# Техническое задание

Разработка алгоритма для анализа и рационализации биматричных игр:

Поиск равновесий по Нэшу и оптимумов по Парето в чистых стратегиях.

# Уточнённое техническое задание

По договорённости с заказчиком был улажен следующий вопрос: нужно ли рассматривать ситуации, когда для определённого игрока в определённой ситуации будет одинаково выгодно воспользоваться несколькими своими стратегиями. Было решено, что такие ситуации должны входить в рассмотрение задачи.

# Решение

1. Для удобства пользователя было разработано меню: программа при нажатии определённого символа выполняет заданные действия. Если пользователь введёт символ, не используемый в данном меню, программа выведет ошибку и предложит ввести символ снова. Реализация меню представлена на рисунке 1.

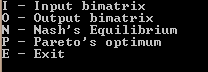


Рисунок 1. Реализация меню в программе

* Символы ‘i’, ‘I’ запускают ввод данных, а это: ввод размерности матрицы и весов каждого игрока в каждой ситуации.
* Символы ‘o’, ‘O’ запускают вывод матрицы игры. Если пользователь захочет сделать вывод матрицы до её ввода, будет выведена ошибка о некорректных данных, далее снова будет выведено меню.
* Символы ‘n’, ‘N’ запускают поиск равновесия по Нэшу. Если пользователь захочет сделать поиск до ввода матрицы, будет выведена ошибка о некорректных данных, далее снова будет выведено меню.
* Символы ‘p’, ‘P’ запускают поиск оптимума по Парето. Если пользователь захочет сделать поиск до ввода матрицы, будет выведена ошибка о некорректных данных, далее снова будет выведено меню.
* Символы ‘e’, ‘E’ осуществляют выход и программы.

1. Для решения поставленной задачи в коде была создана следующая структура (рисунок 2) – структура Bi, состоящая из двух полей типа int: per\_A и per\_B, – из которой впоследствии создаётся матрица. Поля структуры хранят веса игроков A и B соответственно в каждой из ситуаций игры (i, j).

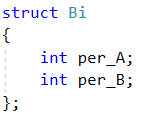


Рисунок 2. Структура Bi

1. Процедура ввода данных состоит из следующих этапов:

* Ввод числа стратегий первого игрока. Если введён символ или число меньше 0, будет выдано сообщение об ошибке.
* Ввод числа стратегий второго игрока. Если введён символ или число меньше 0, будет выдано сообщение об ошибке.
* Ввод весов каждого игрока в каждой ситуации (i, j). Если введён символ, будет выдано сообщение об ошибке.

Реализация функции ввода данных представлена на рисунке 3.

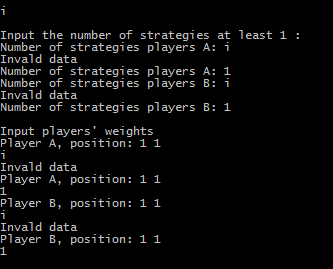


Рисунок 3. Ввод данных

1. Функция вывода матрицы осуществляет печать на экран матрицы игры в виде таблицы, где стратегии первого игрока идут по строкам, а стратегии второго игрока – по столбцам. Реализация функции вывода матрицы представлена на рисунке 4.

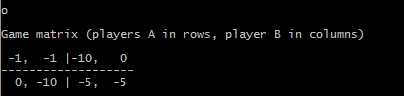


Рисунок 4. Вывод матрицы игры

1. Функция поиска равновесия по Нэшу представлена кодом ниже.

void Nash\_m(const int n, const int m, Bi\*\* res)

{

//выполняется проверка на существование данных

if ((n == 0) || (m == 0))

{

cout << endl << "You didn't insert valid data" << endl << endl;

return;

}

cout << endl;

vector <int>\* res\_j\_perB = new vector <int>[n];

for (int i = 0; i < n; ++i)//per\_B

{

vector<int> local;

int max\_B = INT\_MIN;

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

{

if (res[i][j].per\_B > max\_B)

{

max\_B = res[i][j].per\_B;

local.clear();

local.push\_back(j);

}

else if (res[i][j].per\_B == max\_B)

{

local.push\_back(j);

}

}

res\_j\_perB[i] = local;

}

vector <int>\* res\_i\_perA = new vector <int>[m];

for (int j = 0; j < m; ++j)//per\_A

{

vector<int> local;

int max\_A = INT\_MIN;

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

{

if (res[i][j].per\_A > max\_A)

{

max\_A = res[i][j].per\_A;

local.clear();

local.push\_back(i);

}

else if (res[i][j].per\_A == max\_A)

{

local.push\_back(i);

}

}

res\_i\_perA[j] = local;

}

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

{

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

{

for (int v = 0; v < res\_j\_perB[i].size(); ++v)

{

for (int x = 0; x < res\_i\_perA[j].size(); ++x)

{

if ((res\_j\_perB[i][v] == j) && (res\_i\_perA[j][x] == i))

{

cout << "Nash's Equilibrium : " << res[i][j].per\_A << ", " << res[i][j].per\_B << endl;

cout << "Strategy of player A : " << i + 1 << endl;

cout << "Strategy of player B : " << j + 1 << endl;

}

}

}

}

}

cout << endl;

}

Принцип работы: для каждого игрока ищется ситуация, в которой при определённой стратегии другого игрока исходный получает максимальное количество баллов. Если максимум один, запоминается номер стратегии, при котором он достигается. Если максимумов несколько, запоминаются номера всех стратегий, при которых этот максимум достигается.

Аналогичные действия производятся и для другого игрока.

Далее ищется пересечение стратегий, при которых был достигнут максимум обоих игроков. Полученные ситуации (одна или несколько) будут являться равновесиями по Нэшу. Работа данной функции представлена на рисунках 5-6.

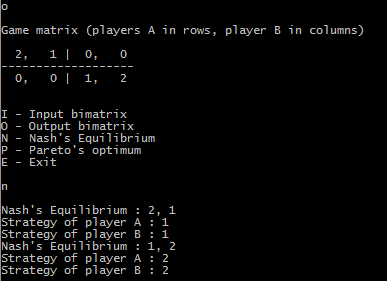


Рисунок 5. Равновесие по Нэшу для игры «Семейный спор»

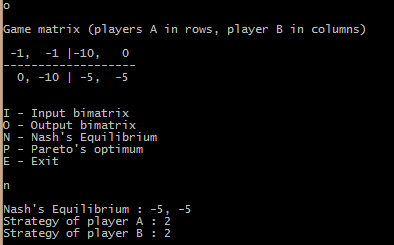


Рисунок 6. Равновесие по Нэшу для игры «Дилемма заключённого»

1. Функция поиска оптимума по Парето представлена кодом ниже.

void Pareto(const int n, const int m, Bi\*\* res)

{

//выполняется проверка на существование данных

if ((n == 0) || (m == 0))

{

cout << endl << "You didn't insert valid data" << endl << endl;

return;

}

cout << endl;

int max = INT\_MIN;

int\*\* loc\_res = new int\* [n];

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

{

loc\_res[i] = new int[m];

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

{

loc\_res[i][j] = res[i][j].per\_A + res[i][j].per\_B;

if (loc\_res[i][j] > max)

{

max = loc\_res[i][j];

}

}

}

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

{

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

{

if (loc\_res[i][j] == max)

{

cout << "Pareto's optimum : " << res[i][j].per\_A << ", " << res[i][j].per\_B << endl;

cout << "Strategy of player A : " << i + 1 << endl;

cout << "Strategy of player B : " << j + 1 << endl;

}

}

}

cout << endl;

}

Принцип работы: для каждой ситуации ищется сумма баллов для обоих игроков. Оптимумом по Парето является максимум среди таких сумм, их может быть несколько.

Работа данной функции представлена на рисунке 7.

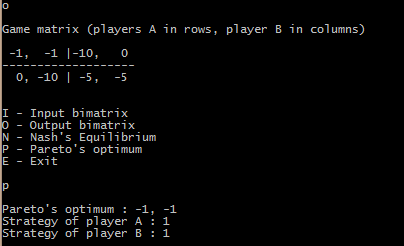
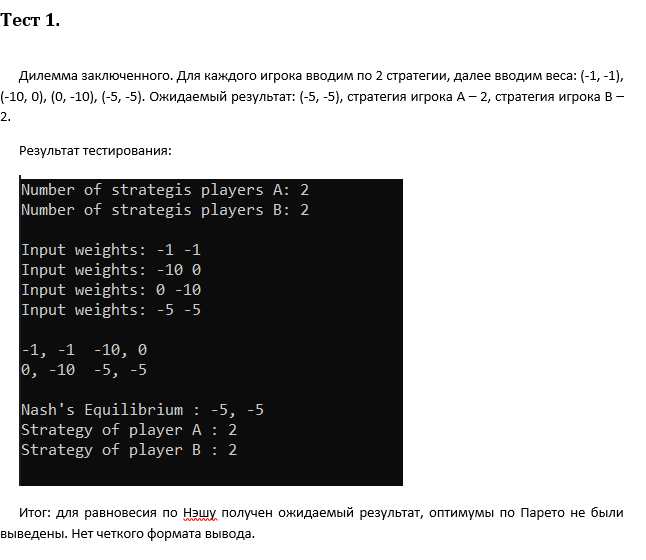


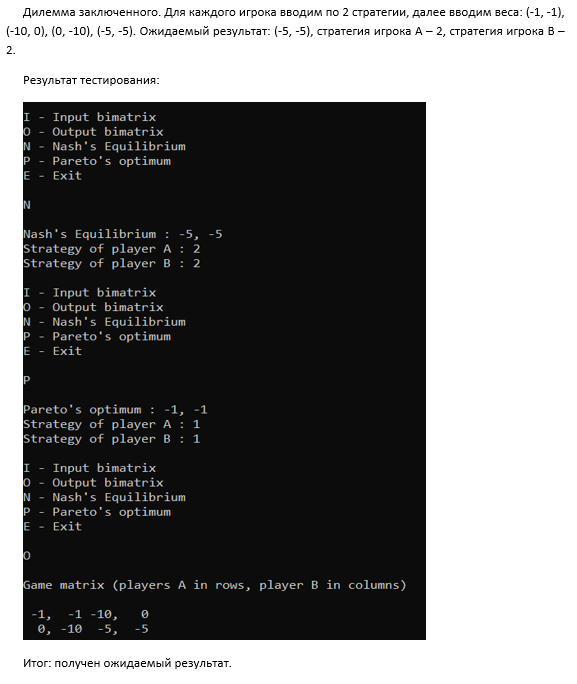
Рисунок 7. Оптимум по Парето для игры «Дилемма заключённого»

# Результаты тестов

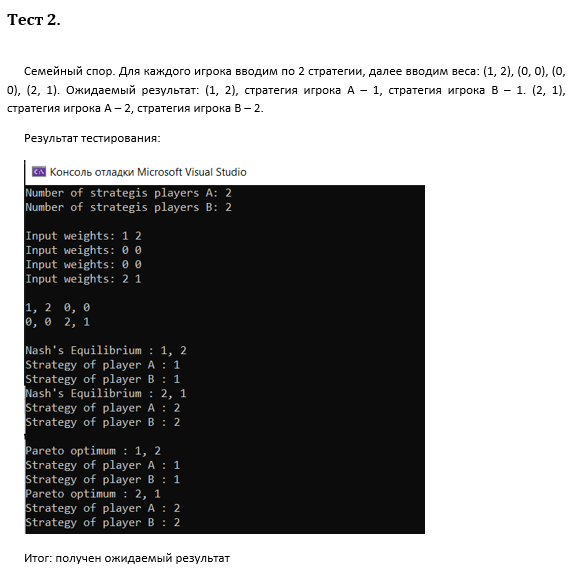
1. Тест 1



Есть некоторые проблемы, которые были исправлены в последующих версиях проекта, о чём говорит результат нового тестирования:

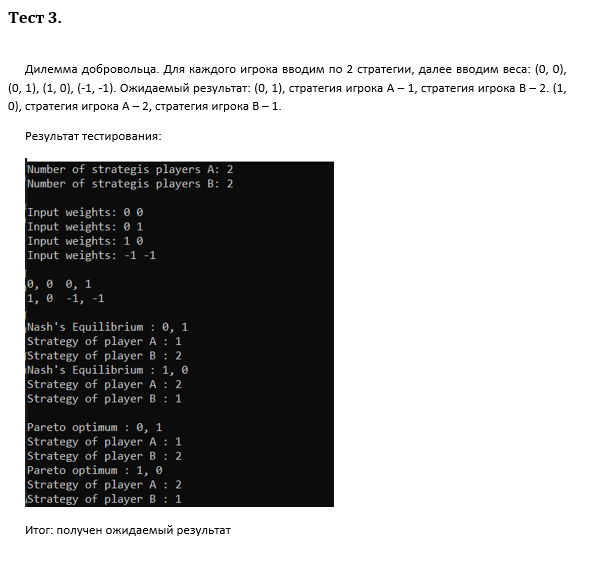


1. Тест 2



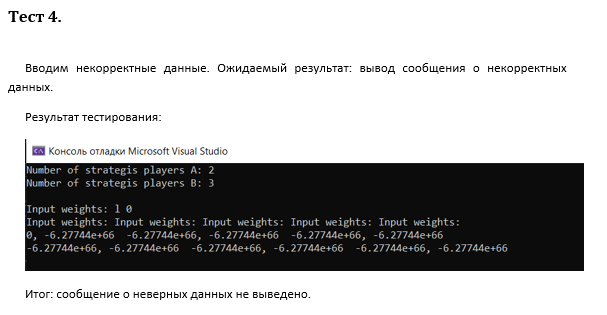
Проблемы не были выявлены.

1. Тест 3



Проблемы не были выявлены.

1. Тест 4



Есть некоторые проблемы, которые были исправлены в последующих версиях проекта, о чём говорит результат нового тестирования:

