Projekt zaliczeniowy – gra PENTAGO

Sztuczna Inteligencja, drugi rok informatyki

Skład zespołu:
Gliwiński Jarosław Marek
Kruczyński Konrad Marek
Urbanek Paweł J.
Grupa dziekańska I5

Poznań, 9. marca 2009

Spis treści

1 Opis gry **3**
1.1 Krótka historia PENTAGO . . . 3
1.2 Opis zasad gry . . . 3
1.2.1 Budowa planszy . . . 3
1.2.2 Fazy i zasady gry . . . 3
1.2.3 Cel gry i jej zakończenie . . . 4
1.2.4 Przykładowa plansza . . . 4

2 Implementacja **4**
2.1 Reprezentacja stanu gry . . . 4
2.1.1 Model reprezentacji . . . 4
2.1.2 Przykładowa reprezentacja . . . 5
2.2 Generacja ruchów dopuszczalnych . . . 6
2.2.1 Realizacja ruchów dopuszczalnych . . . 6
2.2.2 Sposób implementacji poszczególnych typów ruchów . . . 6
2.2.3 Kolejność generowania następników . . . 6
2.2.4 Niedeterminizm mechanizmu generacji następników . . . 6
2.3 Metody przeszukiwania przestrzeni stanów gry . . . 7
2.3.1 Tablica transpozycji . . . 7
2.3.2 Tablica historii ruchów . . . 7
2.4 Funkcja oceny heurystycznej stanu gry . . . 8

3 Testy wydajności i porównania algorytmów **8**

4 Wnioski **11**

1 Opis gry

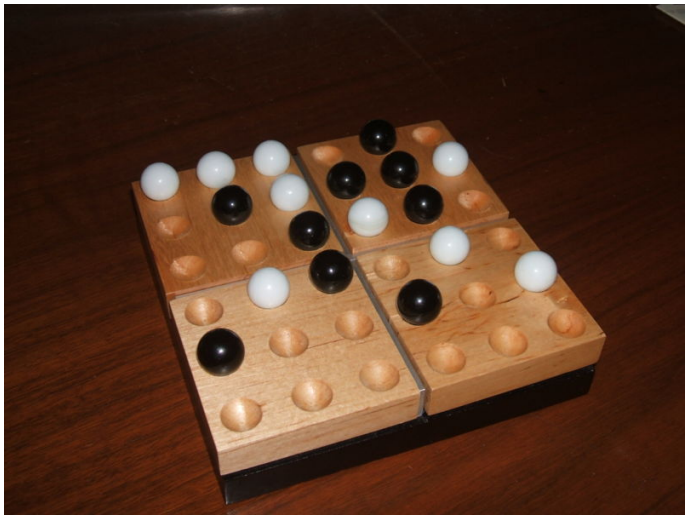
1.1 Krótka historia PENTAGO

Co łatwo zauważyć po zasadach panujących w grze, powstała ona poprzez modyfikację znanej od wieków gry w „kółko i krzyżyk" poprzez powiększenie planszy oraz, co najistotniejsze, możliwości obracania jej części.

1.2 Opis zasad gry

1.2.1 Budowa planszy

Plansza do PENTAGO składa się z 36 pól, przy czym jest ona kwadratem, więc jak łatwo się domyślić, na każdym boku mieści się 6 pól. Nadto jest ona symetrycznie podzielona na cztery równe części (segmenty). Części te w tradycyjnej (tj. drewnianej) wersji są niezależnymi kwadratami, co umożliwia obrót wybranego z nich. Poza tą rożnicą przynależność do konkretnej z nich nie wpływa w żaden sposób na znaczenie pola, tj. wszystkie one są równoważne. Poniższe zdjęcie przedstawia planszę oryginalnej wersji gry.



1.2.2 Fazy i zasady gry

PENTAGO przeznaczone jest dla dwóch graczy, których posunięcia następują kolejno po sobie. Po każdej „naszej" akcji przychodzi kolej na przeciwinika, tj. nie ma ruchu, który daje możliwość ponownego posunięcia. Ruch każdego gracza zazwyczaj składa się z dwóch części, natomiast w wyjątkowych przypadkach tylko z jednej. W pierwszej z nich dokonujemy ustawienia nowego pionka we własnym kolorze w dowolnym, dotąd niezajętym miejscu na planszy. Jeżeli już po takim ustawieniu uda się nam uzyskać jeden ze stanów wygrywających (o czym mowa będzie za chwilę), gra kończy się naszą

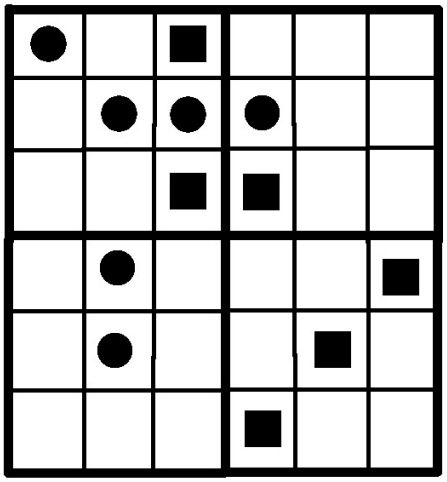
wygraną. W przeciwnym wypadku zmuszeni jesteśmy do wykonania jeszcze drugiej fazy naszego posunięcia, tj. obrotu wybranego (jeden z czterech) segmentu planszy w wybraną (lewoskrętnie bądź też prawoskrętnie) stronę. W tym momencie przychodzi kolej na naszego przeciwnika.

1.2.3 Cel gry i jej zakończenie

Celem gry jest uzyskanie na planszy takiego układu pionów, w którym w jednej linii (pionowo, poziomo lub też ukośnie) ułożonych mamy pięć pionów „naszego" koloru – wówczas następuje wygrana (oraz porażka przeciwnika). Analogiczne ułożenie pionów należących do przeciwnika kończy się naszą przegraną. Należy pamiętać, iż, zgodnie z poprzednim podpunktem, odpowiednie ułożenie możemy uzyskać przed lub po obrocie.

W grze istnieje również możliwość zremisowania – na dwa sposoby. W pierwszym z nich wybrane przez nas *coup de grâce* jest zarazem układem zawierającym wygraną przeciwnika, w drugim zaś następuje zapełnienie wszystkich 36 pól planszy bez osiągnięcia celu gry.

1.2.4 Przykładowa plansza



2 Implementacja

2.1 Reprezentacja stanu gry

2.1.1 Model reprezentacji

Ponieważ plansza jest macierzą pól o wymiarze 6×6, naturalną reprezentacją jest dwuwymiarowa tablica. Ponieważ każde pole może być w jednym z trzech stanów (niewypełnione, zajęte przez nas, zajęte przez naszego przeciwnika), jej elementy muszą

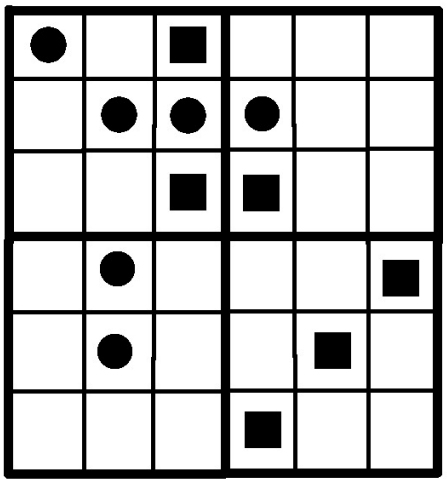
pozwalać na zapis przynajmniej trzech wartości. Klasycznym w wyborem w tym przypadku jest jakiś typ całkowity i na taki też się zdecydowaliśmy. Z uwagi na to, iż językiem wybranym do wykonania implementacji został `C++`, używamy standardowego typu `int`.

Warto dodać, iż z uwagi na łatwość wyboru przekazywania parametru (referencja, kopia) oraz swoiste „opakowania" spośród różnych możliwości reprezentacji tablicy w wyżej wymienionym języku wybraliśmy udostępnianą przez bibliotekę `STL`, tj. stan jest typu

```
vector<vector<int>>
```

Na koniec można zauważyć, iż taka reprezentacja cechuje się pełna izomorficznością ze stanami w tradycyjnej wersji gry, tj. nie spotykamy się ani z nadmiarem (dwa różne zapisy w pamięci dla identycznego układu na drewnianej planszy), ani też, co istotniejsze, z niedomiarem (gdzie dwa różne „drewniane" stany są identycznie odzwierciedlone w komputerze).

2.1.2 Przykładowa reprezentacja



```
1 0 2 0 0 0
0 1 1 1 0 0
0 0 2 2 0 0
0 1 0 0 0 2
0 1 0 0 2 0
0 0 0 2 0 0
```

2.2 Generacja ruchów dopuszczalnych

2.2.1 Realizacja ruchów dopuszczalnych

Wyznaczenie wszystkich ruchów dopuszczalnych w przypadku gry takiej jak PENTAGO jest zadaniem nietrudnym. Ponieważ każdy ruch wiąże się ze wstawieniem nowego piona, bazową ich liczbę określa liczba wolnych pól planszy. W celu ich wyznaczenia skonstruowano funkcję `policzWolne`. Wymienioną liczbę należy pomnożyć przez liczbę możliwych obrotów, tych jest 8, ponieważ wybieramy jeden z czterech segmentów do obrotu i możemy dokonać w jedną z dwóch stron. Nie jest to jednak ilość wszystkich dopuszczalnych ruchów, ponieważ zdarza się, iż dopuszczony zostanie ruch bez obrotu – finalny. Ponieważ brak obrotu nie wybiera konkretnego segmentu, zwiększa on ostatnio wymienioną liczbę dwukrotnie. Należy też mieć na uwadze, iż tego typu ruch jest zawsze ostatnim, kończącym grę.

Ponieważ funkcja wyznaczająca konkretny ruch bazuje na stanie, z którego owo posunięcie wykonujemy oraz numerze kolejnego takiego, toteż podpunkt traktujący o kolejności generacji ruchów dopełni opisu częściowo zawartego w podpunkcie niniejszym.

2.2.2 Sposób implementacji poszczególnych typów ruchów

Pierwotną ideą, jaka towarzyszyła temu zagadnieniu, była implementacja ruchu poprzez przekazanie nowego stanu gry. Trzeba jednak mieć na uwadze fakt, iż niektóre stany mogą być osiągnięte przez wiele różnych ruchów. Wobec tego dla zachowania np. pełnej historii gry należy podać parametry posunięcia, jakimi są współrzędne nowego piona oraz wybrany segment i kierunek obrotu (lub jego brak). Ponieważ jest to jedyny typ ruchu, opis jest już kompletny.

2.2.3 Kolejność generowania następników

Ponieważ każdy następnik jest (wzajemnie) jednoznacznie wyznaczony przez swój (kolejny) numer, toteż kolejność generowania stanów w taki sam sposób zależy tu od kolejności podawanych numerów. Na początek podana zostanie zatem wspomniana zależność poprzez omówienie algorytmu generacji stanu. We wstępie należy jeszcze dodać, iż numeracja następuje od zera.

Algorytm opiera się na serii dzieleń z resztą. Reszta pierwszego (przez 2) mówi o tym, czy próbujemy wykonać akcję ruchu bez obrotu. Jak wiadomo, taka dozwolona jest tylko w przypadku, gdy ruch ten jest finalnym, toteż fakt ten sprawdzany jest przed obliczeniami. Reszta kolejnego dzielenie (przez 4) decyduje o obracanym segmencie. Nadmiar z ostatniej operacji tego typu (ponownie przez 2) mówi o kierunku wykonywanego obrotu

2.2.4 Niedeterminizm mechanizmu generacji następników

W celu uniknięcia powtarzalności ruchów (w szczególności otwarć) związanych z deterministycznym przeglądaniem w określonej kolejności ruchów o tej samej wartości zastosowano prosty mechaniz niedeterministyczny. Przed wejściem w pętlę przeglądającą

następniki danego stanu kolejność elementów w tablicy jest permutowana prostym algorytmem zamiany par liczb. Następnie tablica tak spermutowanych pozycji jest używana do adresowania pośredniego kolejnych stanów w pętli je przeglądającej.

2.3 Metody przeszukiwania przestrzeni stanów gry

2.3.1 Tablica transpozycji

Rekord tablicy transpozycji zrealizowano za pomocą klasy o następującej deklaracji:

```
class tState
{
public:
vector< vector<int> > state;
int value;
int bound;
int depth;

//metody i operatory wycieto

};
```

State to oczywiście stan reprezentowany przez wektor dwuwymiarowy, jak wspomniano wcześniej, value przechowuje wartość uzyskaną dla danego stanu, bound rodzaj ograniczenia, zaś depth głębokość. Utrzymywane są dwie osobne tablice, dla każdej ze stron rozgrywki.

Sama tablica zrealizowana jest za pomocą kontenera zbioru z biblioteki standardowej.

W grze nie występują bicia ani żadne ruchy analogiczne, tak więc nie zachodziła potrzeba ich obsługi.

2.3.2 Tablica historii ruchów

Tablica historii ruchów jest wektorem o rozmiarze równym ilości możliwych do wykonania ruchów, tj 288. Ruch jest identyfikowany przez numer bezwzględny opisany wcześniej oraz ocenę ruchu. Wektor jest posortowany wg drugiego z tych parametrów.

Reguła modyfikacji zapamiętanej wartości jest następująca: na początku wszystkie ruchy mają ocenę 0, a w chwili kiedy alfa-beta wybierze jakiś ruch na danym, $k - tym$ poziomie drzewa, to ocena tego ruchu zwiększa się o współczynnik 2^k, gdzie za poziom zerowy przyjmujemy poziom, gdzie znajdują się liście.

Reguła jest bezkontekstowa. Uznaliśmy, że nie jest opłacalne branie pod uwagę większej ilości informacji, co spowalnia obsługe tablicy, zaś nie wpływa na wyniki działania algorytmu.

2.4 Funkcja oceny heurystycznej stanu gry

Funkcja jest zliczeniowa. Zliczanie odbywa się w wierszach, kolumnach oraz na głównych i „prawie głównych" przekątnych planszy. Wagowanie opiera się na różnicy nieliniowych transformacji maksymalnych rzędów pionów ułożonych przez graczy. Nieliniowa funkcja jest następująca:

```
int NL(int a)
{
int b;
switch(a)
{
case 3: b=4; break;
case 4: b=7; break;
case 5: b=13; break;
case 6: b=11; break;
default: b=a; break;
}
return b;
}
```

Jak widać teoretycznie funkcja zwróci wartości z zakresu (¡-13; 13), jednak w praktyce nie sposób osiągnąć sytuacji, gdy jeden gracz ma ułozony rząd 5 pionów a drugi nie ma nawet rzędu o długości 1. Wartości wag przekształcenia nieliniowego zostały dobrane iteracyjnie, tj poprzez proces analizy rozgrywek po kolejnych zmianach dobranej arbitralnie funkcji-warunku początkowego. Testy przeprowadzano zarówno w programie, jak i „na stole". Funkcja oceny kiepsko bierze pod uwagę sekwencję obrotu w lewo i w prawo tego samego segmentu w dwóch kolejnych ruchach. Człowiek łatwo może wykorzystać to, obracając segment w przeciwną stronę niż „chce" komputer, skutecznie blokując jego ruchy nawet przez kilka tur.

Dla podanego wcześnie przykładowego stanu gry, wartość funkcji wynosi $3 - 3 = 0$

3 Testy wydajności i porównania algorytmów

Dokonano porównania algorytmów:

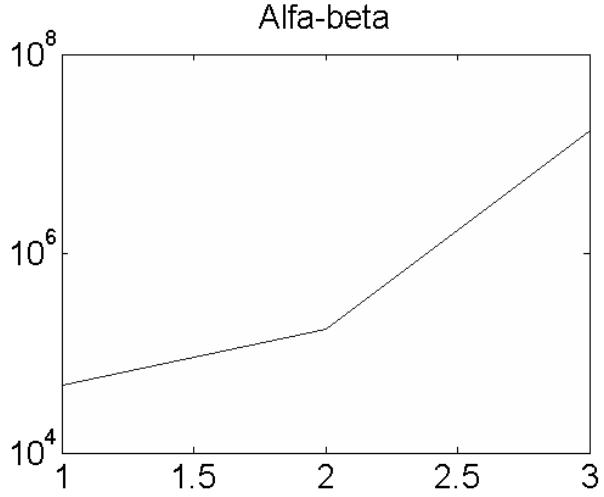
- „czystej" alfa-bety
- alfa-bety z tablicą transpozycji
- alfa-bety z tablicą historii ruchów

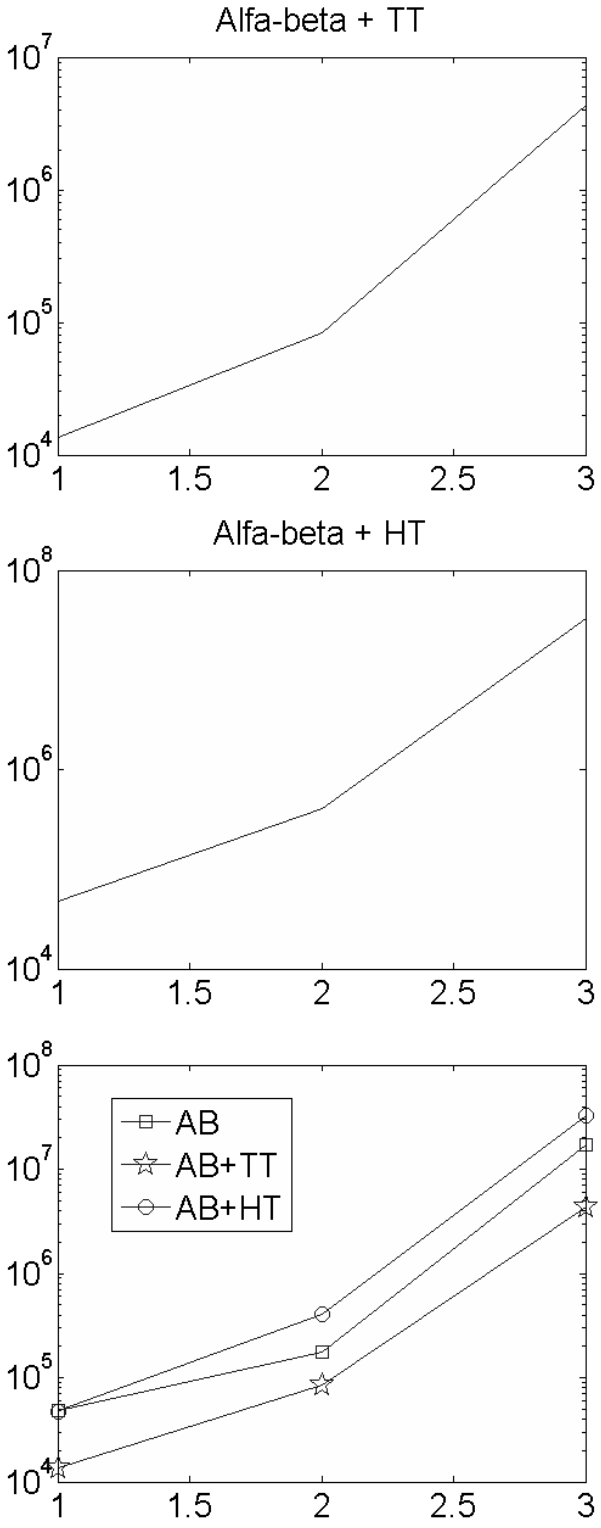
Porównanie odbywało się poprzez pomiar średniej liczy wierzchołków odwiedzonych w grafie we wszystkich ruchach od początku aż do zakończenia rozgrywki. Testy przeprowadzono dla trzech głębokości – od 1 do 3. Wyniki pomiarów przedstawiały się następująco:

```
algorytm średnia odch. std bezwzględne
-------------------------------------------------
Dla głębokości 1:

alfabeta        48274       13657
alfabetaTT      13540       2477
alfabetaHT      47595       7243
-------------------------------------------------
Dla głębokości 2:

alfabeta        175405      24884
alfabetaTT      84082       7021
alfabetaHT      402604      91230
-------------------------------------------------
Dla głębokości 3:

alfabeta        17178322
alfabetaTT      4344483
alfabetaHT      32659452
-------------------------------------------------
```

Dla głębokości 3 przeprowadzono po jednym pomiarze – ze względu na czas trwania rozgrywki.

Wykresy (w skali logarytmicznej, ze względu na różnice rzędu wielkości dla różnych głębokości) prezentują się następująco:




Alfa-beta + TT
10^7
10^6
10^5
10^4
1
1.5
2
2.5
3
Alfa-beta + HT
10^8
10^6
10^4
1
1.5
2
2.5
3
10^8
10^7
10^6
10^5
10^4
AB
AB+TT
AB+HT
1
1.5
2
2.5
3

4 Wnioski

Rozmiar przestrzeni stanów gry wynika z ilości możliwych stanów pola (3) oraz ilości pól (36) i przedstawia się prostym kombinatorycznym wzorem 3^36. Przestrzeń ta maleje wraz z grą, ponieważ zajmowane są kolejne pola – ustalane są zatem ich stany. Zgodnie z przewidywaniami najlepszą z badanych metod modyfikacji alfa-bety okazała się tablica transpozycji. Wynika to z faktu, iż łatwo natrafić w Pentago na powtarzające się podczas przeszukiwania stany. Zupełnie niepasującą do gry okazała się za to metoda opierająca się na historii ruchów – również zgodnie z przewidywaniami.

Człowiekowi względnie łatwo jest wygrać z komputerem, choć wymaga to pewnego obycia w grze. Wynika to z niedoskonałości funkcji stanu oraz złożonej, dwuetapowej natury ruchu.

W grze nie występują cykle, tak więc nie było konieczne podejmowanie żadnych kroków z tym związanych.