Дніпровський ліцей інформаційних технологій

при Дніпровському національному університеті

імені Олеся Гончара

Курсова робота

на тему:

**Шахові королеви**

**у прямокутному паралелепіпеді**

Виконавець:

ліцеїст 10 - Ф - 1 класу

Москаленко Сергій

Науковий керівник:

Ентін Й. А

Дніпро

2018

ЗМІСТ

ВСТУП……………………………………………………………………………………3

Задача, мета, актуальність, оригінальність роботи…………………………………….3

ТЕОРЕТИЧНА ЧАСТИНА……………………………………………………………...4

Характеристика двовимірної задачі…………………………………………………….4

Логіка вирішення задачі…………………………………………………………………5

Перші ферзі на полі…………………………………………………………………..….5

Скорочення варіантів…………………………………………………………………………………………………..6

ПРАКТИЧНА ЧАСТИНА……………………………………………………………….6

Основна форма…………………………………………………………………………...6

ВИСНОВКИ……………………………………………………………………………...8

ДЖЕРЕЛА………………………………………………………………………………..9

ДОДАТКИ………………………………………………………………………………10

**ВСТУП**

**Задача**

Є прямокутний паралелепіпед розмірами A\*А\*В, де В – висота його, А є Z, В є Z.

1) Визначити найбільшу кількість шахових королев, які треба розставити по його кубічних комірках 1Х1Х1, щоб королеви не загрожували одна одній. Королева тримає під боєм усі комірки по одній вертикалі та обом горизонталям, де вона знаходиться, а також по діагоналях (плоских та просторових, що утворюють кут 45° з плоскими), які перетинають її комірку.

2) Визначити найменшу кількість шахових королев, які треба розставити по його кубічних комірках 1Х1Х1, щоб усі комірки були під боєм королев.

**Мета роботи**

Розв’язання класичної задачі теорії алгоритмів для тривимірного варіанту.

**Актуальність**

Визначення найменшої кількості точок спостереження, при яких усе поле зору – під контролем.

**ТЕОРЕТИЧНА ЧАСТИНА**

**Перехід від двовимірного простору до тривимірного**

**Хід королеви**

В класичних шахах (на площині) ферзь ходить вперед, назад, вправо, вліво, та по діагоналях. В нашому тривимірному випадку додаються ходи вгору, вниз та діагоналі вглибину.

Далі на рис.1 показано всі можливі напрямки рухів одного ферзя у 3D полі. (куби із синім контуром – це королеви, а з червоним – її ходи). Всього виходить 27 напрямків.

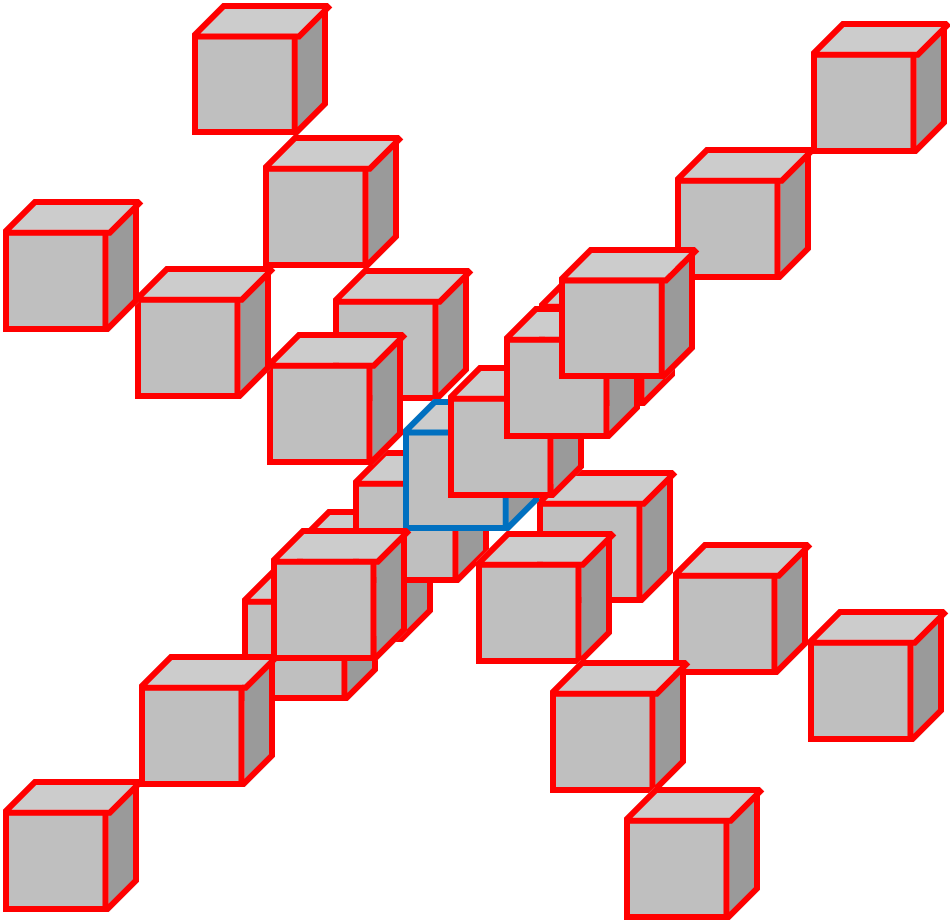
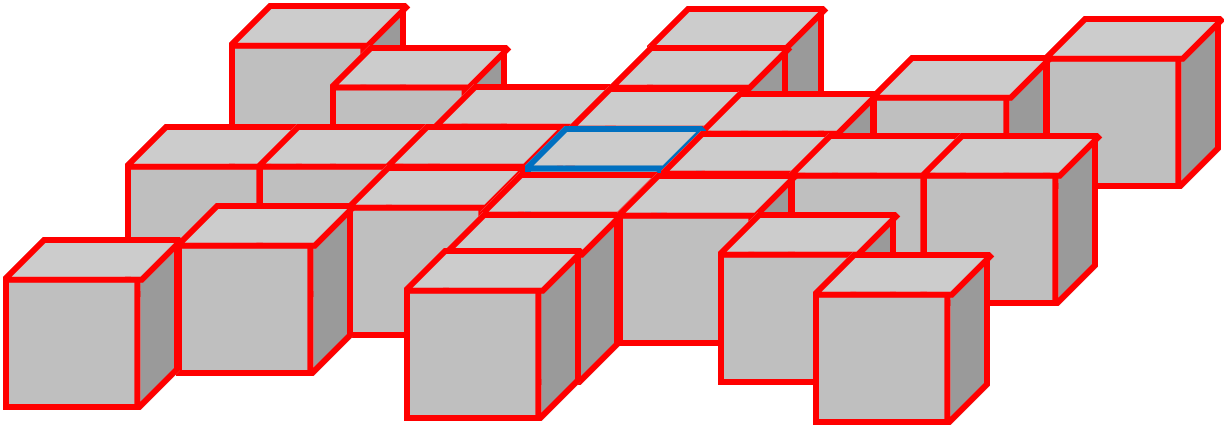
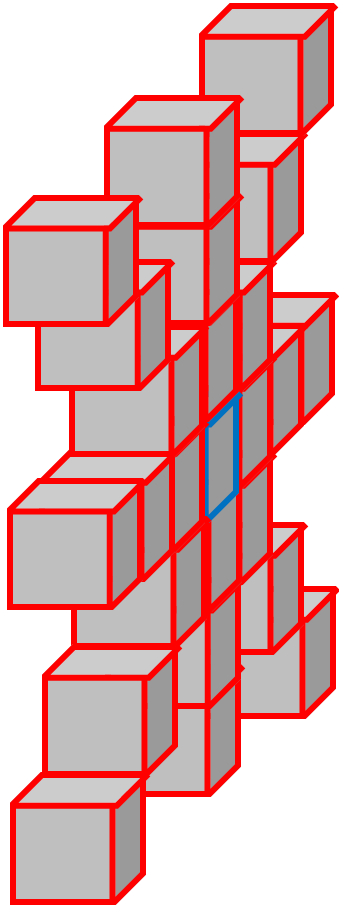
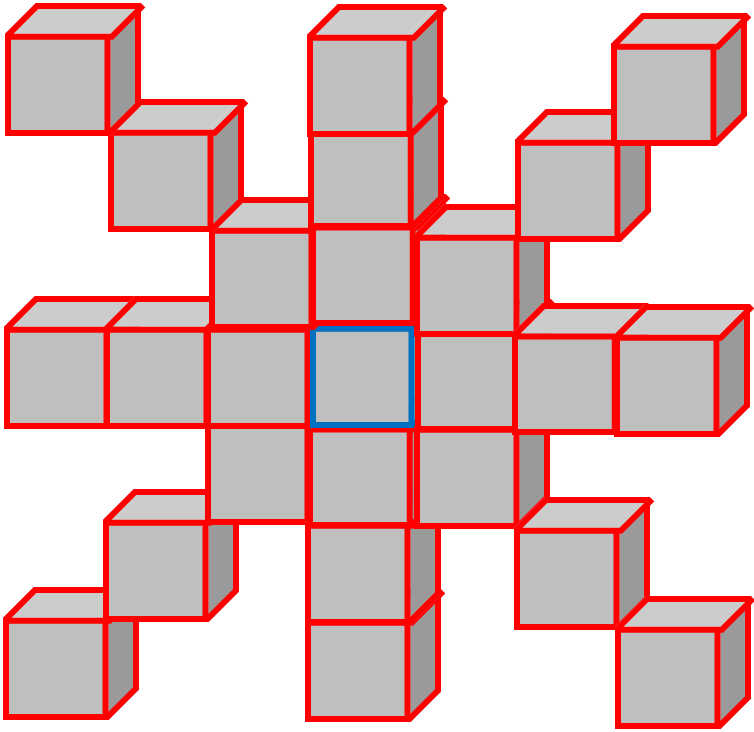


Рис.1 Ходи ферзя

**Логіка вирішення задачі**

Для вирішення задачі використовується алгоритм «backtrack» або метод повного перебору. Вacktrack – це пошук варіантів з поверненням.

Суть алгоритму – в наступному:

1. Ставимо першу королеву на будь-яку клітину дошки.
2. Програма запам’ятовує координати вільних комірок.
3. Додавання на дошку наступного ферзя полягає у переборі вільних комірок – доти, доки знайдеться комірка, яка не знаходиться під боєм ферзів, розставлених на дошці раніше. Таким чином, маємо створити підпрограму, яка викликає сама себе декілька разів (рекурсія).

Оригінальність роботи у тому, що замість традиційної двовимрної задачі розглядається тривимірну.

**Перші ферзі на полі**

Всі розстановки ферзів залежать від розтташування першої ферзі, але щоб не виконувати забагато дій можна скоротити кількість комірок, куди буде ставитися перша ферзь. Вдзеркалюємо поле відносно його центру по осі ZX(ділимо поле на передню та задню частину), ZY(ділимо поле на праву і ліву частину), XY(ділимо поле на нижню і верхню частину). Тому вже маємо, що достатньо тілько 1/8 всього поля, щоб перевірити усі варіанти розстановки ферзів.

**Скорочення варіантів**

Щоб при перегляді всіх варіантів не було однакових, або взаємозаміняємих треба порівнювати варіанти між собою при різних кутах нахилу відносно осі ХУ. Кутів оберту буде всього чотири: 0, 90, 180 і 270.(додаток 2).

**ПРАКТИЧНА ЧАСТИНА**

**Основна форма**

Програма складається з восьми форм:

1. Головне меню
2. Налаштування форми
3. Розстановка королев
4. Додаткова анімація
5. Web – сайт
6. Відео
7. Структура проекту (пояснення)

З форми «Головне меню» можна перейти на будь-яку іншу форму.

На формі «Структура проекту» точніше розповідається про можливості переходу з однієї форми на іншу.

Стрілочки з двома кінцями вказують на те, що з кожної форми можна перейти на іншу.

Стрілочки з одним кінцем вказують на те, що тільки з однієї з цих форм можна перейти на іншу, а повернутися назад не можна.

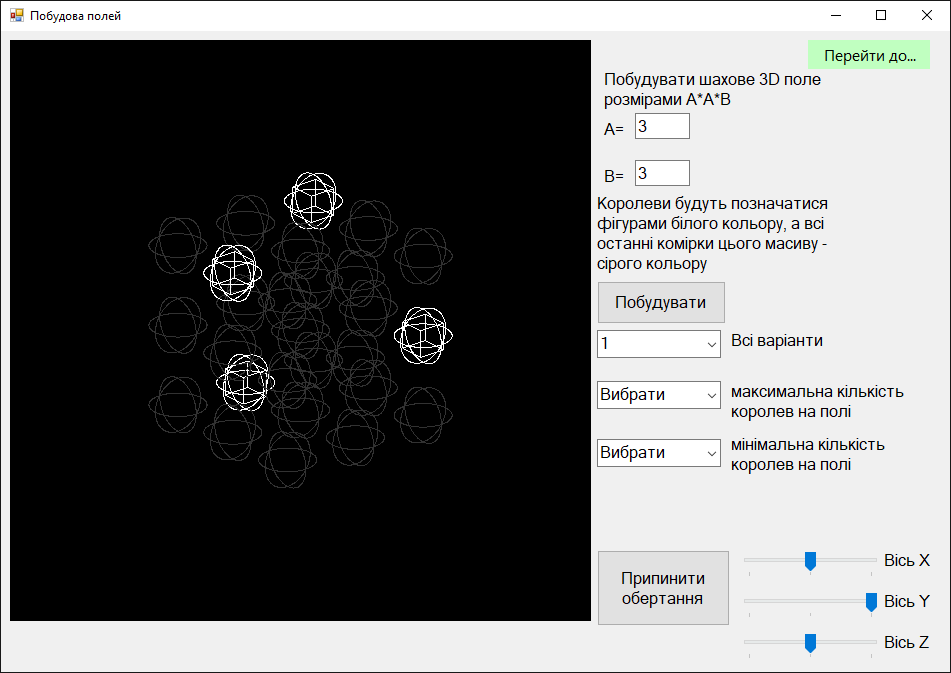
Основна задача вирішується на формі «Розстановка королев» (рис. 2). Щоб побудувати поле, треба вибрати розміри поля (бажано невеликі) і натиснути на кнопку «Побудувати». Далі треба вибрати один із елементів «comboBox». Тоді відобразиться поле, яке можна обертати за будь-яким напрямом. Королеви позначаються кубиками у шарах білого кольору. Поле малюється за допомогою бібліотеки для створення тривимірних зображень OpenGL.

Рис.2 Робочий проект

**ВИСНОВКИ**

Виконуючи цю роботу, я удосконалив свої навички програмування у мові С#, а саме: я навчився користуватися бібліотекою OpenGL, будувати масиви масивів, навчився працювати з формами та переходити з однієї на іншу, використовувати рекурсивні функції.

Задача про шахові королеви виявилася дуже цікавою. Вона допомогла мені розвинути просторове мислення. Мені вдалося реалізувати всі поставлені задачі, програма працює коректно і виводить правильний результат.

Для ускладнення задачі в майбутньому можна додати в 3D поле інші шахові фігури або їх комбінації, розставляти їх так, щоб одна одній не загрожувала. Також я пропоную додати до цієї програми ще одну класичну задачу з шахами про ходи коня, де потрібно побудувати алгоритм послідовності проходження кожної комірки 3D поля конем.

**ДЖЕРЕЛА ІНФОРМАЦІЇ**

1. Окулов С.М. Программирование в алгоритмах. -М.: БИНОМ – Лаборатория Базовых Знаний. – 2007, 384 с.
2. Вирт Н. Алгоритмы и структуры данных. М.: Прогресс. – 1989, 324 с.
3. Algolist.manual.ru

**ДОДАТКИ**

**Додаток 1.**

**Код для ходів королеви.**

//цикл для того, щоб помітити комірки, в яких не може стояти ферзь справа від вже існуючого ферзя

for (i1 = i + 1; i1 < n; i1++)

a[i1, j, u] = 1;

//цикл для того, щоб помітити комірки, в яких не може стояти ферзь ззаду від вже існуючого ферзя

for (j1 = j + 1; j1 < n; j1++)

a[i, j1, u] = 1;

//цикл для того, щоб помітити комірки, в яких не може стояти ферзь зліва від вже існуючого ферзя

for (i1 = i - 1; i1 > -1; i1--)

a[i1, j, u] = 1;

//цикл для того, щоб помітити комірки, в яких не може стояти ферзь попереду від вже існуючого ферзя

for (j1 = j - 1; j1 > -1; j1--)

a[i, j1, u] = 1;

//цикл для того, щоб помітити комірки, в яких не може стояти ферзь по діагоналі (направо + назад) від вже існуючого ферзя

i1 = i;

j1 = j;

while ((j1 + 1 < n + 1) && (i1 + 1 < n + 1))

{

a[i1, j1, u] = 1;

i1++;

j1++;

}

//цикл для того, щоб помітити комірки, в яких не може стояти ферзь по діагоналі (наліво + вперед) від вже існуючого ферзя

i1 = i;

j1 = j;

while ((i1 - 1 > -2) && (j1 - 1 > -2))

{

a[i1, j1, u] = 1;

i1--;

j1--;

}

//цикл для того, щоб помітити комірки, в яких не може стояти ферзь по діагоналі (направо + вперед) від вже існуючого ферзя

i1 = i;

j1 = j;

while ((i1 + 1 < n + 1) && (j1 - 1 > -2))

{

a[i1, j1, u] = 1;

i1++;

j1--;

}

//цикл для того, щоб помітити комірки, в яких не може стояти ферзь по діагоналі (наліво + назад) від вже існуючого ферзя

i1 = i;

j1 = j;

while ((i1 - 1 > -2) && (j1 + 1 < n + 1))

{

a[i1, j1, u] = 1;

i1--;

j1++;

}

//цикл для того, щоб помітити комірки, в яких не може стояти ферзь в всіх залишившихся напрямів вверх

i1 = i;

j1 = j;

for (u1 = u - 1; u1 > -1; u1--)

{

z++;

a[i, j, u1] = 1;

if (((j - z) > -1) && ((i + z) < n))

a[i + z, j - z, u1] = 1;

if (((j + z) < n) && ((i - z) > -1))

a[i - z, j + z, u1] = 1;

if (((j - z) > -1) && ((i - z) > -1))

a[i - z, j - z, u1] = 1;

if (((j + z) < n) && ((i + z) < n))

a[i + z, j + z, u1] = 1;

if ((j - z) > -1)

a[i, j - z, u1] = 1;

if ((i - z) > -1)

a[i - z, j, u1] = 1;

if ((i + z) < n)

a[i + z, j, u1] = 1;

if ((j + z) < n)

a[i, j + z, u1] = 1;

}

//цикл для того, щоб помітити комірки, в яких не може стояти ферзь в всіх залишившихся напрямів вниз

z = 0;

for (u1 = u + 1; u1 < x; u1++)

{

z++;

a[i, j, u1] = 1;

if (((j - z) > -1) && ((i + z) < n))

a[i + z, j - z, u1] = 1;

if (((j + z) < n) && ((i - z) > -1))

a[i - z, j + z, u1] = 1;

if (((j - z) > -1) && ((i - z) > -1))

a[i - z, j - z, u1] = 1;

if (((j + z) < n) && ((i + z) < n))

a[i + z, j + z, u1] = 1;

if ((j - z) > -1)

a[i, j - z, u1] = 1;

if ((i - z) > -1)

a[i - z, j, u1] = 1;

if ((i + z) < n)

a[i + z, j, u1] = 1;

if ((j + z) < n)

a[i, j + z, u1] = 1;

}

**Додаток 2**

**Функція скорочення варіантів:**

// variant – це загальна довжина масиву масивів zi[][,,]

//all\_var[][,,] – це масив у якому зберігаються абсолютно всі варіанти розташування ферзів

bool is\_same(int variant,int zi)

{

bool l = false;

for (int y1 = 0; y1 < variant; y1++)

for (int u = 0; u < x; u++)

{

for (int j = 0; j < n; j++)

{

for (int i = j; i < n; i++)

{

if (all\_var[y1][i, j, u] != z[zi][i, j, u])

{

l = false;

break;

}

else

l = true;

}

}

}

return l;

}