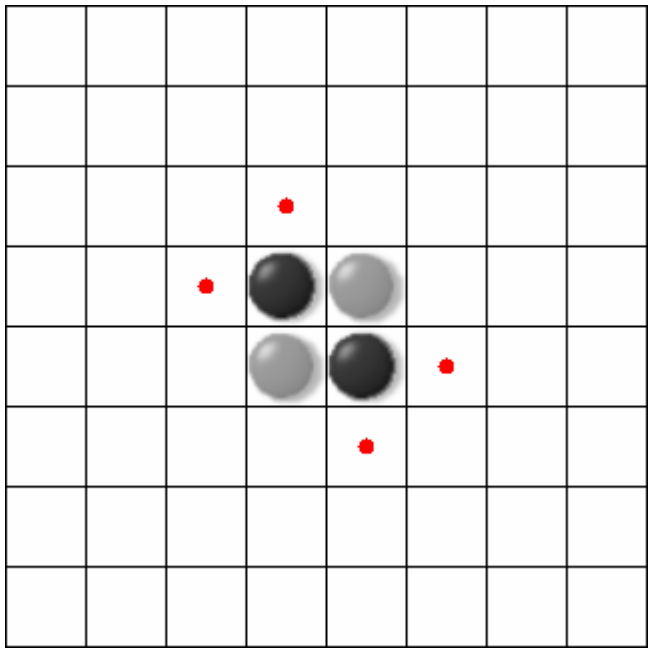
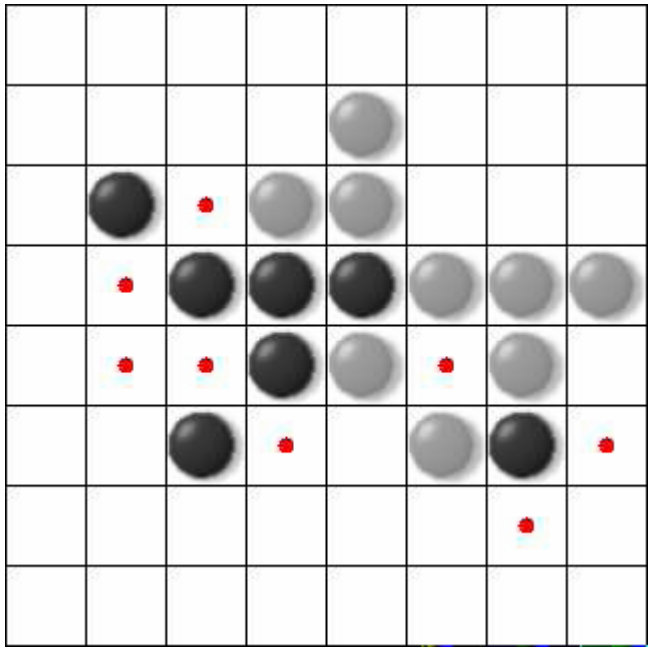
Othello



Autor: Piotr Truszkowski

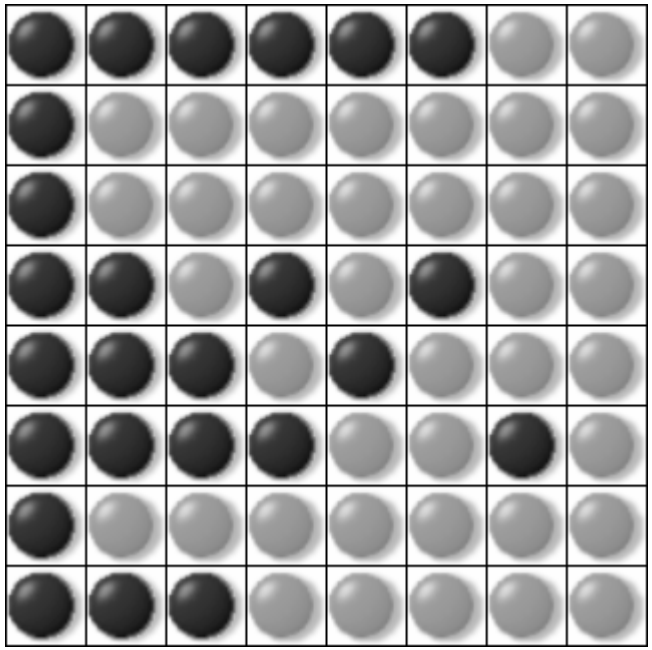
Cel projektu

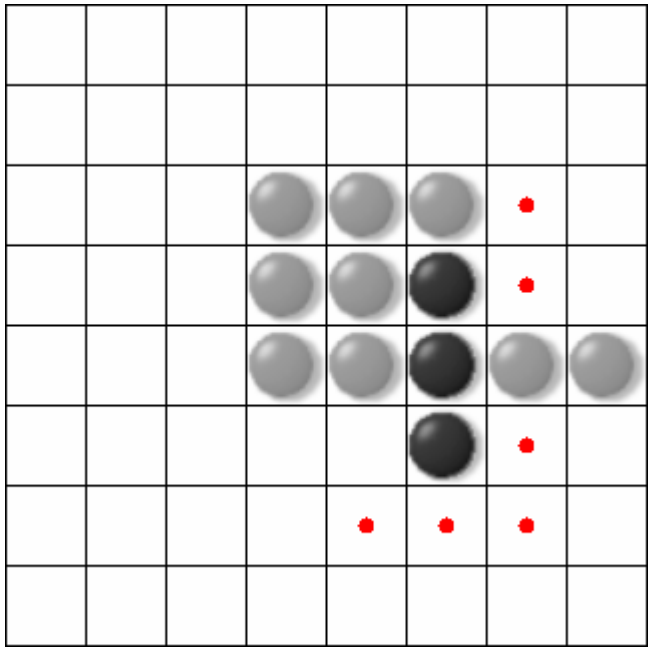
Zaimplementowanie gry Othello(reversi).



Zasady

Gracze wykonują ruchy naprzemiennie, gracz może postawić swój kamień tylko w takim miejscu gdzie będzie mógł odwrócić przynajmniej jeden kamień przeciwnika. Wygrywa ten, który będzie miał więcej kamieni(bądź gdy przeciwnik w trakcie gry straci wszystkie kamienie).





GLEM

Kolejnym celem projektu było zaimplementowanie modułu inteligencji opierającego się na pracach Michaela Buro: ***From Simple Features to Sophisticated Evaluation Functions***. A w istocie na ***Generalized Linear Evaluation Model*** (**GLEM**).

- **Konfiguracja** – to pewna cecha planszy, która zachodzi bądź nie, np: w szachach pytanie czy biały wykonał roszadę, a w othello – czy biały zajął cały róg 3x3.

Funkcja v niech określa czy zadana konfiguracja c jest prawdziwa w p

$$v(c(p)) = \begin{cases} 1, \text{ jeśli konfiguracja } c \text{ jest prawdziwa w } p \\ 0, \text{ wpp} \end{cases}$$

Jak Oceniać plansze

Jeśli każdej konfiguracji przypiszemy pewne wagi to ocena sytuacji sprowadzi się do:

$$e(p) = g\left(\sum w_i v\left(c_i(p)\right)\right)$$

Funkcja $g(x)$ przeprowadza dziedzinę $\Re$ w przedział $[0,1]$. Gdzie 0 można interpretować jako porażkę, 1 jako wygraną.

$$g(x) = \frac{1}{1 + e^{-x}}$$

Jak dobrać wagi

Mając zbiór pozycji na planszy i dla każdej pozycji przyporządkowaną ocenę sytuacji, można ulepszać istniejące wagi na podstawie wzorów:

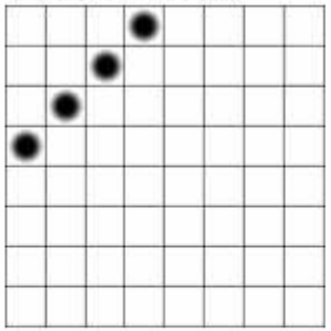
$$\mathcal{E}(w) = \frac{1}{\mathcal{N}} \sum (r_k - e_w(P_k))^2$$

Gdzie $\mathcal{E}(w)$ określa błąd między wzorcową wagą r dla konfiguracji a dotychczasową. Jeśli w potraktujemy jako wektor wag to iteracyjnie można otrzymać coraz lepsze wagi:

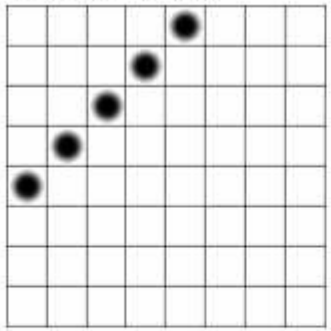
$$w^{(n+1)} = w^{(n)} + 2\alpha \nabla \mathcal{E}(w^{(n)})$$

Dobieranie konfiguracji – patterny

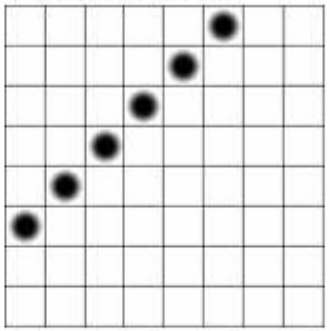
ukośnie 4 pola



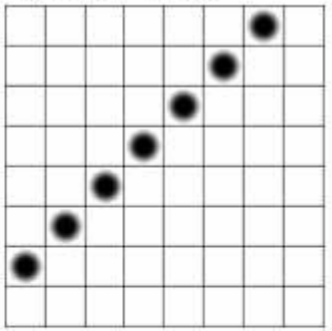
ukośnie 5 pól



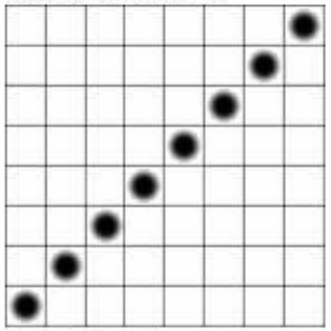
ukośnie 6 pól



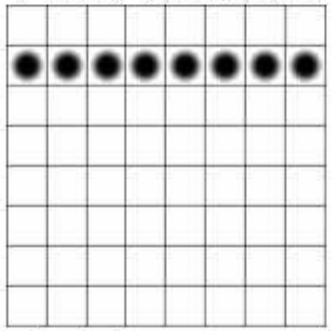
ukośnie 7 pól



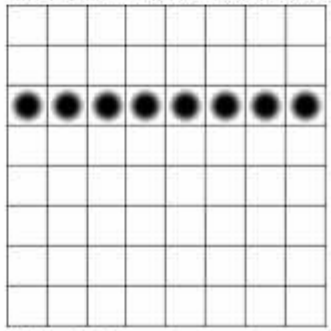
ukośnie 8 pól



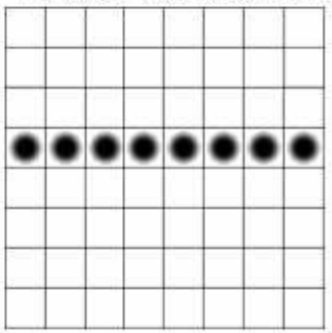
2 linia (pion/poziom)



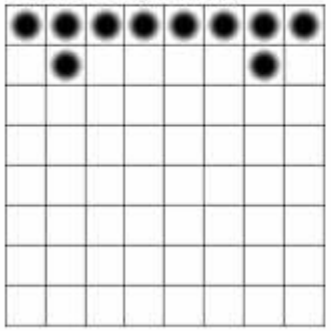
3 linia (pion/poziom)



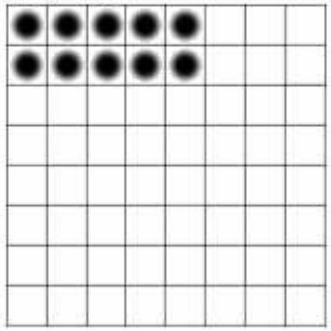
4 linia (pion/poziom)



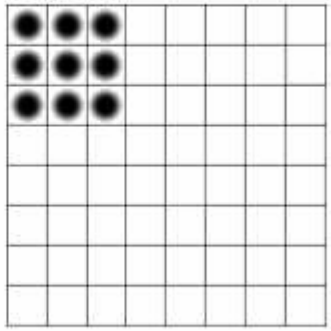
kraniec planszy



róg 2x5



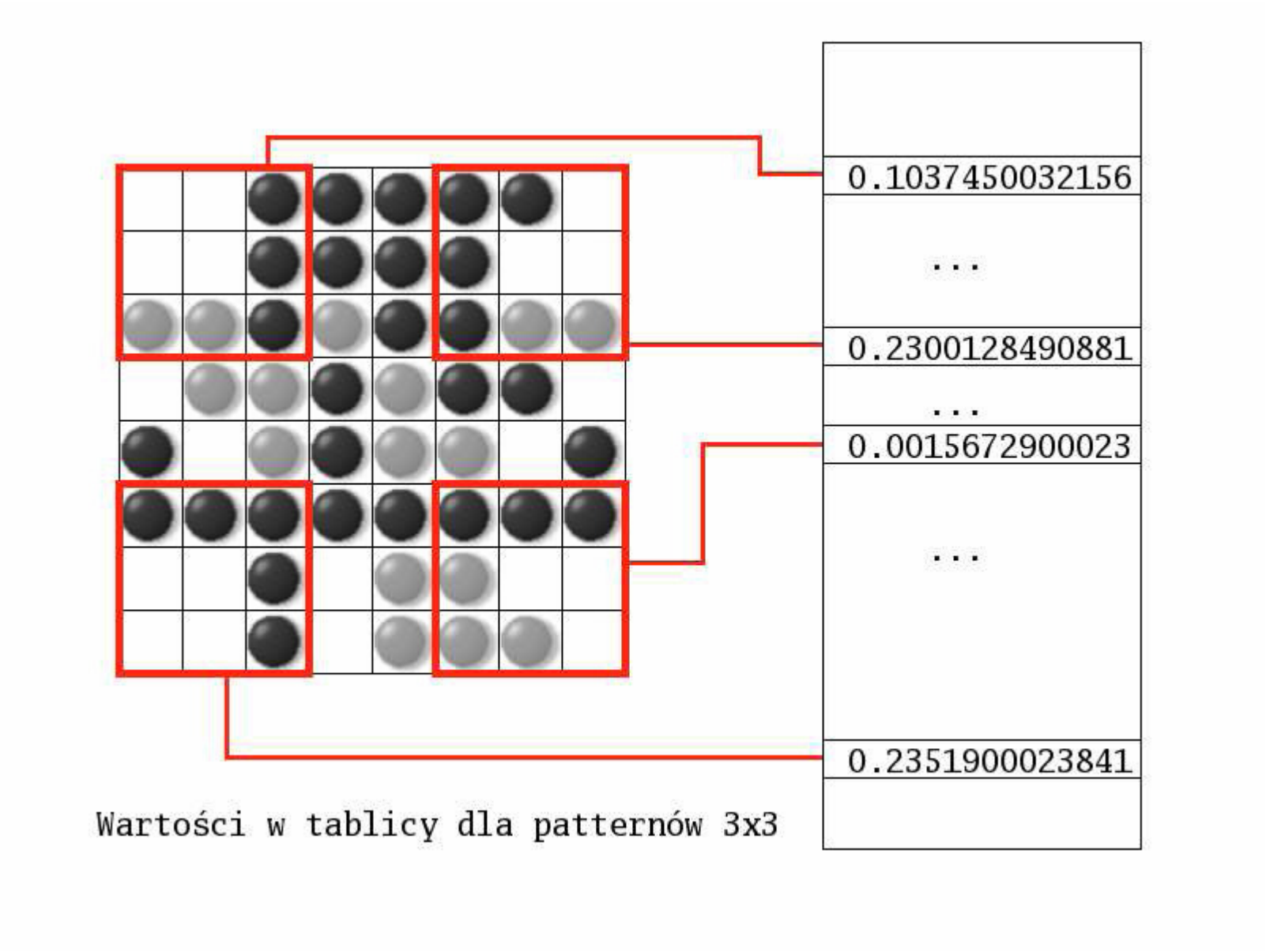
róg 3x3



Realizacja Projektu – podział na moduły

```
module OthelloBoard : BGAME_BOARD =
  struct ... end;;

module OthelloAI (BGAME_BOARD) : BGAME_AI =
  struct ... end;;

module OthelloGUI (BGAME_BOARD) : BGAME_GUI =
  struct ... end;;

module BoardGame : BGAME =
  functor (BGAME_BOARD) ->
  functor (BGAME_AI with ...) ->
  functor (BGAME_GUI with ...) ->
    struct ... end;;

module Othello = BoardGame
                   (OthelloBoard)
                   (OthelloAI (OthelloBoard))
                   (OthelloGUI (OthelloBoard));;
```

Ocenianie – Zastosowanie tablicy haszującej



Czyli...

Ocena sytuacji sprowadza się do wyliczenia:

```
let evaluate hashtbl board =
  List.fold_right
    (fun f v -> (Hash.find hashtbl (f board)) +. v)
    list_of_patterns 0
```

Gdzie `hashtbl` to tablica z wagami wszystkich konfiguracji, `list_of_patterns` to lista funkcji zwracających dla zadanej pozycji konfigurację odpowiadającą patternowi. A `Hash` to moduł tablicy haszującej z przerobioną funkcją haszującą.

Nauka programu

Zaprojektowane zostały dwa sposoby nauki.

- Na przyrządzonych zestawach lekcji – dla każdej pozycji ocena wzorcowa.
- Na grze, program jako wartość oceny wzorcowej przyjmuje wynik działania algorytmu alfabeta-obcięcia.

```
let alphabeta deep board alpha who =
  let rec ab deep board w =

...
  in let result = (ab deep board who)
  in let new_board = snd result
  in better new_board who
    (ai_evaluate new_board who)                    (* stara ocena    *)
    (fst result)                                   (* ocena alfabety *)
; new_board
```

Funkcja `better` dla planszy i gracza weryfikuje wartości wag.

Efekt końcowy: