Sprawozdanie z projektu zaliczeniowego przedmiotu:

# Sztuczna Inteligencja

## Gra: Walls

Wydział Informatyki i Zarządzania
Kierunek: Informatyka
Marcin Domański (89737)
Michał Witczak (89842)
02.10.2010, Poznań

# Spis treści

**1) Opis gry ..... 3**

a) Historia Gry ..... 3

b) Zasady ..... 3

**2. Implementacja ..... 4**

a) O grze ..... 4

Technologie ..... 4

Interfejs ..... 4

Tryby rozgrywki ..... 6

b) Reprezentacja stanu gry ..... 7

Ruch ..... 7

Zapis do pliku ..... 8

c) Metody przeszukiwania przestrzeni stanów gry - algorytmy grające. ..... 8

O algorytmach ..... 8

Tablica przejść ..... 9

**3. Testy wydajności i porównania algorytmów ..... 9**

a) Stany gry do testów ..... 9

b) Tabela wyników ..... 12

c) Wykresy ..... 12

**4. Wnioski, uwagi i podsumowanie ..... 14**

# 1) Opis gry

## a) Historia Gry

Gra Walls jest raczej nieznana pod tym tytułem. Bardziej popularne terminy określają tę grę jako: Käsekästchen (w języku niemieckim), Dots and Boxes, Squares, Paddocks, Square-it, Dots and Dashes, Dots, Smart Dots, Dot Boxing (w języku angielskim) lub wreszcie kwadraty.

Pomysłodawcą gry był Édouard Lucas, który w 1889 zaproponował jej zasady. Gra jest najbardziej znana w Niemczech, gdzie w sześćdziesiątych i siedemdziesiątych dzieci grały w nią w wolnych chwilach w szkołach.

Prawdopodobnie pierwszą implementacją gry na komputer, była wydana w 1989 roku (przez niemiecką firmę 64'er-Markt & Technik) gra na konsole Commodore 64, o tytule Kasekastchen. Obecnie można znaleźć webowe implementacje tej, których zasady uwzględniają dodatkowy ruch po zdobyciu punktu (szczegóły w części poświęconej zasadom).

## b) Zasady

Gra rozgrywana jest na planszy 6x6. Pojedynczym elementem jest kwadrat, którego cztery boki można pomalować. Kwadraty są ułożone w ten, sposób, że krawędź pomiędzy dwoma kwadratami tworzy boki obu kwadratów. Gracze wykonują swoje ruchy naprzemiennie. Ruch polega na wybraniu odpowiedniej krawędzi i zaznaczeniu go kolorem. Zamalować można tylko krawędź, która nie była wcześniej zakolorowana. Gracz, który zamaluje ostatni, czwarty bok kwadratu otrzymuje punkt. Zakończenie gry następuje, gdy wszystkie krawędzie zostaną zamalowane. Oczywiście wygrywa gracz z największą ilością punktów.

Zgodnie z poleceniem zaimplementowaliśmy wersję gry z planszą 6x6 bez dodatkowego ruchu, w przypadku zdobycia punktu. W innych implementacjach gry zasada o dodatkowym ruchu po zdobyciu punktu była dość często stosowana. Można również spotkać wersje, w których za zdobycie danych kwadratów dostaje się różną ilość punktów.

# 2. Implementacja

## a) O grze

### Technologie

Gra została zaimplementowana w języku C# (technologia .NET) z wykorzystaniem środowiska Microsoft Visual Studio 2008, przy użyciu XNA Game Studio 3.1. Do uruchomienia gry wymagane są:

• Microsoft .NET Framework 3.5 Service Pack 1
• Microsoft XNA Framework Redistributable 3.1

Były już osoby, które oddawały projekty z wykorzystaniem tego środowiska, ale w razie czego linki z niezbędnymi wersjami instalacyjnymi załączymy do pliku ReadMe.txt.

### Interfejs

- **MENU**

Opis opcji menu:

**New Game** – przejście do nowej gry, z ustawieniami kolorów i graczy z Menu/Player Options
**Player Options** – Opcje dotyczące tego czy dany gracz sterowany jest przez komputer i jeśli tak to, jakim algorytmem, i do jakiej głębokości
**Load Game** – Odczytuje wcześniej zapisaną grę. (Zapis klawiszem „s” podczas rozgrywki).
**Help** – Krótkie objaśnienie zasad i sterowania.
**About** – O projekcie, autoach i realizacji.
**Exit** – bolesny powrót do rzeczywistości.

Sterowanie w menu:

**↑,↓** – służą do wyboru opcji w menu.

**ENTER** – służy do potwierdzenia wyboru w menu.

- **Player Options**

Osobno dla każdego gracza wybieramy czy jest to gracz komputerowy, czy człowiek oraz ustawiamy, jaki algorytm ma być użyty, w przypadku, gdy dany gracz to gracz komputerowy. Wybór głębokości algorytmów został ograniczony do poziomy 7.

- **Ekran Gry**

Ekran gry został podzielony na dwie części. W części lewej znajduje się plansza, w prawej log przebiegu gry, stan punktowy, oraz informacja o aktualnie grającym graczu.

Plansza została podzielona dwanaście wierszy i kolumn, które zostały opisane na brzegach planszy. Po planszy poruszamy się kursorem, który występuje w postaci pędzla z kolorem aktualnie grającego gracza. Gracz potwierdzając swój ruch maluje płot na swój kolor.

Zdobycie punktu sygnalizowane jest pojawieniem się kwiatka takiego jak na rysunku powyżej. Oczywiście aktualny wynik wyświetlany jest również na panelu po prawej. Koniec gry sygnalizowany jest pojawieniem się stosownej planszy informującej o wyniku rozgrywki.

Oprócz punktów na prawym panelu umieszczono również strzałkę informującą, który z graczy aktualnie wykonuje ruch wskazując na odpowiedni kolor oraz zapis ostatnio wykonanych ruchów w formie [nr wiersza, nr kolumny] w kolorze gracza, który wykonał dany ruch.

Sterowanie w grze:
**↑↓←→** – służą do poruszania pędzlem po planszy.
**ENTER** – służy do malowania danego fragmentu płotu wskazanego przez pędzel.
**„s”** – służy do zapisu stanu gry.
**ESC** – służy do opuszczenia gry; wyjście do menu.
**„b”** – służy do cofnięcia ruchu.

## Tryby rozgrywki

Dostępne są wszystkie tryby, czyli gracz komputerowy przeciwko graczowi komputerowemu, człowiek przeciwko człowiekowi oraz człowiek przeciwko graczowi komputerowemu. Aby ustawić odpowiedni tryb rozgrywki należy przejść do menu „player options”, które zostało opisane wcześniej

## b) Reprezentacja stanu gry

**Plansza**

```
{
{0,2,0,2,0,2,0,2,0,2,0,2,0},
{1,6,1,6,1,6,1,6,1,6,1,6,1},
{0,2,0,2,0,2,0,2,0,2,0,2,0},
{1,6,1,6,1,6,1,6,1,6,1,6,1},
{0,2,0,2,0,2,0,2,0,2,0,2,0},
{1,6,1,6,1,6,1,6,1,6,1,6,1},
{0,2,0,2,0,2,0,2,0,2,0,2,0},
{1,6,1,6,1,6,1,6,1,6,1,6,1},
{0,2,0,2,0,2,0,2,0,2,0,2,0},
{1,6,1,6,1,6,1,6,1,6,1,6,1},
{0,2,0,2,0,2,0,2,0,2,0,2,0},
{1,6,1,6,1,6,1,6,1,6,1,6,1},
{0,2,0,2,0,2,0,2,0,2,0,2,0},
};
```

Reprezentowana jest jako dwumiarowa tablica 13x13. Powyższa przykładowa plansza to plansza występująca na początku rozgrywki.

**Stan graczy**

Oprócz planszy przechowywane są również informacje o ilości punktów zgromadzonych przez graczy oraz o tym, który gracz aktualnie wykonuje ruch.

### Ruch

W celu wykonania ruchu, algorytmy mają do dyspozycji kilka metod, którymi mogą „badać” aktualny stan:

- Metoda *find_succ,* która wyszukuje dozwolone ruchy (w tej grze to po prostu wszystkie niepomalowane płoty). Kolejność ich generowania jest od lewego górnego rogu, schodząc w dół a następnie przesuwając się w prawo. Jeśli funkcja ta jest wywoływana w korzeniu, następuje dodatkowo przemieszanie wyników, co pozwala komputerowi grać w sposób losowy, a nie schematyczny. Zatem ostateczny wynik procedury to losowa (pseudolosowa) kolejność następników. Mieszanie następuje tylko w korzeniu w celu zaoszczędzenia czasu (mieszanie na dalszych poziomach nie ma sensu). Ponadto ilość wykonanych zamian (przetasowań) zależy również od wybranej maksymalnej głębokości przeszukiwania. Im więcej ruchów na przód będzie przewidywał algorytm, tym mniej losowo będzie grał. Jest to także sposób na zaoszczędzenie czasu.
- Metoda *eval,* która ocenia dany stan – w naszym przypadku zlicza kwiaty, czyli podaje aktualny wynik. Nie ma, zatem wag, preferencji określonych pól, układów. Powoduje to tak zwany efekt horyzontu, ale gra Walls ma taką specyfikacje, że często tworzą się długie „korytarze”, w których następuje naprzemienne zdobywanie punktów. Lepsza funkcja eval byłaby, zatem po prostu wykonaniem algorytmu na większą głębokość. Zwracane wartości są z przedziału (-36, +36) ponieważ tylko tyle kwiatów może na planszy wyrosnąć. Komputer chętnie wykorzystuje możliwość zdobycia punktu głównie wówczas, gdy możliwości zdobycia punktów jest nieparzysta liczba. Gdy zaś kwadratów z niepomalowanym tylko jednym płotem jest parzysta liczba, pola te nie są

traktowane priorytetowo. Tylko dla algorytmu działającego na głębokość 1, zdobycie punktu jest nadrzędne.

- Metoda sprawdzająca czy podany stan jest stanem kończącym grę, po prostu sprawdza czy jest jeszcze doniczka, w której nie wyrósł żaden kwiat.

Gdy algorytm znajdzie najodpowiedniejszy ruch, zwraca go w postaci pary liczb, określających współrzędne płotu, który ma zostać pomalowany.

### Zapis do pliku

Zapisywany do pliku jest jedynie stan planszy, punkty, oraz gracz, który aktualnie ma wykonać ruch. Pozwala to na zmienianie w trakcie gry algorytmu, jakim posługiwać będzie się komputer. Wystarczy zapisać daną grę, wyjść do menu (klawiszem ESC) ustawić algorytmy w „opcjach" a następnie wczytać grę z menu „Load Game". Dzięki temu można również użyć opcji gry Człowiek vs Człowiek jako edytora stanu. Malujemy płoty w celu doprowadzenia do odpowiedniego stanu, a następnie zapisujemy grę. Następnie wychodzimy do menu, ustawiamy odpowiednie algorytmy, a następnie ładujemy grę.

## c) Metody przeszukiwania przestrzeni stanów gry - algorytmy grające.

### O algorytmach

Zaimplementowano w sumie trzy algorytmy. Algorytm Min-Max, algorytm Alpha-Beta w wersji zwykłej oraz algorytm Alpha-Beta Fail-Soft z tablicą przejść. Możliwy jest też wybór głębokości przeszukiwanych stanów dla wszystkich trzech algorytmów.

W tablicy transpozycji przechowywany jest:

- identyfikator stanu,
- głębokość na jakiej znaleziono wartość ruchu,
- wartość oceny heurystycznej i jej typ,
- parę, która jest współrzędnymi najlepszego ruchu dla danego stanu.

Zdecydowaliśmy się na pamiętanie tylko jednej pary, a nie wszystkich najlepszych ruchów dla danego stanu z dwóch powodów. Po pierwsze i najważniejsze to ograniczenia pamięciowe. W przypadku przechowywania wielkich list, zdarzało się nam przekroczyć ilość pamięci RAM, zatem zdecydowaliśmy się na wiele optymalizacji i to była jedna z nich. Drugim czynnikiem była nieznaczna różnica w sposobie gry, ponieważ prawdopodobieństwo że człowiek będzie grał dokładnie tak samo jest niezwykle niskie, zatem ryzyko zaobserwowania schematyczności w poczynaniach komputera jest nieznaczne. W przypadku gry komputer vs komputer ryzyko to jest wyższe, jednak jak udało nam się zaobserwować w testach, że ruchy komputera nadal posiadają pewien niedeterminizm. Trzecim czynnikiem jest i tak duża złożoność obliczeniowa zastosowanych algorytmów, wiec nie chcieliśmy jej jeszcze bardziej pogarszać.

### Tablica przejść

Wykorzystanie tablicy przejść „za pierwszym razem” jest nie duże. Gra ma taką specyfikę, że nie można drugi raz w trakcie jednej rozgrywki osiągnąć tego samego stanu. Raz pomalowany płot już taki pozostaje do końca gry. Zatem aby efektywniej korzystać z tablicy, nie jest ona usuwana wraz z końcem gry, ale można z niej korzystać w ewentualnym rewanżu. Co więcej, jest ona zapisywana do osobnego pliku podczas zapisywania całej gry. Dzięki temu można łatwo uzyskać dokładnie taki sam stan programu po wczytaniu gry z menu „Load Game”. Tablica jest ponadto zapisywana z użyciem kompresji GZip.

Ponadto wyliczenie stanów na znacznych głębokościach pochłania dużą ilość czasu, a zatem ewentualne czyszczenie tablicy, czy też ponowne wyliczanie tego samego, wydaje się z punktu widzenia zwykłego działania gry kompletnie bez sensu.

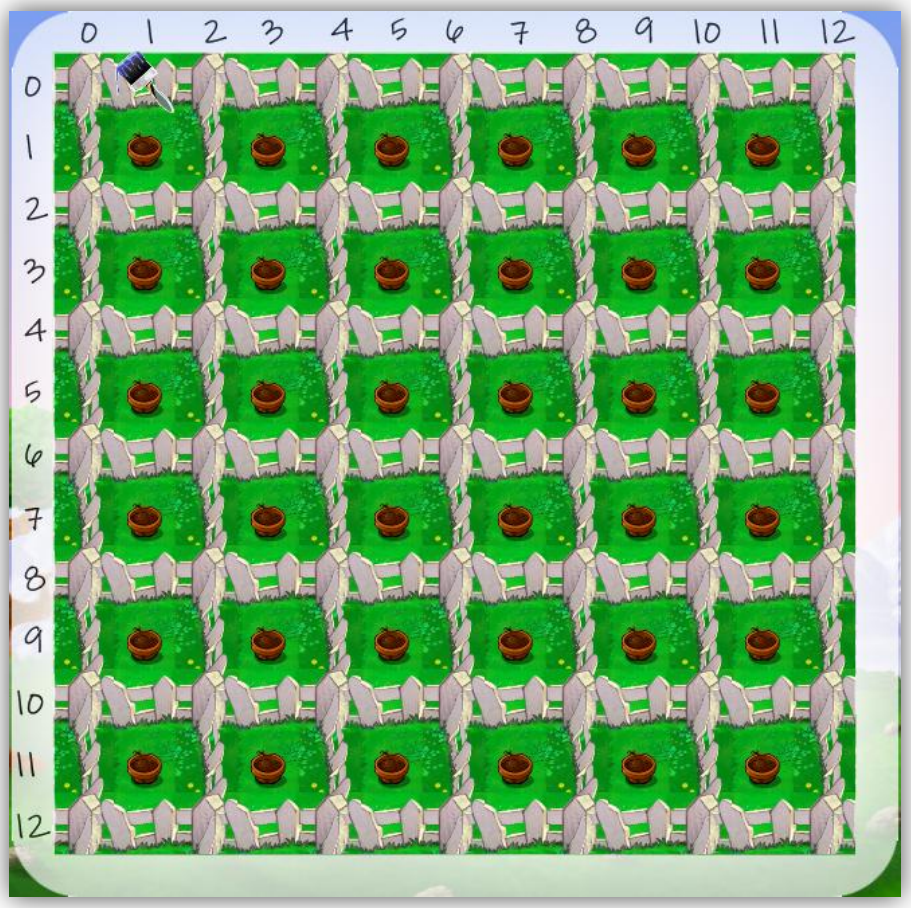
Jak wspomniano wyżej, raz pomalowany płot już takim pozostanie do końca gry, zatem niemożliwe są cykle.Nie ma, zatem potrzeby ich wykrywania.

Aby gra w Walls była przyjemnością, a nie ciągłym oczekiwaniem na ruch komputera, wprowadzono odcięcia czasowe. Gdy przekroczony zostanie czas przeznaczony na obliczenia, algorytm zwróci znaleziony dotychczas najlepszy ruch. Może to spowodować nie najlepsze zagranie, ale dzięki temu najlepsze chwile naszego życia nie miną nam na wpatrywaniu się w plansze, ale na planowaniu własnej strategii zdobycia jak największej liczby kwiatów.

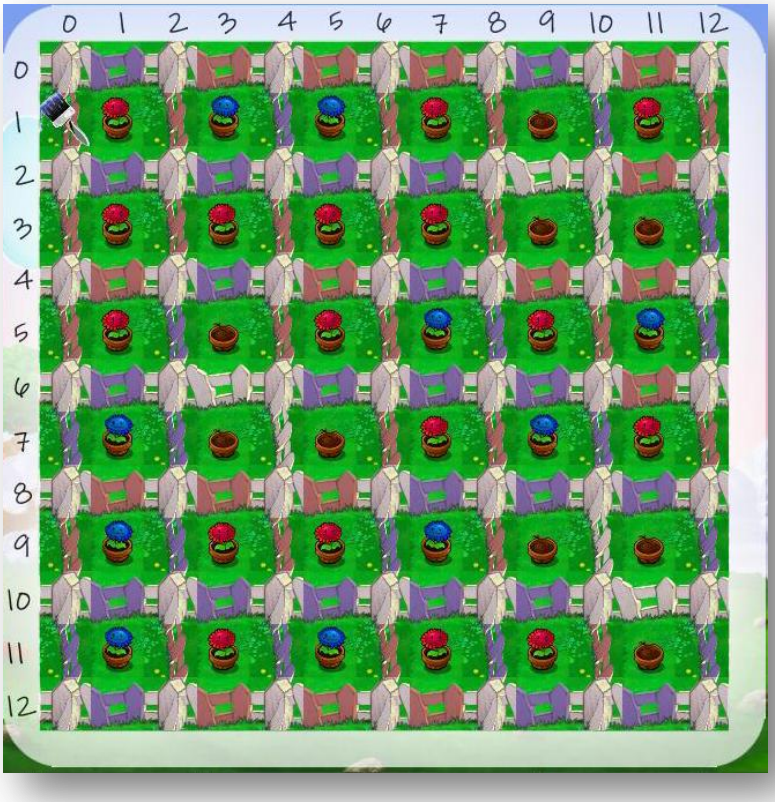
# 3. Testy wydajności i porównania algorytmów

## a) Stany gry do testów

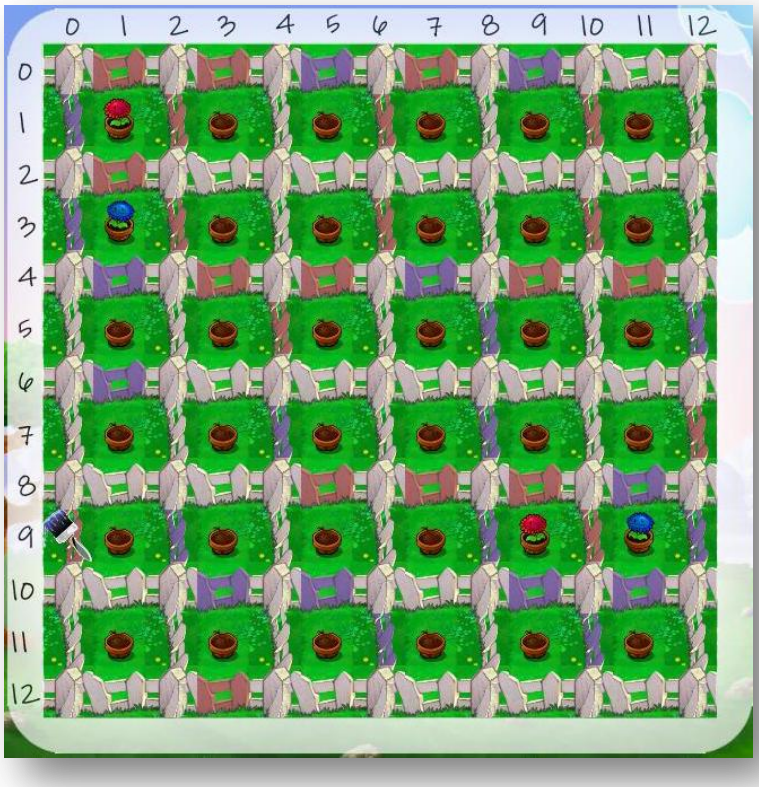
Badania przeprowadzono dla 3 stanów gry. Początkowego, środkowego i bliskiego terminalnemu.




**Stan Początkowy**




**Stan Środkowy**



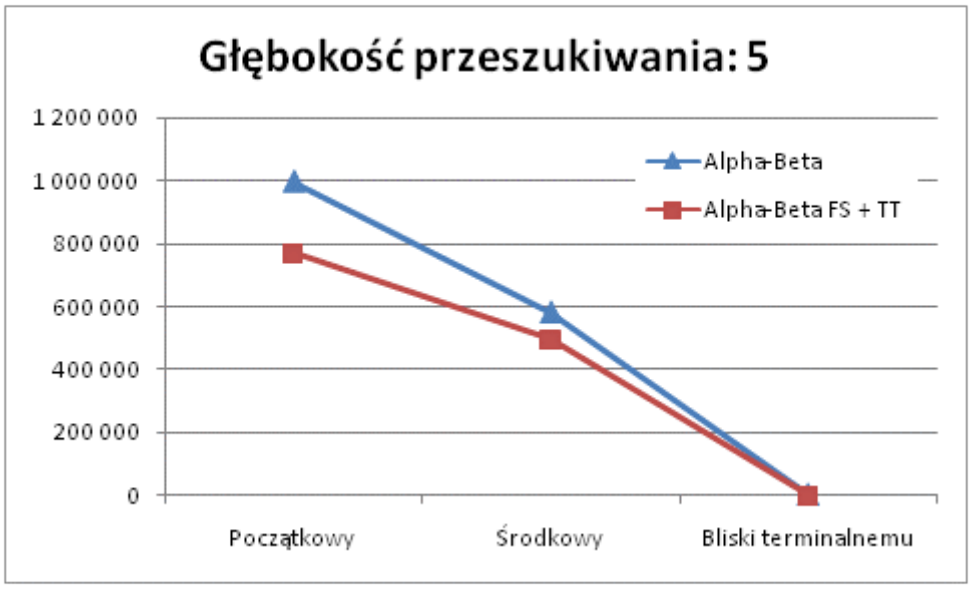

**Stan Końcowy**

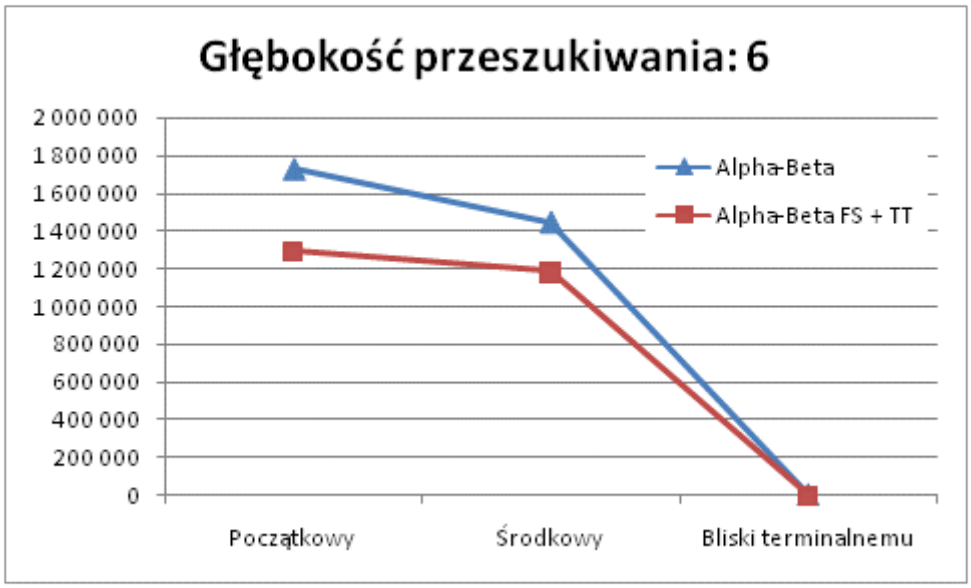
## b) Tabela wyników

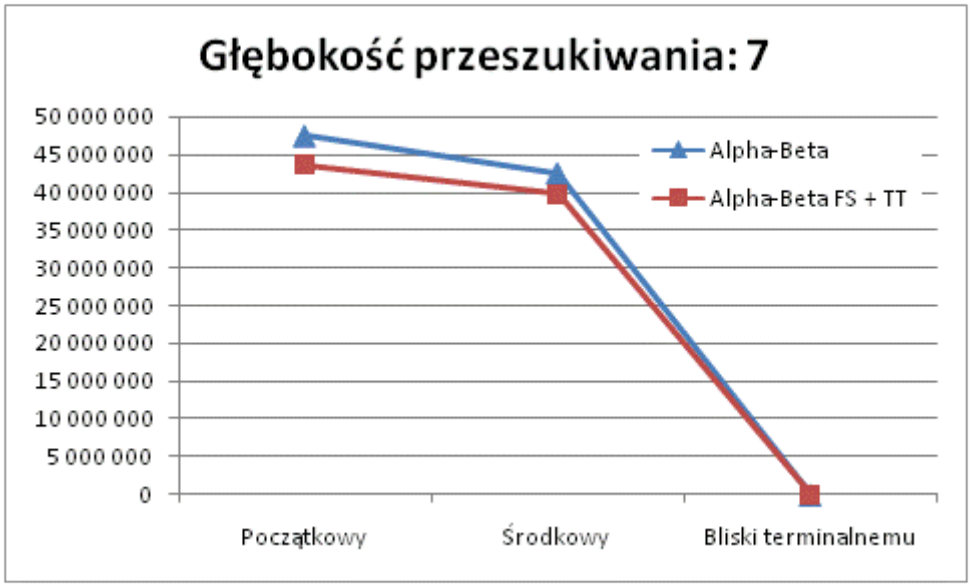
| Stan | αβ | αβFS + TT | Wykorzystanie TT |
|---|---|---|---|
| Głębokość przeszukiwania: 5 | | | |
| Początkowy | 995 043 | 769 976 | 29 382 |
| Środkowy | 579 859 | 498 315 | 15 034 |
| Bliski terminalnemu: | 347 | 369 | 37 |
| Głębokość przeszukiwania: 6 | | | |
| Początkowy | 1 732 167 | 1 293 857 | 63 230 |
| Środkowy | 1 449 845 | 1 183 494 | 51 531 |
| Bliski terminalnemu: | 481 | 441 | 86 |
| Głębokość przeszukiwania: 7 | | | |
| Początkowy | 47 538 635 | 43 492 918 | 423959 |
| Środkowy | 42 580 157 | 39 653 021 | 324 852 |
| Bliski terminalnemu: | 503 | 470 | 101 |

Wartości podane w tabeli są średnią z kilku pomiarów (dla stanu bliskiego terminalnemu z 15, dla algorytmu αβFS + TT dla stanów początkowego i środkowego z, a dla reszty z 5 pomiarów)

## c) Wykresy




Głębokość przeszukiwania: 6
2 000 000
1 800 000
1 600 000
1 400 000
1 200 000
1 000 000
800 000
600 000
400 000
200 000
0
Alpha-Beta
Alpha-Beta FS + TT
Początkowy
Środkowy
Bliski terminalnemu


Głębokość przeszukiwania: 7
50 000 000
45 000 000
40 000 000
35 000 000
30 000 000
25 000 000
20 000 000
15 000 000
10 000 000
5 000 000
0
Alpha-Beta
Alpha-Beta FS + TT
Początkowy
Środkowy
Bliski terminalnemu

## 4. Wnioski, uwagi i podsumowanie

Przeszukiwana przestrzeń stanów to DAG. Rozmiar pełnego grafu maleje wraz z postępem gry, od około 84! do zera (gra kończy się, gdy wszystkie płoty są pomalowane). Obszar przeszukiwania dla głębokości np. $n = 3$ maleje od $84 * 83 * 82 \approx 570\,000$. Obszar przeszukiwania był najmniejszy przy użyciu algorytmu Alpha-Beta Fail-Soft z tablicą przejść. Należy zwrócić uwagę na fakt, iż badania przeprowadzono z „czystą" tablicą transpozycji. W przypadku, gdy wczytamy gotową tablice, przestrzeń przeszukiwania może zostać ograniczona do 1 stanu, odczytanego po prostu bezpośrednio z tablicy. Dzieje się tak często w stanach bliskich terminalnemu. W pamięci komputera rozmiar przestrzeni stanów może wynosić kilkaset mega, a nawet gigabajt. Warto zwrócić uwagę, że gra nie posiada cykli.

Jednak nawet w przypadku gry przeciw algorytmowi przeszukującemu przestrzeń stanów na bardzo dużą głębokość, wygrać z komputerem jest bardzo prosto. Wystarczy grać „czerwonym graczem" tzn. nie rozpoczynać gry. Z obserwacji (i osobistej strategii) wynika, że gracz drugi ma znaczną przewagę. Co więcej, wydaje się, że gracz pierwszy przy założeniu, że obaj gracze grają najlepiej jak się da – nie jest w stanie nawet zremisować. Nie będziemy tego jednak udowodniać, gdyż nie to jest tematem projektu.

Jeśli chodzi o funkcje oceny heurystycznej, naszym zdaniem nie możliwe jest jej polepszenie. Jedynym polepszeniem jest przeszukiwanie na większą głębokość.

Największy problem implementacyjny sprawił fakt, że jeden z programistów nie uprzedzając o tym drugiego zmienił tablice reprezentacji stanu planszy. Działająca wcześniej Alpha-Beta nagle zaczęła „głupieć". Dojście, co jest tego przyczyną wymagało wielu godzin.