# Streszczenie

Blab bla bla bla. Lorem ipsum hahahaahaha, afsafas,fsa fsa fasfasfsafa, safasfsa a to wielce zastraszeni a la ma kota lorem ipsum lakakaka now wies nooo aaajaalajfahfksafsfas, dsafsdfsdfsdgsd. Moja momama fuafajfasfhsja frfrevre, fasgfhsa jadsk, afsafas. Rikoro.

Aha ma jora lajfsaklf sakjfas fsakjfasdf, dshghasm, hgksdhg fasfhka fskajhfsjaf asfjshajkfsdf. Ajfklas fajskfhsjak hjkfhjksd jdsbvks jvhjsdk, hjkghsdkjag, hgjhsdkjgsd.

# Abstract

Fsfasdfdsfsdfsdafdsdsadsa, fsdfasdfsad, fsdfasdfsdafasdjkdsa fjdskhfkjdsaf fdshfjkds hkjdshfkjdsfdksjfhjkad jkvdsjkvhsdkjv vdsjhvdksjvhjksdav vdjkshvkjdshvsdkj.

# Spis treści

[Streszczenie 3](#_Toc452764783)

[Abstract 4](#_Toc452764784)

[Spis treści 5](#_Toc452764785)

[Wykaz ważniejszych oznaczeń i skrótów 6](#_Toc452764786)

[1. Wstęp i cel pracy 7](#_Toc452764787)

[2. Przegląd i analiza dostępnych implementacji silników popularnych gier logicznych 8](#_Toc452764788)

[3. Implementacja generatora silników wybranej gry 9](#_Toc452764789)

[3.1 Problem wyboru gry 9](#_Toc452764790)

[3.2 Wybrana gra – szachy 9](#_Toc452764791)

[3.3 Architektura projektu 9](#_Toc452764792)

[4. Analiza wydajnościowo-poprawnościowa zaimplementowanego generatora silników gier 10](#_Toc452764793)

[5. Podsumowanie 11](#_Toc452764794)

[Wykaz literatury 12](#_Toc452764795)

[Wykaz rysunków 13](#_Toc452764796)

[Wykaz tabel 14](#_Toc452764797)

[Dodatek A: Tytuł dodatku A 15](#_Toc452764798)

# Wykaz ważniejszych oznaczeń i skrótów

ICGA – International Computer Games Association

TCEC – Top Chess Engine Championship

WPF – Windows Presentation Foundation

GUI – graficzny interfejs użytkownika

UI – interfejs użytkownika

.NET – Microsoft .NET Framework

n – liczba wierszy planszy

m – liczba kolumn planszy

k – k-pod-rząd postawionych pionów jest potrzebnych żeby wygrać

p – liczba ruchów (np. stawianych pionów) w turze gracza

q – liczba ruchów (np. stawianych pionów) przez pierwszego gracza w pierwszej turze

# Wstęp i cel pracy

W początkowych dziejach algorytmiki i komputerów znacznie bardziej istotną cechą algorytmów była złożoność pamięciowa [1]. Wynikało to z faktu, że dostępne komputery miały mało pamięci operacyjnej a sama w sobie była ona niezwykle droga. Duża jej ilość jest natomiast niezbędna dla niektórych silników gier logicznych, jako, że muszą one np. generować możliwe posunięcia lub trzymać „drzewo gry” w pamięci. Dokładając do tego niewielką wydajność pierwszych komputerów, pierwsze programy dobrze grające w gry logiczne jak warcaby czy szachy nie pojawiły się aż do późnych lat siedemdziesiątych [2][3][4].

Dzisiaj dzięki dużej, taniej i łatwo dostępnej pamięci operacyjnej, wysokiej wydajności procesorów i rozwojowi algorytmiki mamy do czynienia z prawdziwym rozkwitem dziedziny zajmującej się programami grającymi. Rozwój jest tym bardziej podsycany poprzez nieustanną rywalizację pomiędzy programami grającymi podczas takich wydarzeń jak olimpiada ICGA [5][6] czy TCEC [7]. Na samej olimpiadzie ICGA rozgrywane są corocznie mistrzostwa programów grających w około 36 różnych gier logicznych [8] a autorzy zapewniają możliwość dodania kolejnych gier jeżeli tylko znajdą się chętni [9].

Przy całej tej rywalizacji i rozwoju oczywiste jest, że kolejne generacje silników gier będą coraz bardziej wydajne i coraz lepiej wykorzystujące zasoby sprzętowe. Jedną z możliwości dalszego zwiększania wydajności jest stosowanie optymalizacji sprzętowych – wykorzystywanie konkretnych funkcji architektury procesora, o ile są dostępne. Często wymaga to umieszczania wstawek assemblerowych w kodzie źródłowym programu grającego.

Trzeba jednak przyznać nam, jako gatunkowi ludźkiemu niemały sukces na tym polu, dzięki wspomnianemu szybkiemu rozwojowi wydajności procesorów, pojemności pamięci i efektywności algorytmów udało się w 2007 roku rozwiązać wariant angielski Warcabów [10]. Rozwiązanie jest określane jako *słabe*, tj algorytm zaimplementowany w programie Chinook nie mna gwarancji wygranej z każdym przeciwnikie, ale jest zagwarantowane, że nigdy nie przegra. Tak czy owak, robi to ogromne wrażenie i jest wspólnym osiągnięciem algorytmiki i inżynierii komputerowej – wyznacznikiem naszego postępu jako ludzkość. Algorytmika, która korzysta z dobrodziejstw inżynierii komputerowej, możnaby rzecz „algorytmy zoptymalizowane pod sprzęt” to właśnie bardzo istotna część tej pracy dyplomowej jak i nowoczesnych programów grających.

W niniejszej pracy z racji złożoności tematyki i trudności implementacji wydajnych silników gier logicznych poświecono cały pierwszy rozdział na przegląd i analizę dostępnych implementacji silników popularnych gier logicznych. Przegląd ten przede wszystkim ma za zadanie nakreślić ogólny trend w rozwoju silników gier i co ważniejsze umożliwić autorowi implementację generatora silników dla wybranej gry logicznej w rozdziale drugim.

Głównym celem pracy jest implementacja i analiza generatora silników dla wybranej gry logicznej. Program powinien umożliwić obsługującemu go człowiekowi wygodne wygenerowanie konkretnego silnika wybranej gry logicznej, parametryzowanego takimi wartościami jak: rozmiary planszy, liczba pól aktywnych planszy, zasady gry, liczba graczy. W implementacji głównym wyzwaniem jest napisać taki generator, aby generowane silniki były przynajmniej równie dobre (równie szybkie, z równie mocnym AI) co znane silniki dla wybranej gry. Jest to zadanie zdecydowanie trudne, dlatego też cały trzeci rozdział jest przeznaczony na sprawdzenie poprawności i wydajności zaimplementowanego generatora.

Podsumowując jak już wspomniano praca ma zasadniczo trzy części, w pewnym sensie są więc trzy cele do zrealizowania w pracy. Część pierwsza jest czysto opisowa, skupia się na przeglądzie i analizie dostępnych silników gier logicznych, jej celem jest poznanie trendów implementacyjnych współczesnych programów grających. Część druga jest czysto implementacyjna, jej celem jest powstanie generatora wydajnych i poprawnych silników wybranej gry logicznej. Część trzecia ma najbardziej charakter analizy i testów, jej celem jest zbadanie wydajności zaimplementowanego generatora i jego dogłębne przetestowanie (głównie poprzez testowanie silników przez niego generowanych.

# Przegląd i analiza dostępnych implementacji silników popularnych gier logicznych

W niniejszym rozdziale postanowiono bliżej przyjrzeć się wybranym, dostępnym implementacjom silników popularnych gier logicznych. Ze względu na ograniczoną objętość pracy, należało dokonać mądrych wyborów, tj. wybrać nowoczesne i aktywnie rozwijane silniki (aby uwzględnić trend ich rozwoju), ale też przeznaczone do popularnych gier (większa rywalizacja, więc wyższa jakość implementacji). Autor jako miłośnik czystego kodu, w procesie wyboru silników do analizy, dużą wagę przywiązywał również do tego czy udostępniony kod źródłowy programu grającego jest dobrze ustrukturalizowany i przejrzyście napisany.

Wstępnie zadecydowano również, że najlepsze do analizy będą programy najlepsze w swojej dziedzinie, interesujące są przecież rozwiązania bliskie optymalnym – można zadać pytanie „co takiego znajduje się w kodzie programu grającego, że dało mu przewagę nad innymi”. W tym celu przeanalizowano wyniki z olimpiad ICGA i TCEC.

**Tabela 2.1.** Wyniki turnieju "Top Chess Engine Championship" [5]

|  |  |  |  |
| --- | --- | --- | --- |
| Sezon | Data | Zwycięzca | Drugie miejsce |
| 1 | Dec 2010 – Feb 2011 | Houdini 1.5a | Rybka 4.0 |
| 2 | Feb 2011 – Apr 2011 | Houdini 1.5a | Rybka 4.1 |
| 3 | Apr 2011 – May 2011 | *brak (sezon niedokończony)* | |
| 4 | Jan 2013 – May 2013 | Houdini 3 | Stockfish 250413 |
| 5 | Aug 2013 – Dec 2013 | Komodo 1142 | Stockfish 191113 |
| 6 | Feb 2014 – May 2014 | Stockfish 170514 | Komodo 7x |
| 6 (szachy losowe) | June 2014 – July 2014 | Stockfish 260614 | Houdini 4 |
| 7 | Sep 2014 – Dec 2014 | Komodo 1333 | Stockfish 141214 |
| 8 | Aug 2015 – Nov 2015 | Komodo 9.3x | Stockfish 021115 |
| 9 | May 2016 – Dec 2016 |  |  |

W tabeli 2.1 zaprezentowano wyniki z wszystkich sezonów turnieju szachowego TCEC. Na ich podstawie można stwierdzić, że prawdopodobnie dwa najlepsze programy grające w szachy na dzień dzisiejszy to Stockfish oraz Komodo. Przydałoby się przeanalizować w niniejszym rozdziale oba. Niestety Komodo, który jest prawdopodobnie najlepszym silnikiem szachowym na świecie [25] [26], jest także komercyjny i jego kod źródłowy jest zamknięty. Można jedynie kupić obecną wersję wykonywalną [27] lub poprzednią w wyprzedaży [28], dwie wersje wstecz są natomiast dostępne na licencji freeware [29]. Tak czy owak jako, że kod źródłowy nie jest dostępny analiza nie może zostać przeprowadzona. Szczęśliwie jednak, miejsce drugie z TCEC z 2015 roku, czyli Stockfish jest otwartoźródłowy i dostępny na publicznym repozytorium w serwisie github [30] [31]. W dalszej części tego rozdziału będzie jednym z kluczowych silników poddanych analizie.

**Tabela 2.2.** Wyniki w grze Go 9x9 [11]

|  |  |  |
| --- | --- | --- |
| Miejsce | Nazwa programu | Punkty |
| **1** | **Zen** | 9.5 |
| **2** | **Abakus** | 7.5 |
| **3** | **CGI** | 7.0 |
| 4 | Nomitan | 4.0 |
| 5 | MC\_ark | 2.0 |
| 6 | Wingo | 0.0 |

Program Zen w 2015 wygrał we wszystkich rozgrywanych na Olimpiadzie Gier Komputerowych ICGA wersjach gry Go [11], nie tylko w wersji 9x9, której wyniki przedstawiono w tabeli 2.2 ale też w wersjach 13x13 i 19x19. Byłby więc doskonałym kandydatem na analizę, niestety program został wydany komercyjnie [12]. Autor – japoński programista Yoji Ojima po raz pierwszy w opublikował go jako komercyjny program w roku 2009 i kolejne jego wersje w latach następnych. Najnowsza wersja 6 pochodzi z 2016 i można ją zakupić na stronie [13], nie ma więc mowy o jakiejkolwiek analizie kodu źródłowego.

**Tabela 2.3.** Wyniki w grze Hex 11x11 [11]

|  |  |  |
| --- | --- | --- |
| Miejsce | Nazwa | Punkty |
| **1** | Mo-Hex | 7 |
| **2** | Deep-Hex | 5 |
| **3** | Ezo | 0 |

**Tabela 2.4.** Wyniki w grze Hex 13x13 [11]

|  |  |  |
| --- | --- | --- |
| Miejsce | Nazwa | Punkty |
| **1** | Mo-Hex | 6 |
| **2** | Deep-Hex | 6 |
| **3** | Ezo | 0 |

Innym dominującym silnikiem na wspomnianej wcześniej olimpiadzie okazał się być program MoHex, który wygrał w obu rozgrywanych odmianach gry Hex (11x11 i 13x13) [11] jak pokazano w tabelach 2.3 i 2.4. W tym przypadku okazało się, że przynajmniej jakaś (zapewne starsza) wersja kodu źródłowego jest dostępna [14][15]. Dodatkowym atutem jest fakt, że w mistrzowskim programie Mo-Hex został wykorzystany algorytm polaka, Jakuba Pawlewicza – „new virtual connection implementation”. Po raz pierwszy ten algorytm został zaprezentowany w programie polaka MIMHex, którego kod źródłowy jest dostępny w kilku wersjach, rozwijanych przez różnych autorów [16] [17] [18] [19] [20] [21] [22] [23] [24], co jeszcze bardziej ułatwi analizę.

Ostatnią już grą dla jakiej postanowiono dokonać przeglądu i analizy zaimplementowanych i dostępnych silników jest Connect6. Przyczyną jest pewna fascynacja autora zagadnieniem gier (n,m,k,p,q), gdzie Connect6 jest grą (∞,∞,6,2,1) [34] [35]. Istnieją też inne, niezwiązane bezpośrednio z jakością dostępnych silników tej gry, przyczyny, które zostaną ujawnione w dalszej części pracy.

**Tabela 2.5.** Wyniki w grze Connect6 [11]

|  |  |  |
| --- | --- | --- |
| Miejsce | Nazwa | Punkty |
| 2 | Floating Cloud | 6 |
| 1 | **Explorer** | 6 |
| **3** | **USTB1** | 0 |

Niestety pomimo dogłębnego przeszukania internetu, żaden z programów grających w Connect6 i biorących udział w Olimpiadzie Gier Komputerowych ICGA w 2015 roku nie został odnaleziony w internecie. Ani w formie plików wykonywalnych ani tym bardziej w formie kodu źródłowego. Po dosyć dogłębnych poszukiwaniach programów grających w Connect6 udało się znaleźć jeden dobrze napisany, nazwany „Connect-k”. Jego autor twierdzi, że jedynym innym dostępnym w internecie programem grający w Connect6 jest NCTU6 autorstwa dr-a I-Chen Wu (pomysłodawcy Connect6) [37]. NCTU6 zazwyczaj wygrywa z Connect-k ale jest za to o zamkniętym kodzie i w języku Chińskim [37] [38], odpada więc uwzględnienie go w analizie. Autor tej pracy znalazł też inny silnik Connect6, zaimplementowany przez bliżej nieznaną osobę [39], może posłużyć on do ewentualnego porównania z Connect-k, który zostanie w dalszej części rozdziału dogłębnie zanalizowany.

## Stockfish

Ramadan kokokdad, dashjaf afjkashfjka i fala moko. Ragodago miko miko regajlsafjas, aksgsa jkla. Jakaakaka moranranin polo mdiasdoas.

Ja kaokkka hjfksahfja fsjdkhfas fsdkjfhsdjk vdsjvhds, hgjkdsgds, hgjdkshgjskdhgjksdhgjkasd i jakaa ahkgjsdhjkghasdjkgsadgds. Moja fhjadkfhsjdkfhui fhdsiufhdsui i jok. Hjhfahfsa hjafsaj i jaaa, hjaksfhkjsa jvskld.

## Mo-Hex

Ramadan kokokdad, dashjaf afjkashfjka i fala moko. Ragodago miko miko regajlsafjas, aksgsa jkla. Jakaakaka moranranin polo mdiasdoas.

Ja kaokkka hjfksahfja fsjdkhfas fsdkjfhsdjk vdsjvhds, hgjkdsgds, hgjdkshgjskdhgjksdhgjkasd i jakaa ahkgjsdhjkghasdjkgsadgds. Moja fhjadkfhsjdkfhui fhdsiufhdsui i jok. Hjhfahfsa hjafsaj i jaaa, hjaksfhkjsa jvskld.

## MIMHex

Ramadan kokokdad, dashjaf afjkashfjka i fala moko. Ragodago miko miko regajlsafjas, aksgsa jkla. Jakaakaka moranranin polo mdiasdoas.

Ja kaokkka hjfksahfja fsjdkhfas fsdkjfhsdjk vdsjvhds, hgjkdsgds, hgjdkshgjskdhgjksdhgjkasd i jakaa ahkgjsdhjkghasdjkgsadgds. Moja fhjadkfhsjdkfhui fhdsiufhdsui i jok. Hjhfahfsa hjafsaj i jaaa, hjaksfhkjsa jvskld.

## Connect-k

Program Connect-k powstał w pierwotnej wersji na uniwersytecie w Monnesocie jako projekt semestralny z przedmiotu Sztuczna Inteligencja w 2007 roku [37]. Grupa projektowa składała się z czterech osób ale tylko jedna postanowiła go dalej rozwijać. Jeff Deitch opracował AI opartę o metodę Monte Carlo i sekwencje, Gabe Emerson i Erik Shimshock pracowali nad interfejsem i częścią silnika gry natomiast Michael Levin poza rysowaniem i strategii AI opartej o zagrożenie (Threat-Based Strategy) pracuje nad programem do dziś.



Rysunek 2.1. Graf zależności w programie

Niestety część odpowiedzialna za implementację silnika gry nie jest w programie zbyt dobrze wyodrębniona, mimo, że analizowna jest najnowsza (2.0) wersja programu [40]. Sytuacji nie poprawia fakt, że projekt był rozwijany w czystym języku C na Linux Debianie z bibliotekami GTK+. Kod odpowiadzialny za grafikę i interfejs jest mieszany z kodem odpowiedzialnym za logikę ale za to AI jest dosyć dobrze wyodrębnione i podzielone na 3 wspomniane wcześniej zaimplementowane w programie metody.

### Plik shared.h – definicje stałych i typów używanych w całym programie

Analizę należy zacząć od pliku nagłówkowego który jest załączany dyrektyrwą include w prawie każdym innym pliku z kodem. To tutaj znajdziemy definicję podstawowych stałych jak i ograniczeń dla nich. Poniżej przedstawiono doyć istotny fragment z początku pliku. Zdefiniowane są ograniczenia wartości parametrów k,p,q oraz wielkości planszy. Nie ma generalnie jakichś odgórnych wymagań co do wartości tych ograniczeń. Autorzy programu zdefiniowali je tak a nie inaczej między innymi po to aby obraz gry powstający w plikach definiujących interfejs użytkownika był w miarę czytelny.

/\* We limit the maximum values of these variables; note that these are not

arbitrary limits and should be modified with care \*/

#define MAX\_BOARD\_SIZE 59

#define MAX\_CONNECT\_K 12

#define MAX\_PLACE\_P 12

#define MAX\_START\_Q 6

#define MAX\_DEPTH 9

#define MAX\_BRANCH 32

Z tych stałych możemy też wyciągnąć ciekawy wniosek, że chociaż autorzy nazwali swój program grający „Connect-k”, to tak naprawdę wartości p i q również są parametryzowane. Program jest więc dosyć uniwersalny i bliski prawdziwej grze (m,n,k,p,q) [32] [24] [35]. Część z zadeklarowanych stałych dotyczy zachowania AI, ograniczona jest maksymalna głębokość przeszukiwania jak i maksymalna ilość przeszukiwanych gałęzi gry.

Poniżej przedstawiono nienazwany enum, który przechowuje stałe oznaczające m.in. różne typu pól. Jeżeli pole na planszy gry jest puste to przyjmuje wartość PIECE\_NONE, jeżeli zajętę przez pierwszego lub drugiego gracza to odpowiednio PIECE\_BLACK i PIECE\_WHITE. Ciekawostką jest fakt wprowadzenia w reprezentacji pól definicji pola błędnego PIECE\_ERROR, używane ono jest do otoczenia paskiem o szerokości 1 całej planszy tak aby łatwo wykrywać wychodzenie poza planszę lub źle ustawiony indeks w tablicy. Jest to rozwiązanie o tyle dobre, że w języku C w porównaniu do języków zarządzalnych jak C# nie ma standardowego sposobu wykrywania wyjścia poza granicę tablicy [41] [42] czy też innych tego typu błędów – tablica to po prostu wskażnik, co może prowdzić do wielu błędów. Wprowadzając definicję pola błędnego PIECE\_ERROR przynajmniej część z tych błędów wyjścia indeksu poza normalną planszę możemy uniknąć. Inne zadeklarowane stałe w typ enumie to m.in. znaczniki używane przez AI do określania np. ryzyka w strategii operatej o ryzyko jak i znaczniki używane przy przeszukiwaniu drzewa / potencjalnego drzewa gry.

enum {

PIECE\_ERROR = -1,

/\* Error pieces form a one tile deep border around the board \*/

PIECE\_NONE = 0,

PIECE\_BLACK,

PIECE\_WHITE,

/\* Empty and played pieces \*/

PIECES,

/\* Total number of normal pieces (2) \*/

PIECE\_SEARCHED,

PIECE\_SEARCHED\_MAX = PIECE\_SEARCHED + MAX\_DEPTH,

/\* Markers used by the search system \*/

PIECE\_THREAT0,

PIECE\_MARKER = PIECE\_THREAT0,

/\* These threat markers are usable by the AIs \*/

};

typedef int PIECE;

Do przechowywania informacji o typie pola na planszy (zajęte / wolne / błędne / znaczniki AI,..) używa się typu int poprzez odpowiednio zdefiniowany typedef. Wydaje się to na pozór nieoptymalne, gdyż jeżeli wartości byłyby ze zbioru (pierwszy gracz, drugi gracz, puste) lub nawet z polem błędu to i tak wystarczyłby spokojnie 8-bitowy char lub unsigned char. Jednak znaczniki używane przez AI np. oznaczenia ryzyka przyjmują w tej implementacji Connect-k wartości aż do INT\_MAX. Autorzy potrzebowali większej dziedziny aby lepiej wartościować wagi pozycji czy posunięc. Niemniej jednak można sobie wyobrazić efektywniejszą implementację, gdzie ewentualna informacja o zagrożeniu posunięcia jest trzymana gdzieś indziej a stany planszy są lżejsze. Innym rozwiązaniem byłoby chociaż ograniczenie dziedziny wag pozycji czy zagrożenia do przedziału i tak dosyć obszernego reprezentowanego przez np. typ short.

Poniżej przedstawiono kilka użytecznych makr zdefniowanych również w pliku shared.h. Prawdopodobnie wybrano definicję makr zamiast funkcji z powodów wydajnościowych. Makra te służą do wyznaczania wartości znaczników pól używanych przez AI (np. zagrożenie). Makro piece\_empty(p) sprawdza za to czy pole jest piste – nie jest to całkiem trywialne, ponieważ pole ejst pste wtedy kiedy jest oznaczone jako puste lub jest markerem używanym przez AI.

#define MAX\_THREAT (INT\_MAX - PIECE\_THREAT0)

/\* Highest value a threat marker can have \*/

#define PIECE\_THREAT(n) (PIECE\_THREAT0 + (n))

/\* This marker represents a threat n-turns (of that player) away \*/

#define piece\_empty(p) ((p) == PIECE\_NONE || (p) >= PIECES)

/\* Checks if a piece is an empty or a marker \*/

typedef unsigned int PLAYER;

/\* Type for AIs, this is the index of the AI entry in ai.c \*/

typedef unsigned int BCOORD;

/\* Type for board coordinates \*/

Powyżej przedstawiono również kolejen typedef’y używane tym razem jako odpowiednio indeks gracza AI w metodach sztucznej inteligencji i typ do trzymania współrzędnych na planszy. Po raz kolejny część kodu wydaje się nieoptymalna – współrzędne raczej nie potrzebują tek dużej dziedziny jaką jest unsigned int. Tym razem jednak nieoptymalność nie jest tylko pozorna ale rzeczywiście przy zdefiniowanej na początku pliku stałej ograniczającej maksymalny rozmiar planszy na 59 (MAX\_BOARD\_SIZE) nie potrzebujemy aż tak dużego typu.

### Plansza i stany gry

W dalszej części pliku shared.h znajduje się intuicyjnie podstawowa struktura reprezentująca stan gry. Jest nią struct Board reprezentujący planszę gry w danym momencie i wszystkich momentach w historii. Struktura ta jest niezwykle ważna, gdyż jest używana nie tylko do reprezentacji obecnego stanu gry ale jest także używana przez wszystkie algorytmu AI do reprezentacji wszystkich stanów w drzewie gry itp. Struktura trzyma m.in. wskażnik na poprzednią swoją wersję – czyli de facto poprzedni stan gry w zmiennej parent. Stosuje się także tutaj zmienną typu AllocChain reprezentującą pewien łańcuch alokacji obiektów (alokacji kolejnych stanów planszy np. w algorytmach symulacyjnych AI) niezwykle przydatną do czyszczenia pamięci po wykonaniu obliczeń przez AI – po wykonaniu obliczeń i wyznaczeniu ruchu usuwa się np. potencjalne przyszłe plansze z algorytmów symulacyjnych AI. Zmienna ac musi być jako pierwsza zadeklarowana w strukturze, gdyż często wykonujemy rzutowanie tych struktur aby łatwiej allokować /deallokować pamięć.

typedef struct AllocChain {

gboolean free;

/\* Is this object unallocated? \*/

unsigned int id;

/\* Each object has a unique id \*/

struct AllocChain \*next;

/\* Next object in the chain \*/

} AllocChain;

typedef struct Board {

AllocChain ac;

/\* Allocation chain must be the first member \*/

unsigned int moves\_left;

/\* How many moves the current player has left \*/

struct Board \*parent;

/\* The board preceeding this one in history \*/

gboolean won;

BCOORD win\_x1, win\_y1, win\_x2, win\_y2;

/\* On won boards, used to indicate where the winning line is \*/

PIECE turn;

/\* Whose turn it is on this board \*/

BCOORD move\_x, move\_y;

/\* The move to the next Board in history \*/

PIECE data[];

} Board;

/\* The board structure represents the state of the game board. Do NOT preserve

board pointers across games. \*/

Z punktu widzenia optymalności silnika niezwykle ważne jest jak trzymane są konkretne pola planszy. Tutaj jest to zwykła tablica o nazwie data, która trzyma wartości PIECE, które jak wiemy z poprzedniego podrozdziału są tak naprawdę typu int. Reprezentacja ta nie jest więcj optymalna jak byłby np. bitboard, trzeba jednak zauważyć, że ma pewne zalety [43] [44] i obliczenia / funkcje na niej działające są dzięki temu w miarę proste.

Ostatnim ciekawym fragmentem kodu (poza definicjami struktur dla AI, o czym będzie w dalszym podrozdziale) są funkcje z dyrektywą inline dla wydajności i makra służące do wygodnego dostępu do pól planszy do gry.

static inline PIECE piece\_at(const Board \*b, BCOORD x, BCOORD y)

{

return b->data[(y + 1) \* board\_stride + x + 1];

}

/\* Returns the piece at (x, y) on board b. If the coordinates are out of range,

this function will return PIECE\_ERROR. \*/

static inline void place\_piece\_type(Board \*b, BCOORD x, BCOORD y, PIECE type)

{

b->data[(y + 1) \* board\_stride + x + 1] = type;

}

#define place\_piece(b, x, y) place\_piece\_type(b, x, y, (b)->turn)

#define place\_threat(b, x, y, n) place\_piece\_type(b, x, y, PIECE\_THREAT(n))

/\* Places a piece on board b, overwriting any piece that was previously in that

place \*/

Z powodu użycia tablicy jednowymiarowej zamiast dwuwymiarowej oraz dodatkowego „paska błędu” wokół planszy niezbędne są pewne przeliczenia. Zmienna board\_stride reprezentuje rozmiar planszy z uwzględnieniem szerokości „paska błędu”, czyli jest to rozmiar planszy powiększony o dwa.

Do trzymania stanu użyto po prostu zmiennych globalnych typów opisanych wcześniej ze słówkę extern – tab aby w każdym pliku z dyrektywą include do shared.h można było się do tych zmiennych odnosić.

extern AllocChain \*board\_root;

extern gsize board\_mem;

/\* Variables for the allocation chain \*/

extern Board \*board;

/\* This is the current board. Do NOT modify it, that's cheating. :) \*/

extern int board\_size, board\_stride, move\_no, connect\_k, place\_p, start\_q;

/\* Board size (for all boards), moves in the game, connect\_k to win, place\_p

moves at a time, black has start\_q moves on the first move; do NOT modify

these directly! \*/

Ich właściwa deklaracja, implementacja i ustawienie znajduje się natomiast w pliku state.c.

Board \*board;

AllocChain \*board\_root = NULL;

int board\_size, board\_stride, move\_no, move\_last,

connect\_k = 6, place\_p = 2, start\_q = 1;

gsize board\_mem = 0;

Ja kaokkka hjfksahfja fsjdkhfas fsdkjfhsdjk vdsjvhds, hgjkdsgds, hgjdkshgjskdhgjksdhgjkasd i jakaa ahkgjsdhjkghasdjkgsadgds

/\* Convert a boord coordinate to alpha representation \*/

const char \*bcoord\_to\_alpha(BCOORD x)

{

static char buf[2][32];

static int which;

int i, divisor = 26;

which = !which;

for (i = 0; i < sizeof (buf[which]) - 1; i++) {

div\_t result;

result = div(x, divisor);

buf[which][i] = 'a' + result.rem \* 26 / divisor;

if (i)

buf[which][i]--;

x -= result.rem;

if (!x)

break;

divisor \*= 26;

}

buf[which][i + 1] = 0;

return g\_strreverse(buf[which]);

}

// Get a string representation of board x/y coordinates (d7, h16, etc)

const char \*bcoords\_to\_string(BCOORD x, BCOORD y)

{

static char buf[2][32];

static int which;

which = !which;

g\_snprintf(buf[which], sizeof (buf[which]), "%s%d",

bcoord\_to\_alpha(x), board\_size - y);

return buf[which];

}

/\* Convert a string representation to coordinates \*/

void string\_to\_bcoords(const char \*str, BCOORD \*x, BCOORD \*y)

{

\*x = 0;

\*y = 0;

while (\*str && \*str >= 'a' && \*str <= 'z') {

\*x \*= 26;

\*x += \*str - 'a';

str++;

}

while (\*str && \*str >= '0' && \*str <= '9') {

\*y \*= 10;

\*y += \*str - '0';

str++;

}

if (\*y)

\*y = board\_size - \*y;

}

# Implementacja generatora silników wybranej gry

Ramadan kokokdad, dashjaf afjkashfjka i fala moko. Ragodago miko miko regajlsafjas, aksgsa jkla. Jakaakaka moranranin polo mdiasdoas.

Ja kaokkka hjfksahfja fsjdkhfas fsdkjfhsdjk vdsjvhds, hgjkdsgds, hgjdkshgjskdhgjksdhgjkasd i jakaa ahkgjsdhjkghasdjkgsadgds. Moja fhjadkfhsjdkfhui fhdsiufhdsui i jok. Hjhfahfsa hjafsaj i jaaa, hjaksfhkjsa jvskld.

## Problem wyboru gry

Ramadan kokokdad, dashjaf afjkashfjka i fala moko. Ragodago miko miko regajlsafjas, aksgsa jkla. Jakaakaka moranranin polo mdiasdoas.

Ja kaokkka hjfksahfja fsjdkhfas fsdkjfhsdjk vdsjvhds, hgjkdsgds, hgjdkshgjskdhgjksdhgjkasd i jakaa ahkgjsdhjkghasdjkgsadgds. Moja fhjadkfhsjdkfhui fhdsiufhdsui i jok. Hjhfahfsa hjafsaj i jaaa, hjaksfhkjsa jvskld.

## Wybrana gra – szachy

Ramadan kokokdad, dashjaf afjkashfjka i fala moko. Ragodago miko miko regajlsafjas, aksgsa jkla. Jakaakaka moranranin polo mdiasdoas.

Ja kaokkka hjfksahfja fsjdkhfas fsdkjfhsdjk vdsjvhds, hgjkdsgds, hgjdkshgjskdhgjksdhgjkasd i jakaa ahkgjsdhjkghasdjkgsadgds. Moja fhjadkfhsjdkfhui fhdsiufhdsui i jok. Hjhfahfsa hjafsaj i jaaa, hjaksfhkjsa jvskld.

## Architektura projektu

Ramadan kokokdad, dashjaf afjkashfjka i fala moko. Ragodago miko miko regajlsafjas, aksgsa jkla. Jakaakaka moranranin polo mdiasdoas.

Ja kaokkka hjfksahfja fsjdkhfas fsdkjfhsdjk vdsjvhds, hgjkdsgds, hgjdkshgjskdhgjksdhgjkasd i jakaa ahkgjsdhjkghasdjkgsadgds. Moja fhjadkfhsjdkfhui fhdsiufhdsui i jok. Hjhfahfsa hjafsaj i jaaa, hjaksfhkjsa jvskld.

# Analiza wydajnościowo-poprawnościowa zaimplementowanego generatora silników gier

Ramadan kokokdad, dashjaf afjkashfjka i fala moko. Ragodago miko miko regajlsafjas, aksgsa jkla. Jakaakaka moranranin polo mdiasdoas.

Ja kaokkka hjfksahfja fsjdkhfas fsdkjfhsdjk vdsjvhds, hgjkdsgds, hgjdkshgjskdhgjksdhgjkasd i jakaa ahkgjsdhjkghasdjkgsadgds. Moja fhjadkfhsjdkfhui fhdsiufhdsui i jok. Hjhfahfsa hjafsaj i jaaa, hjaksfhkjsa jvskld.

# Podsumowanie

Ramadan kokokdad, dashjaf afjkashfjka i fala moko. Ragodago miko miko regajlsafjas, aksgsa jkla. Jakaakaka moranranin polo mdiasdoas.

Ja kaokkka hjfksahfja fsjdkhfas fsdkjfhsdjk vdsjvhds, hgjkdsgds, hgjdkshgjskdhgjksdhgjkasd i jakaa ahkgjsdhjkghasdjkgsadgds. Moja fhjadkfhsjdkfhui fhdsiufhdsui i jok. Hjhfahfsa hjafsaj i jaaa, hjaksfhkjsa jvskld.

Ramadan kokokdad, dashjaf afjkashfjka i fala moko. Ragodago miko miko regajlsafjas, aksgsa jkla. Jakaakaka moranranin polo mdiasdoas.

Ja kaokkka hjfksahfja fsjdkhfas fsdkjfhsdjk vdsjvhds, hgjkdsgds, hgjdkshgjskdhgjksdhgjkasd i jakaa ahkgjsdhjkghasdjkgsadgds. Moja fhjadkfhsjdkfhui fhdsiufhdsui i jok. Hjhfahfsa hjafsaj i jaaa, hjaksfhkjsa jvskld.

# Wykaz literatury

1. Łagodne wprowadzenie do analizy algorytmów, Marek Kubale, Wydawnictwo Politechniki Gdańskiej, 2016, ISBN: 978-83-7348-652-2
2. <http://en.wikipedia.org/wiki/Computer_chess>, (data dostępu 02.04.2016)
3. <http://en.wikipedia.org/wiki/Human–computer_chess_matches>, (data dostępu 02.04.2016)
4. <http://www.fierz.ch/history.htm>, (data dostępu 02.04.2016)
5. <https://en.wikipedia.org/wiki/Top_Chess_Engine_Championship>, (data dostępu 02.04.2016)
6. <https://en.wikipedia.org/wiki/International_Computer_Games_Association>, (data dostępu 02.04.2016)
7. <https://en.wikipedia.org/wiki/Computer_Olympiad>, (data dostępu 02.04.2016)
8. <https://icga.leidenuniv.nl/?page_id=1112>, (data dostępu 02.04.2016)
9. Zasady XIX Olimpiady Gier Komputerowych ICGA  
   <https://icga.leidenuniv.nl/wp-content/uploads/2016/03/Rules-ICGA-events-2016-3.pdf>, (data dostępu 02.04.2016)
10. <https://en.wikipedia.org/wiki/Solved_game>, (data dostępu 02.04.2016)
11. <https://icga.leidenuniv.nl/?page_id=1315>
12. <http://senseis.xmp.net/?ZenGoProgram>
13. <https://book.mynavi.jp/tencho6/>
14. <https://boardgamegeek.com/thread/609440/mohex-available-download>
15. <http://benzene.sourceforge.net>
16. <https://github.com/lukaszlew/MiMHex>
17. <https://github.com/ala/MiMHex>
18. <https://github.com/kdudzik/MiMHex>
19. <https://github.com/krzysiocrash/MiMHex>
20. <https://github.com/theolol/MiMHex>
21. <https://github.com/bartoszborkowski/mimhex>
22. <https://github.com/krzysiocrash/patterns>
23. <https://github.com/qelo/MiMHex>
24. <https://github.com/jakubpawlewicz/MiMHex>
25. <https://en.wikipedia.org/wiki/Komodo_(chess)>
26. <https://komodochess.com>
27. <https://komodochess.com/Komodo10-50a.htm>
28. <https://komodochess.com/Komodo9-43a.htm>
29. <https://komodochess.com/pub/komodo-8.zip>
30. <https://stockfishchess.org>, (data dostępu 02.04.2016)
31. <https://github.com/official-stockfish/Stockfish>, (data dostępu 02.04.2016)
32. <https://en.wikipedia.org/wiki/M,n,k-game>
33. <https://en.wikipedia.org/wiki/Tic-tac-toe>
34. <https://en.wikipedia.org/wiki/Gomoku>
35. <https://en.wikipedia.org/wiki/Connect6>
36. A New Family of k-in-a-row Games, I-Chen Wu and Dei-Yen Huang, Department of Computer Science and Information Engineering, National Chiao Tung University, Hsinchu, Taiwan  
    <http://www.connect6.org/k-in-a-row.pdf>
37. <http://risujin.org/connectk/>
38. <http://www.connect6.org>
39. <https://bitbucket.org/BadRobot/connect6/src>
40. <http://risujin.org/pub/connectk/connectk-2.0.tar.gz>
41. <http://stackoverflow.com/questions/671703/array-index-out-of-bound-in-c/671709#671709>
42. <https://msdn.microsoft.com/en-US/library/system.indexoutofrangeexception(v=vs.110).aspx>
43. <https://en.wikipedia.org/wiki/Bitboard>
44. <https://en.wikipedia.org/wiki/Bit_field>

# Wykaz rysunków

**No table of figures entries found.**

# Wykaz tabel

**No table of figures entries found.**

Dodatek A: Tytuł dodatku A

Ramadan kokokdad, dashjaf afjkashfjka i fala moko. Ragodago miko miko regajlsafjas, aksgsa jkla. Jakaakaka moranranin polo mdiasdoas.

Ja kaokkka hjfksahfja fsjdkhfas fsdkjfhsdjk vdsjvhds, hgjkdsgds, hgjdkshgjskdhgjksdhgjkasd i jakaa ahkgjsdhjkghasdjkgsadgds. Moja fhjadkfhsjdkfhui fhdsiufhdsui i jok. Hjhfahfsa hjafsaj i jaaa, hjaksfhkjsa jvskld.