Przedmiot: Metody Inżynierii Wiedzy – Projekt
Tytuł: Szachy komputerowe

Piotr Miązek prezesair@op.pl
Piotr Michalak piotrmichalak@gmail.com
Dawid Ostrowski avedave@gmail.com
AiR, IV rok, lato 2005

26.09.2005

Spis treści

1. Sformułowanie problematyki projektu

Programowanie gier jest jednym z klasycznych tematów badań nad sztuczną inteligencją, a szachy zajmują wśród programowanych gier ważne miejsce. Owo szczególne miejsce zawdzięczają szachy swojej popularności, a także względom historycznym[1].

Przez ponad 50 lat programowanie szachów dorobiło się swojej pokaźnej teorii i praktyki. Mimo to, nikomu do tej pory nie udało się skonstruować programu, którego zasady działania opierałyby się na zasadach funkcjonowania intelektu gracza-człowieka. Udało się natomiast skonstruować silnie grające programy, wykorzystując matematyczne modele gry w szachy. Obecnie najlepsze programy szachowe opierają się na zasadzie brutalnej siły (ang. *brute force*) tzn. na wykorzystaniu szybkości komputera przy zastosowaniu pewnego relatywnie nieskomplikowanego algorytmu. Przy tym wszystkim nastąpił ogromny rozwój wielu dziedzin sztucznej inteligencji, a programowanie gier stało się samoistną gałęzią badań.

Opierając się na teorii gier, szachy można traktować jako grę dwuosobową, skończoną, z pełną informacją, o sumie zerowej.
Poniżej przedstawione są krótko powody takiego zaklasyfikowania szachów:

- W grze bierze udział dwóch graczy (gra dwuosobowa).
- Każda rozgrywka musi się kiedyś skończyć (gra skończona).
- Obaj gracze mają dokładną informację o sytuacji w grze (gra z pełną informacją).
- Cele przeciwników są dokładnie przeciwstawne, zwycięstwo jednego oznacza porażkę drugiego (gra o sumie zerowej).

Do tej samej kategorii przynależy wiele popularnych gier: GO, warcaby, kółko i krzyżyk, reversi itp. Wszystkie te gry programowane są najczęściej z pomocą algorytmów minimaksowych. Jednak każda z nich jest w końcu inną grą, stąd odmiennie dla każdej gry konstruuje się jej funkcję heurystyczną. Nadto zasadniczym wyróżnikiem każdej z gier jest liczba możliwości, które można podjąć przy każdym posunięciu.
Dla gier „małych", takich jak warcaby, udało się już dość dawno metodą „brutalnej siły" pokonać człowieka - mistrza świata. Dla gry bogatszej możliwościami - szachów ten poziom uzyskano dopiero niedawno. Natomiast dla GO, gdzie na każdym posunięciu trzeba wybierać pomiędzy ok. 300-ma możliwościami nie udało się nawet zbliżyć do poziomu gracza-pasjonata.

Gry dwuosobowe, skończone, z pełną informacją, o sumie zerowej stanowią jeden z najprostszych przypadków gry w ogóle. Dla porównania brydż będąc grą bez pełnej informacji jest już znacznie trudniejszy do zaprogramowania. Obecnie zawieszono większość prób skonstruowania programów przypominających swym działaniem rozumowanie człowieka, a skupiono się na uzyskaniu coraz wyższej wydajności modułów elektronicznych oraz udoskonalaniu istniejących algorytmów.

[1] Często uznaje się pracę C.Shannona *„Programming computer for playing chess"* z roku 1950-go za jedną z fundamentalnych prac nad sztuczną inteligencją

2. Cel projektu

Głównym celem projektu było praktyczne zaznajomienie się autorów z problematyką tworzenia programów grających i rozwinięcie umiejętności zdobytych wcześniej - podczas tworzenia komputerowej wersji gry „kółko i krzyżyk" (2 rok studiów).

Na drodze do osiągnięcia powyższego celu należało pogłębić znajomość następujących zagadnień:

- Reprezentacja planszy i figur.
- Generowanie możliwych posunięć.
- Przeszukiwanie drzewa gry.
- Funkcja oceniająca.
- Interakcja z użytkownikiem.

3. Opis realizacji

3.1. Koncepcja

Przez lata, powstało wokół szachów wiele opracowań dotyczących sposobu rozgrywki, strategii, rozwiązań poszczególnych etapów gry (debiut, gra środkowa, końcówka). Ponadto ludzie chcący doskonalić swoje umiejętności gry, przeprowadzają dokładną analizę partii rozgrywanych przez arcymistrzów. Wszystko to tworzy ogromną bazę wiedzy, której chyba żaden człowiek nie przyswoił w całości. Od ilości tej wiedzy zawartej w programie, bezpośrednio zależy jakość funkcji oceniającej, a co za tym idzie - zaawansowanie komputerowego gracza.

Algorytm Minimaks na podstawie symulacji ruchów obu przeciwników można łatwo przerobić na program rozgrywania gry w szachy. Szachy są jednak zbyt skomplikowaną grą do bezpośredniego zastosowania Minimaksu (można by to zrobić na przykład w przypadku dziewięciopolowego kółka i krzyżyka), gdyż drzewo gry w szachy ma ok. 1043 węzłów. Można jednak dokonać przeglądu jedynie fragmentu drzewa gry.

Algorytm alfa-beta wspomagający wybór odpowiedniego fragmentu drzewa gry, opiera się na obserwacji, że dla wyznaczenia minimaksowej wartości pewnego drzewa nie potrzeba znać minimaksowych wartości wszystkich jego węzłów. W algorytmie tym nie rozpatruje się tych pozycji, które nie wpłyną na ocenę sytuacji. Procedura Alfabeta działa przy założeniu, że mamy do czynienia z sekwencyjnymi obliczeniami, istnieje jednak jej odpowiednik dla obliczeń równloległych.

Do oceny układu bierek na planszy potrzebna jest funkcja oceniająca (zwana też funkcją heurystyczną). Człowiek-szachista w swym procesie myślenia dokonuje pewnych ocen końcowych pozycji wariantów, które obliczył: „łatwa wygrana", „co najmniej remis", „przegrana pozycja z szansami na remis ok. 30 proc" itp. Funkcja oceniająca przyporządkowuje danej pozycji wartość z pewnego zbioru - w przypadku algorytmów minimaksowych wartość

liczbową. W założeniu implementacja programowa funkcji oceniającej nie dokonuje żadnych obliczeń drzewa gry. Idealną funkcją oceniającą jest taka funkcja, która zawsze zwraca jako wynik dokładną wartość minimaksową pozycji. Oczywiście skonstruowanie takiej funkcji na ogół nie jest możliwe. Konieczne jest zatem konstruowanie funkcji oceniających opartych na przesłankach heurystycznych.

3.2. Wybrane narzędzia

Projekt zaimplementowano przy użyciu języków C / C++. Za tym wyborem przemawiały następujące fakty:

- Doświadczenie autorów zdobyte podczas prac nad grą „kółko i krzyżyk", która została napisana przy użyciu C / C++.
- Wydajność wspomnianych języków - jest sprawą kluczową dla szybkości przeszukiwania drzew rozwiązań, a co za tym idzie, dla szybkości programu.
- Dostępność wielu materiałów oraz przykładowych implementacji, na których autorzy mogli się wzorować.

3.3. Realizacja

W niniejszym rozdziale opisano bardziej szczegółowo poszczególne decyzje projektowe i implementacyjne.

3.3.1. Generowanie możliwych posunięć

Działanie modułu opiera się na listowych strukturach danych. Wybór listowych struktur danych jest dla algorytmów grafowych naturalny. Generator posunięć programu przegląda szachownicę po kolei od lewego górnego rogu do prawego dolnego wierszami i przy napotkaniu figury gracza będącego na posunięciu generuje wszystkie możliwe ruchy tej figury.

Generator udostępnia trzy procedury: generator posunięć zgodnych z regułami gry, znacznie szybszy generator pseudoposunięć oraz generator pseudoposunięć-bić. Dla list posunięć generowanych przez poszczególne procedury generujące zdefiniowane są odpowiednie procedury sortujące.

(a) Generator posunięć zgodnych z regułami gry

Procedura ta generuje wszystkie posunięcia zgodne z regułami szachów. Wykorzystywana jest jedynie przez funkcje logiczne określające, czy dana pozycja jest matem lub patem lub szachem.

(b) Generator pseudoposunięć

Pseudoposunięcia to wszystkie zgodne z przepisami posunięcia, oraz takie, które pozostawiają króla pod biciem lub podstawiają króla pod bicie. Dopuszczenie podstawki króla czyni generator pseudoposunięć kilkakrotnie szybszym od generatora posunięć zgodnych z przepisami. Generator pseudoposunięć wykorzystywany jest w rejonie pełnego przeszukiwania drzewa gry. W regionie tym lista psuedoposunięć jest sortowana w kolejności:

- Bicia w kolejności wartości materialnej zdobytego materiału
- Pozostałe posunięcia

Wszystkie bicia sortowane są według wartości zysku materialnego. Zyskiem materialnym związanym z danym posunięciem nazywać będę wartość materialną zbitej bierki, a w przypadku promocji wartość promowanej bierki minus wartość znikającego z szachownicy pionka.

(c) Procedura generująca pseudoposunięcia-bicia

Procedura generująca psudoposunięcia-bicia wykorzystywana jest w regionie selektywnego przeszukiwania drzewa. Z procedurą tą związany jest jej parametr „ΔM" oraz stała „TOLERANCJA".

Do listy pseudoposunięć-bić zdefiniowana jest procedura usuwająca wszystkie bicia nie dające zysku materialnego równego co najmniej: („ΔM" - „TOLERANCJA"). Ma ona za zadanie wymusić, by w trakcie przeszukiwania ciągów wymian na zabicie np. skoczka, przeciwnik odpowiadał również zabiciem co najmniej skoczka. Parametr „ΔM" pokazuje dotychczasowy bilans materialny od rozpoczęcia przeszukiwania ciągów wymian (chodzi o przeszukiwanie selektywne, czyli wariant forsowny). Należy jednak przy tym narzucić pewną tolerancję, pozwalającą kontynuować przeszukiwanie, nawet jeśli nie da się wyrównać bilansu materialnego. Innymi słowy, dopuszczalne jest, by na zbicie wieży odpowiedzieć zbiciem tylko skoczka. W przeciwnym razie program nie byłby w stanie obliczyć nawet prostych kombinacji z poświęceniem niewielkiej ilości materiału. Nie potrafiłby się nawet pogodzić z wymianą gońca na skoczka, gdyż goniec ma w programie nieco wyższą wartość materialną niż skoczek. Aktualnie wartość stałej TOLERANCJA ustawiona jest na 125, czyli wartość materialną jednego i jednej czwartej pionka. Dla trzech specyficznych sytuacji szachów: szacha, mata i pata, zdefiniowane zostały odpowiednie funkcje logiczne. Zdefiniowane zostały także różne procedury niezbędne w programie: wykonywanie posunięcia, sprawdzanie czy dane posunięcie jest w danej pozycji możliwe, porównywanie dwóch pozycji itp.

3.3.2. Przeszukiwanie drzewa gry

Przeszukiwanie w programie jest oparte na standardowej zasadzie regionów. Program dzieli drzewo na trzy regiony przeszukiwania. Pierwszy z nich to oczywiście region pełnego przeszukiwania drzewa gry. Po nim następują dwa regiony selektywnego przeszukiwania. Region przeszukiwania pełnego będzie dalej nazywany bazowym.

W przeszukiwaniu program korzysta z generatora pseudoposunięć. Podstawka króla wykrywana jest dopiero w trakcie dalszego przeszukiwania w momencie, gdy wykryta zostanie możliwość bicia króla. Sterowanie powraca wtedy do poprzedniej pozycji na ścieżce przeszukiwania i uruchamiana jest procedura rozpoznająca, czy dopuszczenie do bicia króla nastąpiło z powodu mata, pata czy też zwykłej podstawki.

Funkcja oceniająca programu składa się z dwóch części: funkcji oceniającej materialnie, oraz funkcji oceniającej pozycyjnie. Przeszukiwanie działa według następującej zasady. W

pozycjach końcowych regionu bazowego obliczana jest funkcja oceniająca pozycyjnie, następnie do wyniku tej funkcji dodawany jest wynik przeszukiwania selektywnego, przy czym przeszukiwanie selektywne korzysta z funkcji oceniającej materialnie.

Drugi z regionów przeszukiwania selektywnego, stanowi „wariant forsowny" programu. Wariant forsowny nie może dać w wyniku mniej niż ocena materialna pozycji, w której rozpoczęto przeszukiwanie selektywne. W regionie tym brane są pod uwagę jedynie bicia. Natomiast pierwszy region przeszukiwania selektywnego ma głębokość jednego półruchu i stanowi jakby płynną granicę między przeszukiwaniem pełnym a selektywnym. W zależności od danej pozycji w regionie tym dokonuje się albo przeszukiwanie pełne jak w regionie bazowym albo selektywne jak w wariancie forsownym.

Program korzysta z iteracyjnie pogłębianego przeszukiwania. Pogłębiana jest jednak jedynie głębokość regionu bazowego począwszy od głębokości 2. Maksymalne głębokości regionów selektywnych są stałe i wynoszą 1 dla regionu pierwszego i 6 dla drugiego. Pierwsza iteracja musi zawsze zostać wykonana, niezależnie od ewentualnego przekroczenia limitu czasu. Przeszukiwanie kończy się z chwilą upłynięcia limitu czasowego, który jest jednym z parametrów programu wprowadzanym przez użytkownika.

3.3.3. Funkcja oceniająca

Funkcja oceniająca korzysta z własnych struktur danych opisujących własności pozycji z aktualnego stanu przeszukiwania. Struktury te pozwalają na szybsze obliczenie wyniku f.o. Przykładem tych struktur są listy zawierające pozycje wszystkich poszczególnych typów bierek. Struktury te modyfikowane są w trakcie przeszukiwania za pomocą dwóch funkcji: „MoveFunOcen" oraz „UnMoveFunOcen". Jedna z nich służy do modyfikacji struktur danych f.o. przy wykonaniu posunięcia, druga przy cofnięciu. Tak więc schemat modyfikowania struktur danych funkcji oceniającej odpowiada zasadzie działania metod inkrementalnodekrementalnych. Sam wynik f.o. jest liczbą całkowitą z zakresu <-32768 , +32767>. Jak wspomniano wcześniej f.o. składa się zasadniczo z dwóch części: oceny materialnej i oceny pozycyjnej. F.o. jest funkcją „symetryczną", tzn. w pozycjach powstałych przez zamianę figur białych na czarne, czarnych na białe i odbiciu symetrycznym wzdłuż linii środkowej szachownicy funkcja oceniająca zmienia jedynie znak.

Jak wspomniano w poprzednim rozdziale ocena pozycyjna obliczana jest w pozycjach końcowych obszaru pełnego przeszukiwania, natomiast ocena materialna jest zadaniem przeszukiwania selektywnego. Prowadzi to do powstania efektu „efektem odkładania gróźb". Otóż program odkłada możliwie jak najdalej posunięcia wygrywające materiał, o ile tylko możliwość ich wykonania nie wykracza poza zasięg jej przeszukiwania selektywnego. Na przykład jeżeli w końcówce pionowej program ma możliwość promowania hetmana, ale może to odłożyć o jedno posunięcie, to najpierw wykona posunięcie poprawiające ocenę pozycyjną. Rzecz w tym, że promując hetmana jeszcze w rejonie pełnego przeszukiwania program pozbawiłaby się dodatkowych punktów za zaawansowaną pozycję pionka i dlatego preferuje wykonanie promocji dopiero w obszarze selektywnym. Prowadzi to czasem do podejmowania błędnych decyzji. Dla przykładu w pewnym rodzaju końcówek pionowych zwanym przez szachistów „wyścigami", jak najszybsze promowanie hetmana ma decydujące znaczenie.

(a) Ocena materialna

Ocena materialna jest sumą wartości materialnej znajdujących się na szachownicy bierek. Wartości poszczególnych bierek zostały ustalone na:

- pionek = 100
- skoczek = 290
- goniec = 310
- wieża = 500
- hetman = 950
- król = 20.000

Oczywiście wartość czarnych bierek należy brać z minusem. Wartość materialna króla wykorzystywana jest jedynie w sytuacji, gdy w trakcie przeszukiwania selektywnego program natrafi na pozycję z możliwością zbicia króla w jednym posunięcia. program wartościuje taką pozycję na 20.000 i następuje powrót sterowania. Ponadto zostały wprowadzone dodatkowe specjalne wartościowania liczby pionków na szachownicy oraz wartościowanie przewagi materialnej w stosunku do sumy pozostałego na szachownicy materiału. Celem tych wartościowań jest z jednej strony zachęcenie programu do wymian w sytuacji posiadania przewagi materialnej (unikania wymian w przeciwnym wypadku), z drugiej strony „zniechęcenie" do wymian pionków w tejże sytuacji, gdyż pionki stanowią wtedy potencjalne hetmany. Wartościowanie to wyraża się poniższymi wzorami.

(1) $LP * 30000/(LP + 1) * M$

(2) $MA * 590/M$

We wzorach tych poszczególne identyfikatory oznaczają:

LP - liczba pionków strony mającej przewagę materialną

M - suma wartości materiału na szachownicy, biorąc czarne bierki wyjątkowo z plusem, nie licząc króli.

MA - wartość przewagi materialnej (z plusem).

Jeśli przewagę materialną mają białe, to wartości powyższe dodaje się do wyniku funkcji oceniającej materialnie, a jeśli przewagę mają czarne odejmuje. Liczba 590 we wzorze (2) została tak dobrana, aby z jednej strony wartość wyrażenia była jak największa, a z drugiej strony aby w pozycji: „K+G+S":„K+p" strona silniejsza nie wymieniła skoczka na pionka przeciwnika. Powstaje oczywiście pytanie, czy nie należałoby traktować tych własności jako pozycyjnych i włączyć raczej w ocenę pozycyjną. Jednak takie rozwiązanie wzmogłoby jeszcze „efekt opóźniania gróźb".

(b) Ocena pozycyjna

Klasyfikacja pozycji wyjściowej.
Program klasyfikuje pozycję wyjściową. Nie ma jednak żadnej bazy funkcji oceniających - jak to jest sugerowane w niektórych przykładach - a jedynie bezpośrednio w kodzie programu poszczególne składniki oceny pozycyjnej zależą od wyniku klasyfikacji. Program przyjmuje podział możliwych pozycji na pięć rodzajów:

(1) debiut

(2) gra środkowa

(3) wczesna końcówka

(4) końcówka

(5) matowanie

(ad 1) Pozycja jest klasyfikowana jako debiut, jeżeli co najmniej czternaście bierek, nie licząc króli, znajduje się na swoich pozycjach wyjściowych.

(ad 2) Pozycja jest klasyfikowana jako gra środkowa, jeżeli nie jest debiutem, oraz suma materiału na szachownicy, nie licząc króli i licząc czarne bierki na plus, wynosi co najmniej 4 300.

(ad 3) Pozycja jest klasyfikowana jako „wczesna końcówka", jeśli na szachownicy jest pomiędzy 4 300 a 2 800 p. materiału.

(ad 4) Pozycja jest klasyfikowana jako końcówka, jeżeli na szachownicy pozostało mniej niż 2 800 materiału.

(ad 5) Pozycja jest klasyfikowana jako matowanie, jeżeli jedna ze stron osiągnęła przewagę materialną co najmniej 450, ma co najmniej jednego hetmana lub wieżę, ewentualnie jeśli ich nie ma, to nie ma przy tym ani jednego pionka. Ten ostatni warunek wynika z założenia, że mając jedynie lekkie figury i piony, łatwiej jest najpierw promować hetmana, a dopiero potem matować. Matowanie jest fazą posiadającą własną, specyficzną ocenę pozycyjną. Omawiane dalej składowe oceny pozycyjnej nie odnoszą się do fazy matowania. Ze względu na oszczędność czasu, program dokonuje jedynie klasyfikacji pozycji wyjściowej. Bywa to przyczyną błędnych ocen pozycji końcowych przeszukanego drzewa, jeżeli na ścieżce przeszukiwania nastąpiła zupełna zmiana sytuacji na szachownicy.

Położenie poszczególnych bierek.

Większość składników oceny pozycyjnej to wartościowanie położenia poszczególnych figur względem środka szachownicy oraz względem innych bierek. Działanie funkcji oceniającej jest takie, że dla każdej bierki liczone jest wartościowanie jej położenia i wynik dodawany jest do pewnej zmiennej globalnej. Poniżej podano dwa używane dalej wzory.

$$BliskoscCentrum([i,j]) = 7 - (|3.5 - i| + |3.5 - j|)$$

We wzorze powyższym i oraz j oznaczają współrzędne pola na szachownicy. W programie przyjmuje się, że współrzędne szachownicy są liczbami z przedziału $< 0,7 >$. Tak więc np. pole „c-6", to w programie program pole „2-5". Jak widać powyższy wzór przyjmuje wartości z przedziału $< 0,6 >$.

$$BliskoscPol([i1,j1],[i2,j2]) = 7 - (|i1 - i2| + |j1 - j2|)$$

gdzie [i,j] oznaczają współrzędne pola, podobnie jak w podanym wyżej wzorze. Wzór na bliskość pól przyjmuje wartości z przedziału $< -7,6 >$ (nie ma sensu obliczanie odległości dwóch identycznych pól).

Pionki

Jak wiadomo „struktura pionowa[2] jest duszą szachów". Program rozpoznaje następujące cechy struktury pionowej:

(1) Pionki izolowane

(2) Pionki zdwojone

(3) Pionki w centrum

(4) Dochodzące pionki

(5) Specjalna ocena debiutowa

(ad 1) Za każdego izolowanego białego pionka program odejmuje wartość 20. Oczywiście w przypadku czarnych bierek program postępuje przeciwnie niż w przypadku białych, tzn. np. za izolowanego czarnego pionka program dodaje 20 do oceny pozycji.

(ad 2) Za każdego zdublowanego pionka program odejmuje wartość 10.

(ad 3) W debiucie i grze środkowej dla każdego pionka program wartościuje jego odległość od centrum szachownicy wg. wzoru:

$BliskoscCentrum([i, j])$

Dodatkowo za każdego pionka znajdującego się na jednym z pól „d4", „d5", „e4", „e5" program dolicza do oceny 15, a za pionki znajdujące się na polach „c4", „c5", „d3", „d6", „e3", „e6", „f4", „f5" dolicza 10.

(ad 4) W końcówkach dla każdego pionka obliczane jest jego zawansowanie wg. wzoru:

$WSPCZYNNIK * (j - 1)2$ dla białych pionków
$WSPCZYNNIK * (6 - j)2$ dla czarnych pionków

gdzie j oznacza współrzędną poziomą pionka na szachownicy.

Powyższa wartość dodawana jest do oceny pozycyjnej. $WSPCZYNNIK$ przyjmuje wartość 2 we wczesnej końcówce i 4 w końcówce. Ponadto program oblicza liczbę dokonanych na aktualnej ścieżce promocji pionków i za każdą promocję dodaje/odejmuje do/od oceny wartość: 30 * $WSPCZYNNIK$ W przeciwnym razie program odkładałaby możliwe jak najdalej promowanie pionków, aby nie stracić pozycyjnego zysku z pionka na siódmej linii („efekt opóźniania gróźb").

(ad 5) Jeżeli w debiucie, oba z pionków sprzed hetmana i króla nie zostały rozwinięte, to program odejmuje od oceny 50.

Skoczki

Program rozpoznaje tylko jedną własność położenia skoczka:

[2]Struktura pionowa - rozmieszczenie pionów na szachownicy

(1) Odległość od centrum.

Własność jest badana we wszystkich fazach gry (oprócz matowania) i do oceny dodaje sięwartość: $4 * BliskoscCentrum$

Gońce

Dla gońców ocena liczona jest według następującego wzoru:

$ABB = EM + OM + DL + OW$

gdzie:

EM - punkty za obecność bierek przeciwnika na diagonalach gońca: Skoczek=3, Goniec=4, Wieża=5, Hetman=9, Król=10

OM - punkty za obecność własnych bierek na diagonalach gońca: Skoczek=1, Goniec=1, Wieża=2, Hetman=3

DL - długość diagonali gońca (Liczba możliwych posunięć gońca z danego pola na pustej szachownicy) - 7

OW - punkty za własne pionki na diagonali gońca Za każdego pionka z tyłu gońca: +1 Za każdego pionka z przodu gońca: jeśli pionek jest na własnej połowie: -5 jeśli pionek jest na połowie przeciwnika: +1 Tak obliczona ocena związana z danym gońcem mnożona jest przez współczynnik i dodawana do oceny całkowitej. Współczynnik zależy od fazy gry i wynosi: 3 dla gry środkowej i 1 dla wczesnej końcówki. W końcówkach i w debiucie pozycja gońca nie jest wartościowana.

Wieże

Dla wieży rozpatrywane są następujące elementy oceny pozycyjnej:

(1) Wieża na otwartej linii.

(2) Bliskość króla przeciwnika.

(3) Mobilność wieży.

(4) Wieże złączone.

(5) Wieża na siódmej linii.

(ad 1) Dla każdej wieży na otwartej linii dodawane jest do oceny 10, a dla wieży na półotwartej linii 4.Własność nie jest brana pod uwagę w końcówkach.

(ad 2) W grze środkowej i końcówce wartościowana jest odległość wieży od króla przeciwnika zgodnie ze wzorem:

$2 * BliskoscPol(Wieza, KrolWroga)$

Wartość ta dodawana do oceny pozycji (wartość powyższa może być ujemna). Własność nie jest brana pod uwagę w debiucie.

(ad 3) Mobilność wieży oblicza się ze wzoru:

2∗Liczba atakowanych przez wieżę pól

Wartość powyższa dodawana jest do oceny.

(ad 4) Za każdą parę złączonych wież doliczane jest 20 punktów.

(ad 5) Dla wieży na siódmej linii dodaje się do oceny 27 punktów. Jeżeli wieża na siódmej linii odcina króla na ósmej to dodaje się jeszcze 13.

Hetmany

Dla hetmanów rozpatrywane są następujące elementy oceny pozycyjnej:

(1) Hetman w centrum.

(2) Bliskość króla przeciwnika.

(ad 1) Punkty za odległość od centrum liczone są według wzoru:

$WSPCZYNNIK * BliskoscCentrum$

i dodawane do oceny, przy czym w debiucie $WSPCZYNNIK$ przyjmuje wartość -2, w grze środkowej 0, we wczesnej końcówce 2, i w końcówce 4.

(ad 2) Odległość hetmana od króla przeciwnika liczona jest według wzoru:

$3 * BliskoscPol(Hetman, KrolPrzeciwnika)$

Wartość powyższa dodawana jest do oceny w grze środkowej i w końcówce.

Król

Rozpatrywane są następujące elementy oceny pozycji króla:

(1) Bezpieczeństwo króla.

(2) Centralizacja króla.

(ad 1) Bezpieczeństwo króla jest bardzo ważnym czynnikiem w debiucie i w grze środkowej. Nie jest ono wartościowane w końcówkach, gdzie znaczenia nabiera centralizacja króla. Program ocenia bezpieczeństwo króla na podstawie dwóch własności: położenia króla i obecności pionków przed królem. Niżej podane są wartości dodawane przez program do oceny w zależności od położenia białego króla (dla czarnego króla należy brać przeciwne wartości dla przeciwległych pól).

b1,g1 : 25b2,a1,g2,h1 : 15c1,a2,h2 : 10 f1 : 5 d1 : -5 c2-f2 : - 40 a3-h3 : - 100 a4-h4 : - 200 pozostałe : - 400

Ponadto za każdego pionka znajdującego się na jednym z trzech pól bezpośrednio przed królem (lub dwóch, jeżeli król stoi na bandzie) program dolicza do oceny 15. Jeżeli jednak wszystkie trzy pola przed królem znajdującym się na pierwszej linii są zajęte przez pionki. to program odlicza od oceny 45, niwelując punkty za obecność tych pionków. Chodzi o uniknięcie słabości pierwszej linii i wymuszenie na programie zrobienia królowi „furtki".

(ad 2) W końcówkach miejsce wartościowania bezpieczeństwa króla zajmuje wartościowanie odległości od centrum szachownicy. Dodawana do oceny wartość obliczana jest według wzoru:

$$WSPCZYNNIK * BliskoscCentrum$$

gdzie $WSPCZYNNIK$ przyjmuje wartość 3 dla wczesnej końcówki i 8 dla końcówki.

4. Podsumowanie

4.1. Wnioski

Realizacja projektu okazała się dużym wyzwaniem, wymagającym znacznego wysiłku w następujących aspektach:

- Wyszukiwanie informacji z dziedziny gier, a w szczególności szachów.
- Wyszukiwanie informacji z dziedziny modelowania matematycznego gry w szachy, algorytmów generowania posunięć, funkcji oceniających itp.
- Próby implementacyjne w C / C++ / Java.
- Sporządzenie dokumentacji (LATEX).

Jednym z głównych, pozytywnych efektów projektu, było poznanie i zgromadzenie w jednym miejscu dość szerokiej wiedzy z zakresu komputerowego modelowania gry w szachy. Przytoczone w niniejszym dokumencie analizy, algorytmy i inne rozwiązania, stanowią dobry punkt

wyjściowy przy rozpoczynaniu prac nad implementacjami szachów.

Sama implementacja, stworzona na potrzeby projektu, pozostawia niestety wiele do życzenia. Osiągnięto co prawda rezultat w postaci programu potrafiącego rozgrywać partię w sposób nietrywialny, lecz należy otwarcie stwierdzić, iż program posiada wciąż liczne błędy i niedoróbki. Interfejs użytkownika jest bardzo ubogi - jak większość interfejsów tekstowych. Nie ma możliwości zapisu czy odczytu rozgrywki oraz innych możliwości, których użytkownik zwykle oczekuje od „gry".

4.2. Obserwacje

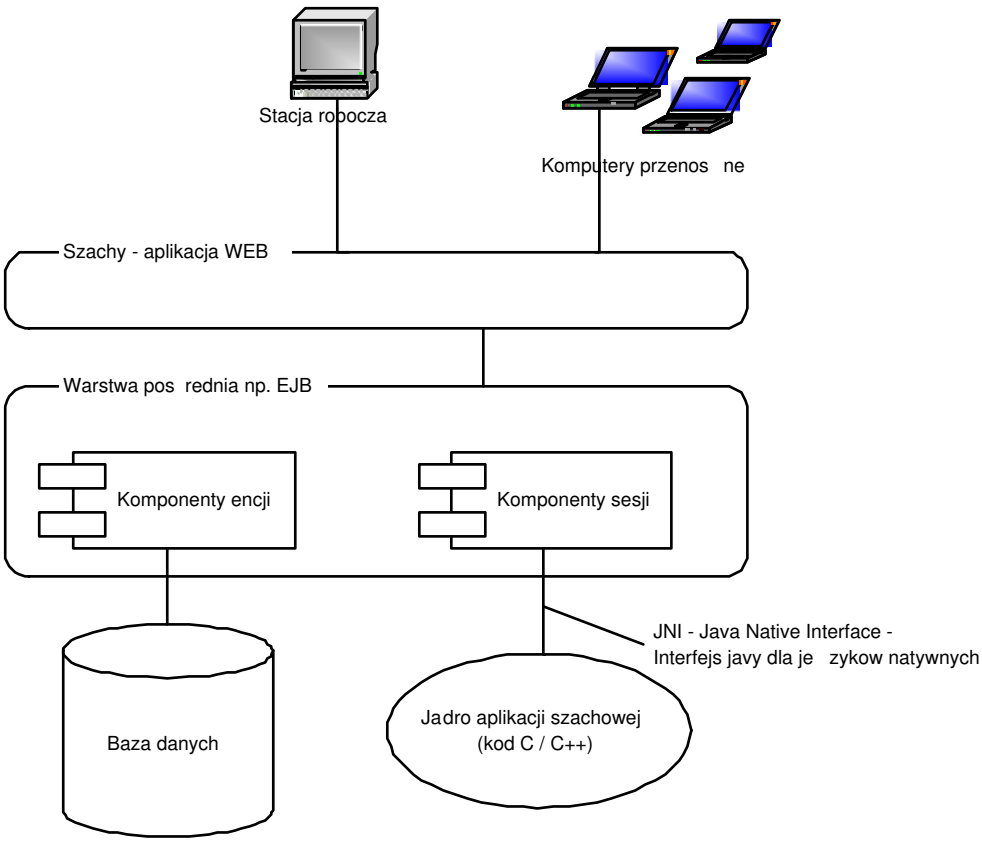
Podczas działania programu zdarzają się problemy z pamięcią. W zależności od ilości dostępnej pamięci operacyjnej oraz ilości analizowanych poziomów drzewa rozwiązań, zdarzają się przypadki zawieszenia programu. Wykorzystanie języka C / C++ dało z jednej strony szybkość przetwarzania, a z drugiej niejako „umożliwiło" wystąpienie problemów z pamięcią.

Nieliczne próby implementacyjne z wykorzystaniem języka Java ujawniły znany skądinąd fakt, że w porównaniu do kodu w C / C++, kod Javy wykonuje się od kilku do kilkunastu razy wolniej. W zastosowaniach wymagających wielu obliczeń - takich jak gra w szachy - powyższego faktu nie można pominąć.

4.3. Przyszłe prace

Rezultaty uzyskane podczas prac nad niniejszym projektem stanowią mocną podstawę pod rozwój aplikacji szachowej. Można rozważyć wydzielenie jądra systemu stanowiącego jedynie jednostkę odpowiadającą za ruchy gracza - komputera oraz określenie przejrzystych interfejsów komunikacji z takim jądrem. Uzyskując w ten sposób niezależność jądra decyzyjnego, umożliwiamy swobodny rozwój aplikacji użytkownika końcowego bez narzucania konkretnej technologii.

Jako przykład konkretnego zastosowania można podać aplikację udostępniającą poprzez sieć możliwość zagrania w szachy z komputerem lub innym człowiekiem. Schemat takiego rozwiązania zaprezentowano poniżej:



Rysunek 1: Propozycja schematu systemu