SetTop(b, p.Y * buttonSize + 5);
```

```
                }
                else if (p.type == 3) //sciana - granice planszy
                {
                    b = new Button() { Width = buttonSize, Height = buttonSize };
                    b.Click += b_Click;
                    b.Tag = new Coordinates(p.X, p.Y);
                    b.IsEnabled = false;
                    b.Style = this.Resources["ButtonWallStyle"] as Style;
                    string letters = "ABCDEFGHIJKLMN";
                    if (p.X == 0 && p.Y > 0 && p.Y < HelperClass.boardHeight + 1)
                        b.Content = p.Y;
                    else if (p.Y == 0 && p.X > 0 && p.X < HelperClass.boardHeight + 1)
                    {
                        b.Content = letters[p.X - 1];
                    }

                    b.Background = new SolidColorBrush(Colors.Blue);
                    canvas.Children.Add(b);

                    Canvas.SetLeft(b, p.X * buttonSize + 5);
                    Canvas.SetTop(b, p.Y * buttonSize + 5);
                }
            }
        }
        public void resetValues()
        {
            HelperClass.nowMove = 2;
            firstMove = true;

            whoMove();

            HelperClass.nowMove = 2; //kto teraz rusza: 1 - gracz 1, 2 - gracz 2
            HelperClass.pointsArr = new Point[HelperClass.boardWidth + 2,
HelperClass.boardHeight + 2];
            HelperClass.tempArr = new int[HelperClass.boardWidth + 2,
HelperClass.boardHeight + 2];
            startOrRestart();
            HelperClass.isFreedom = false;
            HelperClass.scorePlayer1 = 0;
            HelperClass.scorePlayer2 = 0;
            draw();
        }
        void b_Click(object sender, RoutedEventArgs e)
        {
            var button = sender as Button;
            Coordinates coord = button.Tag as Coordinates;



            if (firstMove == true) //jesli pierwszy ruch - rozpoczecie gry, to
zapamietuje pierwszy ruch w lastMove
            {
                HelperClass.lastMove = new Coordinates(coord.CX, coord.CY);
                firstMove = false;
            }


            game.playGame(coord.CX, coord.CY, HelperClass.nowMove);

            draw();

            if (HelperClass.isFreedom == true)
            {
                infoTextBlock.Text = "Freedom!";
```

```
            }
            else infoTextBlock.Text = "";

            if (game.checkGameOver() == true)
            {
                drawGameOver();
                //MessageBox.Show("Koniec Gry!");
            }

            whoMove();


        }

        private void restartButton_Click(object sender, RoutedEventArgs e)
        {
            resetValues();
            welcomeButtons.Visibility = System.Windows.Visibility.Visible;
            canvas.Visibility = Visibility.Collapsed;
            gamePanel.Visibility = Visibility.Collapsed;
        }

        private void twoPlayersButton_Click(object sender, RoutedEventArgs e)
        {
            HelperClass.gameType = HelperClass.GameType.twoPlayers;
            welcomeButtons.Visibility = System.Windows.Visibility.Collapsed;
            canvas.Visibility = Visibility.Visible;
            gamePanel.Visibility = Visibility.Visible;

        }

        private void withComputerButton_Click(object sender, RoutedEventArgs e)
        {
            HelperClass.gameType = HelperClass.GameType.withComputer;
            welcomeButtons.Visibility = System.Windows.Visibility.Collapsed;
            canvas.Visibility = Visibility.Visible;
            gamePanel.Visibility = Visibility.Visible;
        }

        private void autorsButton_Click(object sender, RoutedEventArgs e)
        {
            MessageBox.Show("Sztuczna Inteligencja\nPolitechnika Poznańska
2013\n\nRafał Nowicki\nMarcin Gmyz\nŁukasz Szabelski\n\nProgram jest implementacją gry
'Freedom' z zasadami wymyślonymi przez Veljko Cirovic i Nebojsa Sankovic w 2010
roku.","Autorzy",MessageBoxButton.OK,MessageBoxImage.Information);
        }

        private void instructionsButton_Click(object sender, RoutedEventArgs e)
        {
            MessageBox.Show("Freedom to strategiczna gra planszowa dla dwóch graczy z
rodziny 'N in a row'.\nPrzypomina nieco gry takie jak: 'Connect four' lub 'Kółko i
Krzyżyk', jednak znacznie się od nich różni.\n\nGra toczy się na planszy o wymiarach
8x8 pól. Jej celem jest posiadanie większej niż przeciwnik ilości (pionowych,
poziomych, lub ukośnych) 'czwórek' o tym samym kolorze, po zakończeniu gry.\nGra
rozpoczyna się od pustej planszy. Każdy z graczy, w każdym ruchu umieszcza jeden
pionek na planszy. Rozpoczynający kładzie pionek w dowolnym miejscu.\nW każdym
kolejnym ruchu gracze mogą położyć pionek jedynie na polach pustych - sąsiadujących z
pionkiem, który ostatnio został położony. Jeżeli dojdzie do sytuacji, w której gracz
nie może wykonać ruchu, ze względu na zajęte wszystkie sąsiadujące pola, otrzymuje on
przywilej 'Freedom' ('Wolność') - dzięki czemu może postawić pionek na dowolnym,
pustym polu.\nWygrywa ten, kto ułożył więcej 'czwórek'.", "Instrukce",
MessageBoxButton.OK, MessageBoxImage.Information);
        }
    }
```

Zawartość pliku MainWindow.xaml zawierającego opis interfejsu w języku XAML:

```
<Window x:Class="Freedom.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Freedom Board Game" Height="720" Width="800" Background="#FF8CA7C1">
    <Window.Resources>
        <Style x:Key="Player1" TargetType="Button">
            <Setter Property="BorderBrush" Value="Black"/>
            <Setter Property="Background">
                <Setter.Value>
                    <ImageBrush ImageSource="Images/blue.png" />
                </Setter.Value>
            </Setter>
        </Style>
        <Style x:Key="Player2" TargetType="Button">
            <Setter Property="BorderBrush" Value="Black"/>
            <Setter Property="Background">
                <Setter.Value>
                    <ImageBrush ImageSource="Images/green.png" />
                </Setter.Value>
            </Setter>
        </Style>
        <Style x:Key="Player1Last" TargetType="Button">
            <Setter Property="Background">
                <Setter.Value>
                    <ImageBrush ImageSource="Images/blueLast.png" />
                </Setter.Value>
            </Setter>
        </Style>
        <Style x:Key="Player2Last" TargetType="Button">
            <Setter Property="Background">
                <Setter.Value>
                    <ImageBrush ImageSource="Images/greenLast.png" />
                </Setter.Value>
            </Setter>
        </Style>
        <Style x:Key="Player1Live" TargetType="Button">
            <Setter Property="Background">
                <Setter.Value>
                    <ImageBrush ImageSource="Images/blueLive.png" />
                </Setter.Value>
            </Setter>
            <Setter Property="BorderBrush" Value="Blue" />
        </Style>
        <Style x:Key="Player2Live" TargetType="Button">
            <Setter Property="Background">
                <Setter.Value>
                    <ImageBrush ImageSource="Images/greenLive.png" />
                </Setter.Value>
            </Setter>
            <Setter Property="BorderBrush" Value="Green" />
        </Style>
        <Style x:Key="Player1End" TargetType="Button">
            <Setter Property="Background">
                <Setter.Value>
                    <ImageBrush ImageSource="Images/blueEnd.png" />
                </Setter.Value>
            </Setter>
        </Style>
        <Style x:Key="Player2End" TargetType="Button">
            <Setter Property="Background">
                <Setter.Value>
                    <ImageBrush ImageSource="Images/greenEnd.png" />
```

```
                </Setter.Value>
            </Setter>
        </Style>

        <Style x:Key="nextMoveBlue" TargetType="Image">
            <Setter Property="Source" Value="Images/blueLast.png"/>
        </Style>
        <Style x:Key="nextMoveGreen" TargetType="Image">
            <Setter Property="Source" Value="Images/greenLast.png"/>
        </Style>

        <Style x:Key="ButtonWallStyle" TargetType="Button">
            <Setter Property="OverridesDefaultStyle" Value="True"/>

            <Setter Property="Template">
                <Setter.Value>
                    <ControlTemplate TargetType="Button">
                        <Border Name="border"
                        BorderThickness="1"
                        Background="{TemplateBinding Background}">
                            <ContentPresenter HorizontalAlignment="Center"
VerticalAlignment="Center" />
                        </Border>
                        <ControlTemplate.Triggers>
                            <Trigger Property="IsEnabled" Value="false">
                                <Setter TargetName="border" Property="Background"
Value="#AAFFFFFF"/>
                            </Trigger>
                        </ControlTemplate.Triggers>
                    </ControlTemplate>
                </Setter.Value>
            </Setter>
        </Style>

    </Window.Resources>
    <Grid>
        <StackPanel Orientation="Vertical">
            <StackPanel.Background>
                <ImageBrush ImageSource="Images/bg.png"/>
            </StackPanel.Background>
            <Label Content="Freedom Board Game" FontSize="50" FontWeight="Bold"
HorizontalAlignment="Center" Height="83" VerticalAlignment="Top" FontFamily="Days"
Foreground="White"/>
            <StackPanel  Name="gamePanel" Visibility="Hidden" Background="#77FFFFFF">
                <StackPanel  HorizontalAlignment="Center" Orientation="Horizontal">
                    <TextBlock Name="whickPlayerTextBlock" HorizontalAlignment="Left"
TextAlignment="Center" Height="50" FontSize="18" FontWeight="Bold" Margin="0,0,0,0"
Padding="10,12,0,0" TextWrapping="Wrap" VerticalAlignment="Top" Width="223"/>
                    <Image x:Name="nextMoveImage" Style="{StaticResource
nextMoveBlue}" Width="40"/>
                    <TextBlock x:Name="infoTextBlock" HorizontalAlignment="Left"
Height="50" FontSize="26" FontWeight="Bold" Margin="0,0,0,0" Padding="10,6,0,0"
TextWrapping="Wrap" Text="" VerticalAlignment="Top" Width="223" Foreground="White"/>
                    <Button Name="restartButton" Content="Restart" Height="32"
HorizontalAlignment="Right" Width="100" Margin="0,0,100,0"
Click="restartButton_Click"/>
                </StackPanel>
            </StackPanel>
            <StackPanel Name="welcomeButtons" Orientation="Vertical">
                <StackPanel Orientation="Horizontal" HorizontalAlignment="Center"
VerticalAlignment="Center" Margin="0,80,0,0">
                    <Button Name="twoPlayersButton" Content="Dwóch graczy"
Background="#005be4" BorderBrush="Black" Foreground="White" FontSize="16" Width="200"
Height="200" Margin="0,0,5,0" Click="twoPlayersButton_Click"/>
```

```
                    <Button Name="withComputerButton" Content="Z komputerem"
Background="#349405" BorderBrush="Black" Foreground="White" FontSize="16"  Width="200"
Height="200" Margin="5,0,0,0" Click="withComputerButton_Click"/>
                </StackPanel>
                <Button Name="autorsButton" Content="Autorzy" Background="#cf3a00"
BorderBrush="Black" Foreground="White" FontSize="16" Width="200" Height="40"
Margin="0,10,0,0" Click="autorsButton_Click" />
                <Button Name="instructionsButton" Content="Instrukcje"
Background="#cf3a00" BorderBrush="Black" Foreground="White" FontSize="16" Width="200"
Height="40" Margin="0,10,0,0" Click="instructionsButton_Click" />
            </StackPanel>
            <Canvas Name="canvas"  Visibility="Hidden" HorizontalAlignment="Center"
VerticalAlignment="Center" Width="510" Height="510" Margin="80,20,0,0">
            </Canvas>


        </StackPanel>
    </Grid>
</Window>
```

Zawartość pliku FreedomProlog.txt zawierającego kod w języku Prolog.

```
sklej([],X,X).
sklej([X|L1],L2,[X|L3]) :-
      sklej(L1,L2,L3).

odwroc([],[]).
odwroc([H|T],L) :-
      odwroc(T,R),
      sklej(R,[H],L).

przeciwnik(1,2).
przeciwnik(2,1).

ilePkt(Tab,Gracz,Wynik):-
            punkty(Tab,Gracz,Przeciwnik,0,0,Wynik),
            przeciwnik(Gracz,Przeciwnik).

punkty([Gracz|_],Gracz,_,_,_,0).

punkty([Przeciwnik|Tail],Gracz,Przeciwnik,_,_,Wynik):-
            punkty2(Tail,Gracz,Przeciwnik,1,0,Wynik).

punkty([3|Tail],Gracz,Przeciwnik,_,_,Wynik):-
            punkty2(Tail,Gracz,Przeciwnik,1,0,Wynik).

punkty([0|Tail],Gracz,Przeciwnik,_,_,Wynik):-
            punkty2(Tail,Gracz,Przeciwnik,0,0,Wynik).

punkty2([Gracz],Gracz,_,_,_,0).

punkty2([Przeciwnik],_,Przeciwnik,Bord,Pkt,Wynik):-
            NewBord is Bord +1,
            ocenPkt(NewBord,Pkt,Wynik).
```

```
punkty2([3],_,_,Bord,Pkt,Wynik):-
            NewBord is Bord +1,
            ocenPkt(NewBord,Pkt,Wynik).

punkty2([0],_,_,Bord,Pkt,Wynik):-
            ocenPkt(Bord,Pkt,Wynik).

punkty2([Przeciwnik|Tail],_,Przeciwnik,_,_,0).

punkty2([3|Tail],_,_,_,_,0).

punkty2([0|Tail],Gracz,Przeciwnik,Bord,Pkt,Wynik):-
            punkty2(Tail,Gracz,Przeciwnik,Bord,Pkt,Wynik).

punkty2([Gracz|Tail],Gracz,Przeciwnik,Bord,Pkt,Wynik):-
            NewPkt is Pkt+1,
            punkty2(Tail,Gracz,Przeciwnik,Bord,NewPkt,Wynik).

ocenPkt(0,0,0).
ocenPkt(0,1,2).
ocenPkt(0,2,15).
ocenPkt(0,3,40).
ocenPkt(0,4,80).
ocenPkt(1,0,0).
ocenPkt(1,1,3).
ocenPkt(1,2,18).
ocenPkt(1,3,45).
ocenPkt(1,4,90).
ocenPkt(2,0,0).
ocenPkt(2,1,5).
ocenPkt(2,2,20).
ocenPkt(2,3,50).
ocenPkt(2,4,100).

%======= Przegladanie tablic heurystyczna

% poziomo ================================================

budujSzesc(Tab,Wynik):-budujSzesc2(Tab,Wynik,5).
budujSzesc2([Head|Tail],[Head|Wynik],Count):-
            NewCount is Count-1,
            budujSzesc2(Tail,Wynik,NewCount).
budujSzesc2([Head|_],[Head],0).


poziomo([Head],IleX,0,Gracz,Wynik):-
            sprawdzWiersz(Head,IleX,Gracz,Wynik).

poziomo([Head|Tail],IleX,IleY,Gracz,Wynik):-
            NewIleY is IleY-1,
            sprawdzWiersz(Head,IleX,Gracz,WynikWiersza),
            poziomo(Tail,IleX,NewIleY,Gracz,WynikTemp),
            Wynik is WynikWiersza + WynikTemp.
```

```
sprawdzWiersz([Head|Tail],IleX,Gracz,Wynik):-
            NewIleX is IleX-1,
            sprawdzWiersz(Tail,NewIleX,Gracz,WynikTemp),
            budujSzesc([Head|Tail],GotowaSzesc),
            ilePkt(GotowaSzesc,Gracz,Punkty),
            Wynik is WynikTemp + Punkty.

sprawdzWiersz(Tab,0,Gracz,Wynik):-
            budujSzesc(Tab,GotowaSzesc),
            ilePkt(GotowaSzesc,Gracz,Wynik).
% Pionowo ================================================

dajWartoscGlowy([Head],Head).

dajWartoscGlowy([Head|Tail],Head,Tail).

zamienWierKol(Temp,Wynik):-zamienWierKol2(Temp,[],[],[],Wynik).

zamienWierKol2([],[],TempHead,WynikTemp,[TempHead|WynikTemp]).

zamienWierKol2([],[Head],TempHead,WynikTemp,Wynik):-
zamienWierKol2([],[],Head,[TempHead|WynikTemp],Wynik).

zamienWierKol2([],TempTail,TempHead,WynikTemp,Wynik):-
            odwroc(TempTail,OdwroconyTempTail),

zamienWierKol2(OdwroconyTempTail,[],[],[TempHead|WynikTemp],Wynik).

zamienWierKol2([Head],TempTail,TempHead,WynikTemp,Wynik):-
            dajWartoscGlowy(Head,WartoscGlowy),

zamienWierKol2([],TailGlowy,[WartoscGlowy|TempHead],WynikTemp,Wynik).

zamienWierKol2([Head|Tail],TempTail,TempHead,WynikTemp,Wynik):-
            dajWartoscGlowy(Head,WartoscGlowy),

zamienWierKol2(Tail,TempTail,[WartoscGlowy|TempHead],WynikTemp,Wynik).

zamienWierKol2([Head],TempTail,TempHead,WynikTemp,Wynik):-
            dajWartoscGlowy(Head,WartoscGlowy,TailGlowy),

zamienWierKol2([],[TailGlowy|TempTail],[WartoscGlowy|TempHead],WynikTemp,Wy
nik).

zamienWierKol2([Head|Tail],TempTail,TempHead,WynikTemp,Wynik):-
            dajWartoscGlowy(Head,WartoscGlowy,TailGlowy),

zamienWierKol2(Tail,[TailGlowy|TempTail],[WartoscGlowy|TempHead],WynikTemp,
Wynik).

pionowo(Tablica,IleX,IleY,Gracz,Wynik):-
            zamienWierKol(Tablica,OdwroconaTablica),
            poziomo(OdwroconaTablica,IleY,IleX,Gracz,Wynik).
```

```
% Z gory na dol \   ==========================================

skosOdLewej([[X00,X01,X02,X03,X04|_],[X10,X11,X12,X13,X14,X15|_],[X20,X21,X
22,X23,X24,X25,X26|_],[X30,X31,X32,X33,X34,X35,X36,X37|_],
      [X40,X41,X42,X43,X44,X45,X46,X47,X48|_],[_,X51,X52,X53,X54,X55,X56,X5
7,X58,X59],[_,_,X62,X63,X64,X65,X66,X67,X68,X69],
      [_,_,_,X73,X74,X75,X76,X77,X78,X79],[_,_,_,_,X84,X85,X86,X87,X88,X89]
,[_,_,_,_,_,X95,X96,X97,X98,X99]],Gracz,WYNIK):-
            ilePkt([X00,X11,X22,X33,X44,X55],Gracz,W1),
            ilePkt([X01,X12,X23,X34,X45,X56],Gracz,W2),
            ilePkt([X02,X13,X24,X35,X46,X57],Gracz,W3),
            ilePkt([X03,X14,X25,X36,X47,X58],Gracz,W4),
            ilePkt([X04,X15,X26,X37,X48,X59],Gracz,W5),
            ilePkt([X10,X21,X32,X43,X54,X65],Gracz,W6),
            ilePkt([X11,X22,X33,X44,X55,X66],Gracz,W7),
            ilePkt([X12,X23,X34,X45,X56,X67],Gracz,W8),
            ilePkt([X13,X24,X35,X46,X57,X68],Gracz,W9),
            ilePkt([X14,X25,X36,X47,X58,X69],Gracz,W10),
            ilePkt([X20,X31,X42,X53,X64,X75],Gracz,W11),
            ilePkt([X21,X32,X43,X54,X65,X76],Gracz,W12),
            ilePkt([X22,X33,X44,X55,X66,X77],Gracz,W13),
            ilePkt([X23,X34,X45,X56,X67,X78],Gracz,W14),
            ilePkt([X24,X35,X46,X57,X68,X79],Gracz,W15),
            ilePkt([X30,X41,X52,X63,X74,X85],Gracz,W16),
            ilePkt([X31,X42,X53,X64,X75,X86],Gracz,W17),
            ilePkt([X32,X43,X54,X65,X76,X87],Gracz,W18),
            ilePkt([X33,X44,X55,X66,X77,X88],Gracz,W19),
            ilePkt([X34,X45,X56,X67,X78,X89],Gracz,W20),
            ilePkt([X40,X51,X62,X73,X84,X95],Gracz,W21),
            ilePkt([X41,X52,X63,X74,X85,X96],Gracz,W22),
            ilePkt([X42,X53,X64,X75,X86,X97],Gracz,W23),
            ilePkt([X43,X54,X65,X76,X87,X98],Gracz,W24),
            ilePkt([X44,X55,X66,X77,X88,X99],Gracz,W25),
            WYNIK                                                    is
W1+W2+W3+W4+W5+W6+W7+W8+W9+W10+W11+W12+W13+W14+W15+W16+W17+W18+W19+W20+W21+
W22+W23+W24+W25.

% Z dolu na gore /  ==========================================

odwrocTablice([Head|Tail],[NewHead|Wyn]):-
      odwroc(Head,NewHead),
      odwrocTablice(Tail,Wyn).


odwrocTablice([Head|_],[NewHead]):-
      odwroc(Head,NewHead).

skosOdPrawej(Tablica,Gracz,Wynik):-
      odwrocTablice(Tablica,NewTablica),
      skosOdLewej(NewTablica,Gracz,Wynik).
```

```
%======= oblicz uzytecznosc stanu
%==========Sprawdzanie wymiarow tablicy =====
sprawdzWymiarTablicy([_],1).
sprawdzWymiarTablicy([_|Tail],Wynik):-
sprawdzWymiarTablicy(Tail,TempWynik),Wynik is TempWynik+1.
%=========================================

obliczStan(Tablica,Wynik):-
            sprawdzWymiarTablicy(Tablica,Wymiar),
            IleX is Wymiar-6,
            IleY is Wymiar-1,
            poziomo(Tablica,IleX,IleY,1,PktKompPoziomo),
            pionowo(Tablica,IleY,IleX,1,PktKompPionowo),
            skosOdLewej(Tablica,1,PktKompLewy),
            skosOdPrawej(Tablica,1,PktKompPrawy),
            poziomo(Tablica,IleX,IleY,2,PktGraczPoziomo),
            pionowo(Tablica,IleY,IleX,2,PktGraczPionowo),
            skosOdLewej(Tablica,2,PktGraczLewy),
            skosOdPrawej(Tablica,2,PktGraczPrawy),
            WynikKomp is PktKompPoziomo + PktKompPionowo + PktKompLewy +
            PktKompPrawy ,
            WynikGracz is PktGraczPoziomo + PktGraczPionowo + PktGraczLewy
            + PktGraczPrawy,
            Wynik is WynikKomp-WynikGracz.

%=========================================================================
====%====== mini Max ==============

czyRownyElWiersza([TypElem|_],0,TypElem).
czyRownyElWiersza([_|Tail],WspX,TypElem):-NewWspX is WspX -
1,czyRownyElWiersza(Tail,NewWspX,TypElem).

czyRownyElTab([Head|_],WspX,0,TypElem):-
czyRownyElWiersza(Head,WspX,TypElem).
czyRownyElTab([_|Tail],WspX,WspY,TypElem):-NewWspY is WspY-1,
czyRownyElTab(Tail,WspX,NewWspY,TypElem).

zamienElWiersza([_|Tail],0,TypElem,[TypElem|Tail]).
zamienElWiersza([Head|Tail],WspX,TypElem,[Head|NewTail]):-NewWspX is WspX -
1,zamienElWiersza(Tail,NewWspX,TypElem,NewTail).

zamienElTab([Head|Tail],WspX,0,TypElem,[NewHead|Tail]):-
zamienElWiersza(Head,WspX,TypElem,NewHead).
zamienElTab([Head|Tail],WspX,WspY,TypElem,[Head|NewTail]):-NewWspY is WspY-
1, zamienElTab(Tail,WspX,NewWspY,TypElem,NewTail).

czyNiePusty(Tablica,WspX,WspY):-czyRownyElTab(Tablica,WspX,WspY,1).
czyNiePusty(Tablica,WspX,WspY):-czyRownyElTab(Tablica,WspX,WspY,2).
czyNiePusty(Tablica,WspX,WspY):-czyRownyElTab(Tablica,WspX,WspY,3).

czyPelna(1).
czyPelna(2).
czyPelna(3).
czyPelna(1|_).
czyPelna(2|_).
```

```
czyPelna(3|_).
czyPelna([Head]):-czyPelna(Head).
czyPelna([Head|Tail]):-czyPelna(Tail),czyPelna(Head).

czyFreedom(Tablica,OstatniX,OstatniY):-
            Yminus is OstatniY-1,
            Yplus is OstatniY +1,
            Xminus is OstatniX -1,
            Xplus is OstatniX +1,
            czyNiePusty(Tablica,Xminus,Yminus),
            czyNiePusty(Tablica,Xminus,OstatniY),
            czyNiePusty(Tablica,Xminus,Yplus),
            czyNiePusty(Tablica,OstatniX,Yminus),
            czyNiePusty(Tablica,OstatniX,Yplus),
            czyNiePusty(Tablica,Xplus,Yminus),
            czyNiePusty(Tablica,Xplus,OstatniY),
            czyNiePusty(Tablica,Xplus,Yplus).

miniMax(Tablica,_,_,_,0,Wynik):-obliczStan(Tablica,Wynik).
miniMax(Tablica,_,_,_,_,Wynik):-
czyPelna(Tablica),obliczStan(Tablica,Wynik).

miniMax(Tablica,Xnext,Ynext,1,_,-10):-
            not(czyRownyElTab(Tablica,Xnext,Ynext,0)).

miniMax(Tablica,Xnext,Ynext,2,_,10):-
            not(czyRownyElTab(Tablica,Xnext,Ynext,0)).

miniMax(Tablica,Xnext,Ynext,2,Poziom,Wynik):-
            NewPoziom is Poziom -1,
            czyFreedom(Tablica,Xnext,Ynext),
            czyRownyElTab(Tablica,Xnext,Ynext,0),
            zamienElTab(Tablica,Xnext,Ynext,2,NewTab),
            sprawdzWymiarTablicy(Tablica,RozmiarTablicy),

przegladajTablice(NewTab,0,0,RozmiarTablicy,RozmiarTablicy,RozmiarTablicy,1
,NewPoziom,Wynik).

miniMax(Tablica,Xnext,Ynext,2,Poziom,Wynik):-
            XM is Xnext-1,
            XP is Xnext+1,
            YM is Ynext-1,
            YP is Ynext+1,
            NewPoziom is Poziom -1,
            czyRownyElTab(Tablica,Xnext,Ynext,0),
            zamienElTab(Tablica,Xnext,Ynext,2,NewTab),
            not(czyFreedom(Tablica,Xnext,Ynext)),
            przegladajTablice(NewTab,XM,YM,XP,YP,XP,1,NewPoziom,Wynik).

miniMax(Tablica,Xnext,Ynext,1,Poziom,Wynik):-
            NewPoziom is Poziom -1,
            czyFreedom(Tablica,Xnext,Ynext),
            czyRownyElTab(Tablica,Xnext,Ynext,0),
            zamienElTab(Tablica,Xnext,Ynext,1,NewTab),
            sprawdzWymiarTablicy(Tablica,RozmiarTablicy),
```

```
przegladajTablice(NewTab,0,0,RozmiarTablicy,RozmiarTablicy,RozmiarTablicy,2
,NewPoziom,Wynik).

miniMax(Tablica,Xnext,Ynext,1,Poziom,Wynik):-
            XM is Xnext-1,
            XP is Xnext+1,
            YM is Ynext-1,
            YP is Ynext+1,
            NewPoziom is Poziom -1,
            czyRownyElTab(Tablica,Xnext,Ynext,0),
            zamienElTab(Tablica,Xnext,Ynext,1,NewTab),
            not(czyFreedom(Tablica,Xnext,Ynext)),
            przegladajTablice(NewTab,XM,YM,XP,YP,XP,2,NewPoziom,Wynik).

przegladajTablice(Tablica,Xdo,Ydo,Xdo,Ydo,_,1,Poziom,Wynik):-
            miniMax(Tablica,Xdo,Ydo,1,Poziom,Wynik).

przegladajTablice(Tablica,Xdo,Ydo,Xdo,Ydo,_,2,Poziom,Wynik):-
            miniMax(Tablica,Xdo,Ydo,2,Poziom,Wynik).

przegladajTablice(Tablica,Xdo,Yod,Xdo,Ydo,Xpomocniczy,2,Poziom,Wynik):-
            NewYod is Yod +1,
            miniMax(Tablica,Xdo,Yod,2,Poziom,WynikMinMax),

przegladajTablice(Tablica,Xpomocniczy,NewYod,Xdo,Ydo,Xpomocniczy,2,Poziom,W
ynikSprawdz),
            Wynik is min(WynikMinMax,WynikSprawdz).

przegladajTablice(Tablica,Xdo,Yod,Xdo,Ydo,Xpomocniczy,1,Poziom,Wynik):-
            NewYod is Yod +1,
            miniMax(Tablica,Xdo,Yod,1,Poziom,WynikMinMax),

przegladajTablice(Tablica,Xpomocniczy,NewYod,Xdo,Ydo,Xpomocniczy,1,Poziom,W
ynikSprawdz),
            Wynik is max(WynikMinMax,WynikSprawdz).


przegladajTablice(Tablica,Xod,Yod,Xdo,Ydo,Xpomocniczy,Gracz,Poziom,Wynik):-
            NewXod is Xod +1,
            miniMax(Tablica,Xod,Yod,Gracz,Poziom,WynikMinMax),

przegladajTablice(Tablica,NewXod,Yod,Xdo,Ydo,Xpomocniczy,Gracz,Poziom,Wynik
Sprawdz),
            Wynik is max(WynikMinMax,WynikSprawdz).

%dlaJakich(_,Xdo,Ydo,Xdo,Ydo,_,_,_,Xdo,Ydo).

dlaJakich(Tablica,Xod,Yod,_,_,_,Poziom,MaxWartosc,Xod,Yod):-
            miniMax(Tablica,Xod,Yod,1,Poziom,MaxWartosc).

dlaJakich(Tablica,Xdo,Yod,Xdo,Ydo,Xpomocniczy,Poziom,MaxWartosc,Xwybr,Ywybr
):-
            NewYod is Yod +1,
            miniMax(Tablica,Xdo,Yod,1,Poziom,WynikMinMax),
            MaxWartosc\=WynikMinMax,
```

```
dlaJakich(Tablica,Xpomocniczy,NewYod,Xdo,Ydo,Xpomocniczy,Poziom,MaxWartosc,
Xwybr,Ywybr).

dlaJakich(Tablica,Xod,Yod,Xdo,Ydo,Xpomocniczy,Poziom,MaxWartosc,Xwybr,Ywybr
):-
            NewXod is Xod +1,
            miniMax(Tablica,Xod,Yod,1,Poziom,WynikMinMax),
            MaxWartosc\=WynikMinMax,

dlaJakich(Tablica,NewXod,Yod,Xdo,Ydo,Xpomocniczy,Poziom,MaxWartosc,Xwybr,Yw
ybr).

wybierzNastepny(Tablica,Xostatni,Yostatni,Poziom,WynikX,WynikY):-
            XM is Xostatni-1,
            XP is Xostatni+1,
            YM is Yostatni-1,
            YP is Yostatni+1,
            przegladajTablice(Tablica,XM,YM,XP,YP,XP,1,Poziom,WynikMax),

dlaJakich(Tablica,XM,YM,XP,YP,XP,Poziom,WynikMax,WynikX,WynikY).


wybierzNastepnyX(Tablica,Xostatni,Yostatni,Poziom,WynikX):-
            wybierzNastepny(Tablica,Xostatni,Yostatni,Poziom,WynikX,_).

wybierzNastepnyY(Tablica,Xostatni,Yostatni,Poziom,WynikY):-
            wybierzNastepny(Tablica,Xostatni,Yostatni,Poziom,_,WynikY).
```