Министерство образования и науки Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего профессионального образования
«Казанский (Приволжский) федеральный университет»

Институт математики и механики им. Н.И.Лобачевского

Кафедра высшей математики и математического моделирования

Выпускная квалификационная работа на тему:

Решение задач теории матричных игр с использованием среды программирования Delphi

Работа завершена:

«___»_____2015 г. _____ А.А.Малаев

Работа допущена к защите:

Научный руководитель

д.ф.-м.н, доцент

«___»_____2015 г. _____ А.Н.Фролов

Заведующий кафедрой ВМиММ,

д.ф-м.н., профессор

«___»_____2015 г. _____ Ю.Г.Игнатьев

Казань 2015

Содержание

Введение.....3

Глава I. Теория игр. Матричные игры.....5

1.1. Основные понятия теории игр.....5

1.2. Постановка матричной игры. Седловая точка.....6

1.3. Смешанные стратегии. Метод Гаусса.....9

1.4. Итерационный метод Брауна-Робинсона.....12

Глава II. Решение задач теории игр с помощью языка программирования Delphi.....15

2.1. Нахождение верхней и нижней цены игры. Седловая точка.....31

2.2. Решение матричных игр в смешанных стратегиях.....41

2.3. Решение матричных игр методом Брауна-Робинсона.....50

Заключение.....60

Спилок использованной литературы.....61

Введение

Целью данной работы является изучение теории игр, в частности теории матричных игр. Изучение литературы и использование среды программирования Delphi в решении задач теории матричных игр.

Теория игр была основана Джоном фон Нейманом и Оскаром Моргенштерном в их первой работе "The Theory of Games and Economic Behavior", изданной в 1944 году. В 1928 году в математических анналах фон Нейманом была опубликована статья "О теории общественных игр", в которой впервые было применено понятие "теория игр". Использование этого понятия объясняется схожестью логики принятия решений в таких играх, как шахматы и покер. Характерным для таких ситуаций является то, что результат для принимающего решение зависит не только от его решения, но и от того, какое решение примут другие. Поэтому оптимальный исход не может быть получен в результате принятия решения одним лицом.

Другим предшественником теории игр по праву считается французский математик Э. Борель (1871-1956). Некоторые фундаментальные идеи были независимо предложены А. Вальдом (1902-1950), заложившим основы нового подхода к статистической теории принятия решений.

Первые приложения теория игр нашла в математической статистике. Во время второй мировой войны и сразу после нее теорией игр серьезно заинтересовались военные, которые увидели в ней аппарат для исследования стратегических решений. Ее использовали как плодотворный источник теоретических моделей в экономике и социологии. Методы теории игр используются также в теории операций и в линейном программировании.

На промышленных предприятиях теория игр может применяться для выбора оптимальных решений, например, при создании рациональных запасов сырья, материалов, полуфабрикатов, когда противоборствуют две тенденции: увеличения запасов, гарантирующих бесперебойную работу производства, и сокращения запасов в целях минимизации затрат на их

хранение. В сельском хозяйстве теория игр может применяться при решении таких экономических задач, как выбор для посева одной из возможных культур, урожай которых зависит от погоды, если известны цена единицы той или иной культуры и средняя урожайность каждой культуры в зависимости от погоды (например, будет ли лето засушливым, нормальным или дождливым); в этом случае одним из игроков выступает сельскохозяйственное предприятие, стремящееся обеспечить наибольший доход, а другим — природа.

На практике часто появляется необходимость согласования действий фирм, объединений, министерств и других участников проектов в случаях, когда их интересы не совпадают. В таких ситуациях теория игр позволяет найти лучшее решение для поведения участников, обязанных согласовывать действия при столкновении интересов.

Теория игр все шире проникает в практику экономических решений и исследований. Ее можно рассматривать как инструмент, помогающий повысить эффективность плановых и управленческих решений.

Глава I. Теория игр. Матричные игры.

1.1. Основные понятия теории игр.

Термин «игра» применяется для обозначения совокупности правил и соглашений, которыми руководствуются субъекты, поведение которых здесь рассматривается. Игра – упрощенная модель конфликта. Для решения конфликтных ситуаций разработан специальный аппарат – теория игр. Стороны, участвующие в игре – игроки. Исход игры называется выигрышем. Правила – система условий определяющая: 1) варианты действия игроков, 2) объем информации каждого игрока о поведении партнеров, 3) выигрыш к которому приводит каждая совокупность действий. Как правило, выигрыш (или проигрыш) может быть задан количественно; например, можно оценить проигрыш нулем, выигрыш единицей, а ничью в ½. Игра парная, если в ней два игрока и множественная, если больше. Игра, в которой выигрыш одного игрока равен проигрышу второго – называется игрой с нулевой суммой или антагонистической. Ход игрока – выбор и осуществление одного из предусмотренных правилами действий. Ходы могут быть личными и случайными. Личный ход – это сознательный выбор игроком одного из возможных действий (например, ход в шахматной партии). Случайный ход – это случайно выбранное действие (выбор карты из перетасованной колоды). Стратегией игрока называется совокупность правил, определяющих выбор его действия при каждом личном ходе в зависимости от сложившейся ситуации. Игра называется конечной, если у каждого игрока имеется конечное число стратегий, и бесконечной – в противном случае. Для того чтобы найти решение игры, следует для каждого игрока найти стратегию, которая удовлетворяет условию оптимальности, т.е. один игрок должен получить максимальный выигрыш, в то же время второй игрок должен иметь минимальный проигрыш. Такие стратегии называют оптимальными,

оптимальные стратегии должны удовлетворять условию устойчивости, т.е. любому игроку было бы невыгодно отклонятся от выбранной стратегии.

1.2. Постановка матричной игры. Седловая точка

В простейшей математической модели конечной конфликтной ситуации имеется два участника и выигрыш одного равен проигрышу другого. Такая модель называется антагонистической игрой двух лиц с нулевой суммой. Игра состоит из двух ходов: игрок А выбирает одну из возможных стратегий A_i, $i=\overline{1,m}$, а игрок В выбирает одну из возможных стратегий B_j, $j=\overline{1,n}$. Каждый выбор производится при полном незнании выбора соперника. В результате выигрыш игроков составит соответственно a_{ij} и (- a_{ij}). Цель игрока А - максимизировать величину a_{ij}, а игрока В - минимизировать эту величину.

Матрица, составленная из величин a_{ij}, $i=\overline{1,m}$, $j=\overline{1,n}$,

$$A=\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ & & \dots & \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}_{m\times n}$$

является платежной матрицей, или матрицей игры. Каждый элемент платежной матрицы a_{ij}, $i=\overline{1,m}$, $j=\overline{1,n}$ равен выигрышу А (проигрышу В), если он выбрал стратегию A_i, $i=\overline{1,m}$, а игрок В выбирал стратегию B_j, $j=\overline{1,n}$.

Задача каждого из игроков - найти наилучшую стратегию игры, при этом предполагается, что противники одинаково разумны, и каждый из них делает все, чтобы получить наибольший доход.

Если игрок А выбрал стратегию A_i, $i=\overline{1,m}$, то в худшем случае (например, если его ход известен В) он получит выигрыш $\alpha_i = \min\limits_j a_{ij}$. Предвидя

такую возможность, игрок А должен выбрать такую стратегию, чтобы максимизировать свой минимальный выигрыш.

$$\alpha = \max_i \alpha_i = \max_i (\min_j a_{ij}) .$$

Величина α - гарантированный выигрыш игрока А называется нижней ценой игры. Стратегия $A_{i\text{опт}}$, обеспечивающая получение выигрыша α, называется максиминной.

Если первый игрок будет придерживаться своей максиминной стратегии, то у него есть гарантия, что он в любом случае выиграет не меньше α.

Аналогично определяется наилучшая стратегия второго игрока. Игрок В при выборе стратегии B_j, $j = \overline{1,n}$ в худшем случае получит проигрыш $\beta_j = \max_i a_{ij}$. Он выбирает стратегию $B_{j\text{опт}}$, при которой его проигрыш будет минимальным и составит

$$\beta = \min_j \beta_j = \min_j (\max_i a_{ij}) .$$

Величина β - гарантированный проигрыш игрока В называется верхней ценой игры. Стратегия $B_{j\text{опт}}$, обеспечивающая получение проигрыша β, называется минимаксной.

Если второй игрок будет придерживаться своей минимаксной стратегии, то у него есть гарантия, что он в любом случае проиграет не больше β.

Фактический выигрыш игрока А (проигрыш игрока В) при разумных действиях партнеров ограничен верхней и нижней ценой игры. Для матричной игры справедливо неравенство $\alpha \le \beta$.

Если $\alpha = \beta = v$, т.е.

$$\max_i(\min_j a_{ij}) = \min_j(\max_i a_{ij}) = v,$$

то выигрыш игрока А (проигрыш игрока В) определяется числом v. Оно называется ценой игры.

Если $\alpha = \beta = v$, то такая игра называется игрой с седловой точкой, элемент матрицы $a_{i_{опт}\, j_{опт}} = v$, соответствующий паре оптимальных стратегий ($A_{i_{опт}}$, $B_{j_{опт}}$), называется седловой точкой матрицы. Этот элемент является ценой игры.

Седловой точке соответствуют оптимальные стратегии игроков. Их совокупность - решение игры, которое обладает свойством: если один из игроков придерживается оптимальной стратегии, то второму отклонение от своей оптимальной стратегии не может быть выгодным.

Если игра имеет седловую точку, то говорят, что она решается в чистых стратегиях.

Наличие седловой точки в игре - это далеко не правило, скорее, исключение. Существует разновидность игр, которые всегда имеют седловую точку, и, значит, решаются в чистых стратегиях. Это так называемые игры с полной информацией.

Игрой с полной информацией называется такая игра, в которой каждый игрок при каждом личном ходе знает всю предысторию ее развития, т.е. результаты всех предыдущих ходов.

Каждая игра с полной информацией имеет седловую точку, следовательно, решается в чистых стратегиях, т.е. имеется пара оптимальных чистых стратегий, дающая устойчивый выигрыш, равный ν.

Если такая игра состоит только из личных ходов, то при применении каждым игроком своей оптимальной чистой стратегии она должна кончаться выигрышем, равным цене игры. Скажем, шахматная игра, как игра с полной информацией, либо всегда кончается выигрышем белых, либо всегда - выигрышем черных, либо всегда – ничьей.

1.3. Смешанные стратегии

Если платежная матрица не имеет седловой точки, т.е. $\alpha < \beta$ и $\alpha \le \nu \le \beta$, то поиск решения игры приводит к применению сложной стратегии, состоящей в случайном применении двух и более стратегий с определенными частотами.

Сложная стратегия, состоящая в случайном применении всех стратегий с определенными частотами, является смешанной.

В игре, матрица которой имеет размерность m × n, стратегии первого игрока задаются наборами вероятностей $\overline{X}$ (x_1, x_2,..., x_m), с которыми игрок применяет свои чистые стратегии. Эти наборы можно рассмотреть как m-мерные векторы, для координат которых выполняются условия

$\sum_{i=1}^{m} x_i = 1$, $x_i \ge 0$, $i = \overline{1,m}$.

Аналогично для второго игрока наборы вероятностей определяют n-мерные векторы $\overline{Y}$ (y_1, y_2,..., y_n), для координат которых выполняются условия

$\sum_{j=1}^{n} y_j = 1$, $y_j \ge 0$, $j = \overline{1,n}$.

Выигрыш первого игрока при использовании смешанных стратегий определяют как математическое ожидание выигрыша, т.е. он равен

$$\sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij} x_i y_j .$$

Согласно теории Неймана, каждая конечная игра имеет, по крайней мере, одно решение, возможно, в области смешанных стратегий. Применение

оптимальной стратегии позволяет получить выигрыш, равный цене игры: $\alpha \le v \le \beta$. Применение первым игроком оптимальной стратегии $\overline{X}_{\text{опт}}$ должно обеспечить ему при любых действиях второго игрока выигрыш не меньше цены игры. Поэтому выполняется соотношение

$$\sum_{i=1}^{m} a_{ij} x_{i\,onm} \ge v, \quad j = \overline{1,n}.$$

Аналогично для второго игрока оптимальная стратегия $\overline{Y}_{\text{опт}}$ должна обеспечить при любых стратегиях первого игрока проигрыш, не превышающий цену игры, т.е. справедливо соотношение

$$\sum_{j=1}^{n} a_{ij} y_{j\,onm} \le v, \quad i = \overline{1,m}.$$

Если платежная матрица не содержит седловой точки, то задача определения смешанной стратегии тем сложнее, чем больше размерность матрицы. Поэтому матрицы большой размерности целесообразно упростить, уменьшив их размерность путем вычеркивания дублирующих (одинаковых) и не доминирующих стратегий. Дублирующие стратегии – это стратегии, у которых соответствующие элементы платежной матрицы одинаковы.

Если все элементы i-й строки платежной матрицы больше соответствующих элементов k-й строки, то i-я стратегия игрока А будет доминирующей над k-й стратегией. Если все элементы j-го столбца платежной матрицы меньше соответствующих элементов k-го столбца, то j-я стратегия игрока В является доминирующей над k-й стратегией.

Решение матричной игры в смешанных стратегиях Методом Гаусса

Пусть имеется произвольная матричная игра без седловой точки с платёжной матрицей $\|a_{ij}\|_{m\times n}$. Как известно, основная задача теории игр заключается в определении оптимальных стратегий игроков и цены игры. Пусть $p^* = (p_1, p_2, \ldots, p_m)$, $q^* = (q_1, q_2, \ldots, q_n)$ – оптимальные смешанные стратегии игроков *A* и *B* соответственно. Соотношения между p^* и ценой игры **V** можно формализовать в виде системы уравнений:

$$\begin{cases} a_{11}p_1 + a_{21}p_2 + \ldots + a_{m1}p_m = v, \\ a_{12}p_1 + a_{22}p_2 + \ldots + a_{m2}p_m = v' \\ \qquad\qquad \ldots \\ a_{1n}p_1 + a_{2n}p_2 + \ldots + a_{mn}p_m = v, \\ p_1 + p_2 + \ldots + p_m = 1 \end{cases} \qquad (1)$$

причём $p_i \geq 0 \ \ (i = \overline{1,m})$

Аналогично соотношения между q^* и ценой игры **V** можно формализовать в виде системы неравенств:

$$\begin{cases} a_{11}q_1 + a_{12}q_2 + \ldots + a_{1n}q_n = v, \\ a_{21}q_1 + a_{22}q_2 + \ldots + a_{2n}p_n = v' \\ \qquad\qquad \ldots \\ a_{m1}q_1 + a_{m2}q_2 + \ldots + a_{mn}q_n = v, \\ q_1 + q_2 + \ldots + q_m = 1 \end{cases} \qquad (2)$$

причем $q_j \geq 0 \ (j = \overline{1,n})$

Решая полученные уравнения (1) и (2) методом Гаусса получаем:

- цену игры : V
- оптимальные смешанные стратегии $p^* = (p_1^*, p_2^* \ldots p_m^*)$ и $q^* = (q_1^*, q_2^* \ldots q_m^*)$.

1.4. Итеративный метод Брауна-Робинсона (метод фиктивного разыгрывания)

Часто в практических задачах нет необходимости находить точное решение матричной игры. Достаточно найти приближённое решение, которое даёт средний выигрыш, близкий к цене игры и приближённые оптимальные стратегии игроков.

Ориентировочное значение цены игры может дать уже простой анализ матрицы выигрышей и определение нижней и верхней цен игры. Если они близки, то поисками точного решения заниматься не обязательно, так как достаточно выбрать чистые минимаксные стратегии. Если же они не близки, можно получить приемлемое для практики решение с помощью численных методов решения игр, из которых рассмотрим метод итераций.

Пусть разыгрывается матричная игра Γ_A с матрицей $A=\{a_{ij}\}$ размера $(m\times n)$. Идея метода – многократное фиктивное разыгрывание игры с заданной матрицей. Одно разыгрывание игры будем называть партией, число которых неограниченно.

В 1-ой партии оба игрока выбирают совершенно произвольные чистые стратегии. Пусть игрок 1 выбрал i-ю стратегию, а игрок 2 – j-ю стратегию. Во второй партии игрок 1 отвечает на ход игрока 2 той своей стратегией, которая даёт ему максимальный выигрыш. В свою очередь, игрок 2, отвечает на этот ход игрока 1 своей стратегией, которая обращает его проигрыш в минимум. Далее третья партия.

С ростом числа шагов процесса смешанные стратегии, которые приписываются игрокам, приближаются к их оптимальным стратегиям. Этот процесс приближённого нахождения оптимальных стратегий игроков называется *итеративным* , а его шаги – итерациями.

Итак, предположим, что за первые k разыгрываний игрок 1 использовал i-ю чистую стратегию ξ_i^k раз (i=1,…,m), а игрок 2 j-ю чистую стратегию η_j^k

раз (j=1,…,n). Тогда их смешанными стратегиями будут векторы $x^k = (\xi_1^k/k,...,\xi_m^k/k), y^k = (\eta_1^k/k,...,\eta_n^k/k)$.

Игрок 1 следит за действиями игрока 2 и с каждым своим ходом желает получить как можно больший выигрыш. Поэтому в ответ на применение игроком 2 своей смешанной стратегии y^k, он будет использовать чистую стратегию i_{k+1} , которая обеспечит ему лучший результат при разыгрывании (k+1)-ой партии. Игрок 2 поступает аналогично. В худшем случае каждый из них может получить:

$$\bar{\nu}^k = \max_i \sum_j^n a_{ij}\eta_j^k = \sum_j^n a_{i_{k+1}}\eta_j^k$$

$$\underline{\nu}^k = \min_j \sum_i^m a_{ij}\xi_i^k = \sum_i^m a_{ij_{k+1}}\xi_i^k$$

где $\bar{\nu}^k$ - наибольшее значение проигрыша игрока 2 и $\underline{\nu}^k$ - наименьшее значение выигрыша игрока 1.

Рассмотрим отношения, которые определяют средние значения проигрыша игрока 2 и выигрыша игрока 1:

$$\bar{\nu}^k/k = \max_i \sum_j^n a_{ij}\eta_j^k/k = \sum_j^n a_{i_{k+1}j}\eta_j^k/k$$

$$\underline{\nu}^k/k = \min_j \sum_i^m a_{ij}\xi_i^k/k = \sum_i^m a_{ij_{k+1}}\xi_i^k/k$$

Пусть ν - цена матричной игры Γ_A. Её значение будет больше выигрыша игрока 1, но меньше проигрыша игрока 2, т. е.

$$\max_k \underline{\nu}^k/k \le \nu \le \min_k \bar{\nu}^k/k. \qquad (1)$$

Таким образом, получен итеративный процесс, позволяющий находить приближённое решение матричной игры, при этом степень близости приближения к истинному значению игры определяется длиной интервала

$$\left[\max_k \underline{\nu}^k/k, \min_k \bar{\nu}^k/k\right].$$

Сходимость алгоритма гарантируется следующей теоремой.

Теорема. $\lim\limits_{k\to\infty}(\min\limits_{k} \bar{\nu}^{k} \big/ k) = \lim\limits_{k\to\infty}(\max\limits_{k} \underline{\nu}^{k} \big/ k) = \nu$.

Схема доказательства.

Лемма. Для всякой матрицы A и $\forall\varepsilon>0$ *существует такое* k_0, *что*

$$\min_{k_0} \bar{\nu}^{k} / k - \max_{k_0} \underline{\nu}^{k} / k < \varepsilon .$$

При предельном переходе в неравенстве (1) при $k\to\infty$ имеем:

$$\lim_{k\to\infty}(\max_{k} \underline{\nu}^{k} \big/ k) \le \nu \le \lim_{k\to\infty}(\min_{k} \bar{\nu}^{k} / k). \qquad (2)$$

Отсюда получаем оценку разности пределов:

$$0 \le \lim_{k\to\infty}(\min_{k} \bar{\nu}^{k} \big/ k) - \lim_{k\to\infty}(\max_{k} \underline{\nu}^{k} \big/ k).$$

Из *леммы* следует, что

$$\lim_{k\to\infty}(\min_{k} \bar{\nu}^{k} / k - \max_{k} \underline{\nu}^{k} / k) < \varepsilon .$$

На основании неравенства (1) имеем: $\begin{matrix} \liminf\limits_{k\to\infty}(\min\limits_{k} \bar{\nu}^{k} / k) \ge \nu \\ \limsup\limits_{k\to\infty}(\max\limits_{k} \underline{\nu}^{k} / k) \le \nu \end{matrix}$.

Следовательно, в силу ограниченности пределов

$$\lim_{k\to\infty}(\min_{k} \bar{\nu}^{k} / k) - \lim_{k\to\infty}(\max_{k} \underline{\nu}^{k} / k) < \varepsilon .$$

Получаем оценку для разности пределов:

$$0 \le \lim_{k\to\infty}(\min_{k} \bar{\nu}^{k} / k) - \lim_{k\to\infty}(\max_{k} \underline{\nu}^{k} / k) < \varepsilon \text{ для } \forall\varepsilon>0.$$

Можем заключить, что $\lim\limits_{k\to\infty}(\min\limits_{k} \bar{\nu}^{k} / k) = \lim\limits_{k\to\infty}(\max\limits_{k} \underline{\nu}^{k} / k)$. Осталось показать равенство пределов ν. Это следует из неравенства (2).

Итак, $\lim\limits_{k\to\infty}(\min\limits_{k} \bar{\nu}^{k} / k) = \lim\limits_{k\to\infty}(\max\limits_{k} \underline{\nu}^{k} / k) = \nu$.

Глава II. Решение задач теории игр с помощью языка программирования Delphi

Математическое моделирование как инструмент познания завоевывает все новые и новые позиции в различных областях деятельности человека. Оно становится главенствующим направлением в проектировании и исследовании новых систем, анализе свойств существующих систем, выборе и обосновании оптимальных условий их функционирования и т.п.

Изучение математического моделирования открывает широкие возможности для осознания связи информатики с математикой и другими науками. Абстрактное моделирование с помощью компьютеров – вербальное, информационное, математическое – в наши дни стало одной из информационных технологий в познавательном плане исключительно мощной.

Общее в моделях то, что во всех случаях модель в определённом смысле заменяла сам исследуемый объект. Вместо исходного объекта (оригинала) использовалась его модель, модель являлась представлением объекта в некоторой форме, отличной от формы его реального существования.

Модель – это материальный или идеальный объект, который строится для изучения исходного объекта (оригинала) и который отражает наиболее важные качества и параметры оригинала.

Практически во всех науках о природе, живой и неживой, об обществе, построение и использование моделей является мощным орудием познания. Реальные объекты и процессы бывают столь многообразны и сложны, что лучшим способом изучения часто является построение модели, отражающей лишь какую – то часть реальности.

В любом случае модель строится для с целью узнать про объект что – либо новое или сохранить об объекте информацию, которая может стать недоступной в будущем.

Как правило, процесс изучения, связанный с использованием моделей и называемый моделированием не заканчивается созданием одной модели. Построив модель и получив с её помощью, какие – либо результаты, соотнося их с реальностью и если это соотношение даёт неудовлетворительные результаты, то в построенную модель вносят коррективы или даже создают другую модель. В случае достижения хорошего соответствия с реальностью выясняют границы применения модели. Это очень важный вопрос, он решается путём сравнения модели с оригиналом путём сравнения предсказаний, полученных с помощью компьютерной модели. Если это сравнение даёт удовлетворительные результаты, то модель принимают на вооружение, если нет, приходится создавать другую модель.

Математическое моделирование относится к классу знакового моделирования, при этом модели могут создаваться из любых математических объектов, чисел, функций, уравнений, графиков, графов.

В моделировании существует два пути:

- Модель может быть похожей копией объекта, выполненной из другого материала и в другом масштабе, с отсутствием ряда деталей.
- Модель может отражать реальность более абстрактно – словесным описанием формализованным по каким – то правилам, соотношениям.

Существует следующая классификация абстрактных моделей:

Вербальные:

- Эти модели используют последовательности предложений на формализованных диалектах естественного языка для описания той или иной области действительности.

Математические:

- Это очень широкий класс знаковых моделей, основанных на формальных языках над конечными алфавитами, широко использующих те или иные математические методы.

Информационные:

- Это класс знаковых моделей описывающих информационные процессы в системах самой разнообразной природы.

Граница между вербальными, математическими и информационными моделями может быть проведена весьма условно; можно считать информационные методики подклассом математических моделей.

Математическая модель выражает существенные черты объекта или процесса языком уравнений и других математических средств. Огромный толчок развитию математическому моделированию дало появление ЭВМ, хотя сам метод появился тысячи лет назад.

Понятие «аналитическое» решение и «компьютерное» решение не противостоят друг другу, так как:

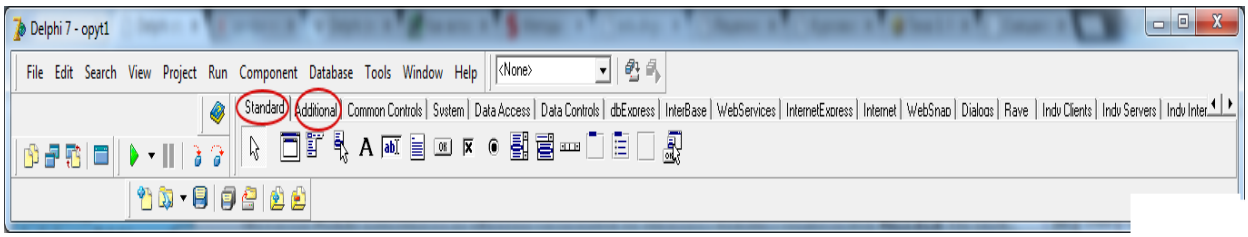
Всё чаще компьютеры при математическом моделировании используются не только для численных расчётов, но и для аналитических преобразований.

Результат аналитического исследования часто выражен в столь сложной форме, что при взгляде на неё не складывается восприятие описываемого ею процесса. Эту формулу можно протабулировать, представить графически, проиллюстрировать в динамике, то есть проделать то, что называется визуализацией абстракции.

В нашем случае мы будем математически моделировать с помощью среды или языка программирования Delphi. Коротко о Delphi, Delphi – это императивный, структурированный, объектно-ориентированный язык программирования со строгой статической типизацией переменных. Основная область использования – написание прикладного программного обеспечения.

Основная наша задача, да и цель всей ВКР написание программного обеспечения для решения задач (примеров) теории «Матричных игр» с заданной платежной матрицей.

Что будет использоваться в нашем продукте, какие компоненты Delphi?



В основном мы будем работать со страницами Standart и Additional главного меню, в режиме *приложения*, для этого необходимо просто запустить программу и в открытом окне продолжить работу. Немного информации о ранее сказанных страницах:

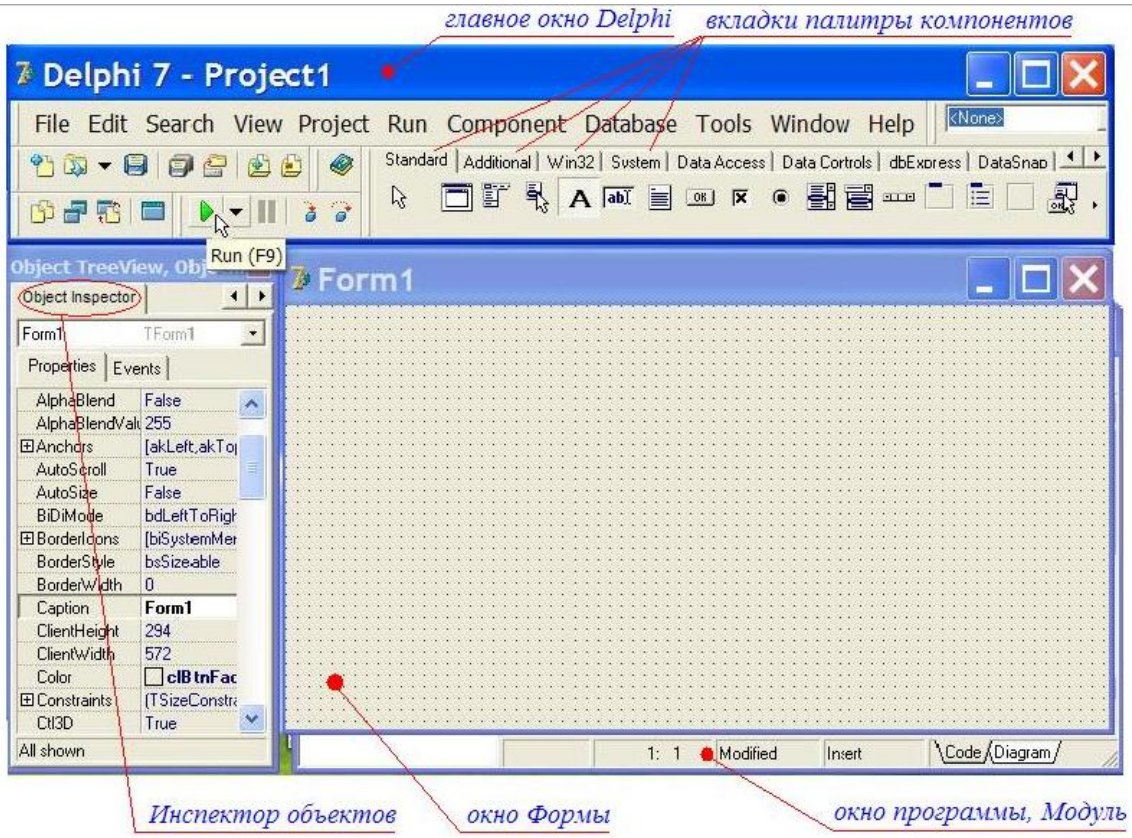
Standart. На этой странице расположены стандартные для ***Windows*** интерфейсные элементы, применяющиеся наиболее часто. В следующей таблице представлен список компонентов страницы **Standart**:

Пикто-грамма	Имя	Назначение
	MainMenu	**Главное меню** программы. Компонент способен создавать и обслуживать сложные иерархические меню.
	PopupMenu	**Всплывающее меню**. Обычно это меню появляется после нажатия правой кнопкой мыши.
A	***Label***	**Метка**. Используется для размещения не очень длинных сообщений в виде статического текста
ab	***Edit***	**Строка ввода**. Предназначена для ввода пользователем текстовой информации в виде одной строки. Имеет возможности по управлению вводимой информацией, например, переопределению символов до их появления в поле ввода, что используется в формах для ввода пароля.
	Memo	Многострочный **текстовый редактор**. Используется для ввода пользователем и отображения многострочного текста **без функций форматирования**.
OK	***Button***	**Командная кнопка**. Используется для реализации в программе команд с помощью обработчика события OnClick этого компонента.
	CheckBox	**Независимый переключатель**. Используется его свойство ***Checked*** (отмечено), имеющее значения ***true*** или ***false***, меняющееся при щелчке

		мышью.
	RadioButton	**Зависимый переключатель**. Используется для выбора **только одного** из нескольких вариантов. Для этого компонент объединяется как минимум с одним или несколькими такими же компонентами в группу. Щелчок по компоненту приводит к его выделению и снятию выделения ранее выбранного компонента. Также имеет свойство ***Checked***.
	ListBox	**Список выбора**. Содержит список предлагаемых вариантов (опций) и даёт возможность проконтролировать текущий выбор.
	ComboBox	**"Выпадающий" список** выбора. Представляет собой комбинацию компонентов **Edit**и **ListBox.**
	ScrollBar	**Полоса прокрутки**. Представляет собой вертикальную или горизонтальную полосу, управляющую визуальным представлением компонентов, не помещающихся целиком в окне программы.
	GroupBox	**Контейнер группы** компонентов. Используется для группировки нескольких связанных по смыслу компонентов.
	RadioGroup	Группа **зависимых переключателей**. Содержит специальные свойства для обслуживания нескольких связанных между собой зависимых переключателей.
	Panel	**Панель**. Этот компонент, как и GroupBox, служит для объединения нескольких компонентов. Содержит внутреннюю и внешнюю кромки, что позволяет создавать эффекты "вдавленности" и "выпуклости".
	ActionList	**Список действий**. Служит для централизованной реакции программы на действия пользователя, связанные с выбором одного из группы однотипных управляющих элементов, таких как опции меню, кнопки и т.д.

Из данного списка компонентов со страницы Standart, для создания приложения пригодятся компоненты *MainMenu, Label, Edit, Memo* и *Button.* Со страницы Additional мы воспользуемся компонентом *StringGrid,* которая понадобится для ввода платежной матрицы пользователем в программу, для дальнейшей ее обработки и решения поставленной задачи.

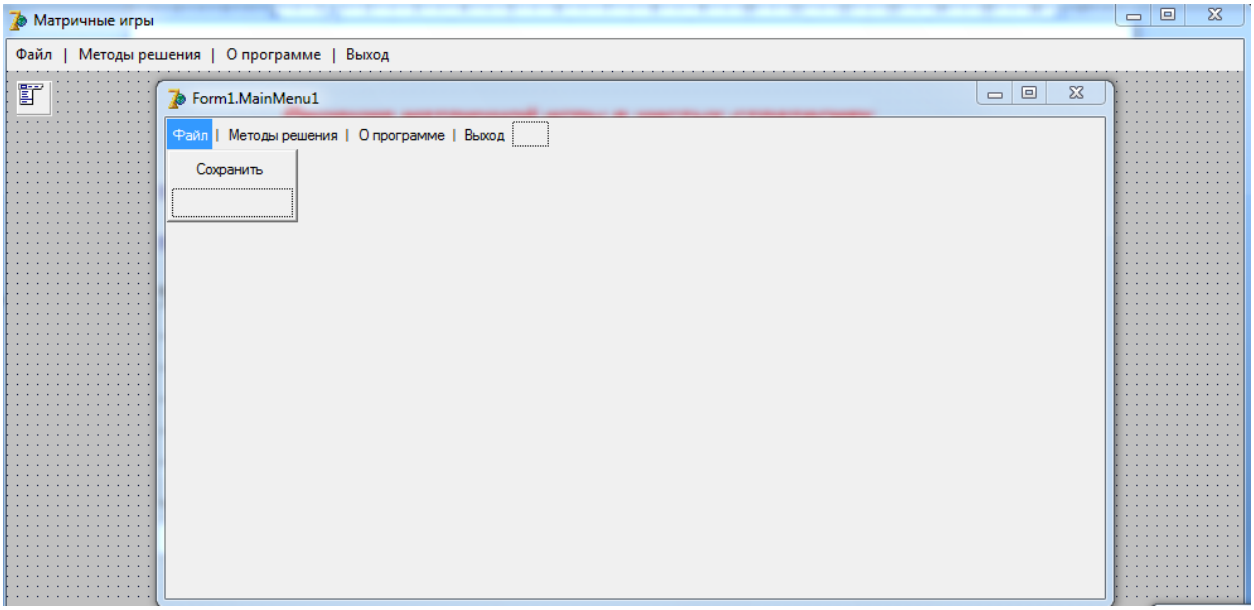
Приступим к созданию нашего приложения. Первое что мы должны сделать, это запустить Delphi, перед нами появится во такое окно:



Перед нами четыре **окна Delphi**. Вверху во всю ширину экрана окно управления проектом и средой разработки - **главное окно Delphi.** При его сворачивании сворачиваются и все остальные. Слева - **Инспектор объектов.** В нём задаются свойства составляющих нашу программу компонентов. И наконец, в центре одно над другим два окна Delphi, окно формы будущей программы и окно программной начинки. Прямо перед нами - окно, которое в Delphi называется *Форма*. Именно Форма является визуальным прообразом нашей будущей программы.

Переименуем нашу форму как «Матричные игры», просто изменим свойство *Caption* в инспекторе объектов нашей формы из Form1 в Матричные игры. Как упоминалось ранее, нам для работы пригодятся несколько компонентов со страницы Standart и Additional. Для этого просто нажимаем выбираем на нужной нам странице компонент, который мы будем использовать и закидываем его на форму. Первым чем мы воспользуемся будет компонент *MainMenu,* который пригодится нам для интерфейса нашего приложения. Установим данный компонент на форму, и подпишем

соответствующим образом. Вот как будет выглядеть наша форма, после установки компонента:



Меню Файл будет состоять из подменю Сохранить. Сразу представим код программы, который будет выполняться при нажатии на эту кнопку:

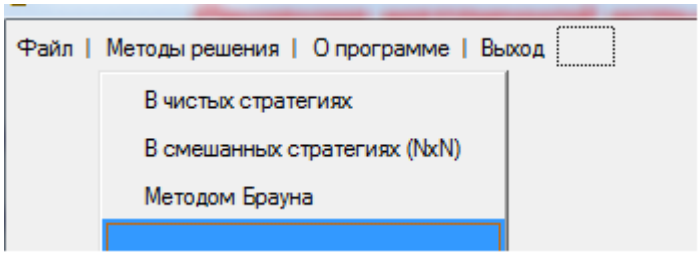
```
procedure TForm1.N13Click(Sender: TObject);
var
Bit:TBItmap;
begin
BitMap:=TBitMap.Create;
ShowMessage('Файл сохранен на диске "D"');
Try
Rect:=Bounds(0,0,Form1.Width,Form1.Height);
BitMap.Width:=Form1.Width;
BitMap.Height:=Form1.Height;
BitMap.Canvas.CopyRect(Rect,Form1.Canvas,Rect);
BitMap.SaveToFile('d:\bitmap.bmp');
finally
BitMap.Free
end;
```

end;

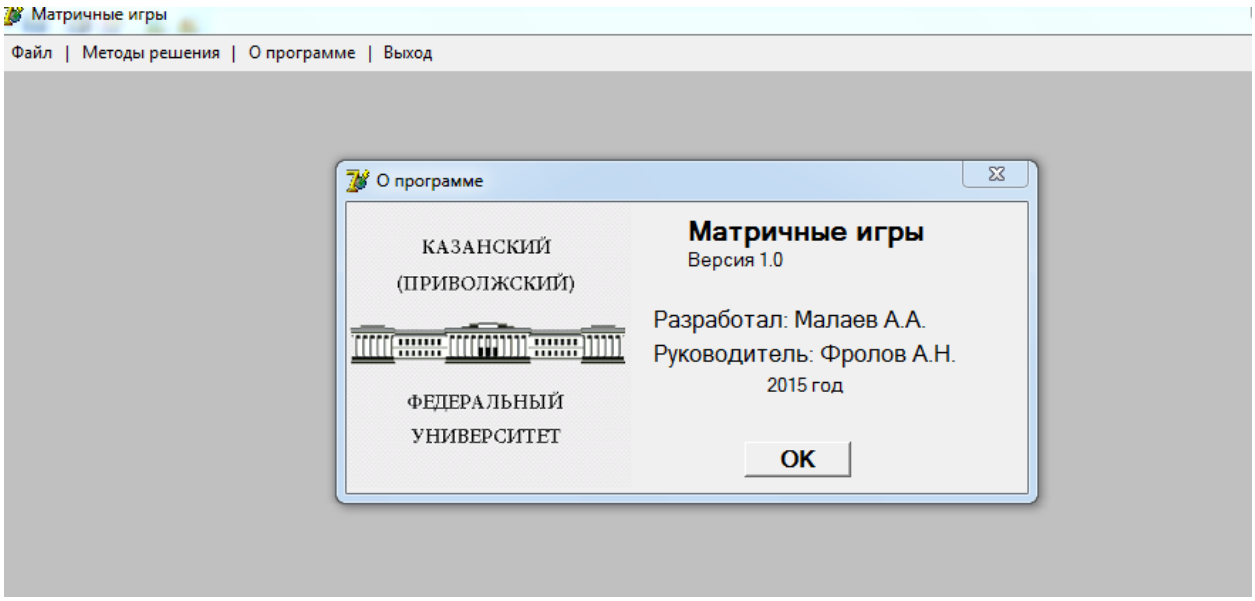
Данный код сохраняет на диске D компьютера канвас формы в виде картинку, то есть делает скриншот видимой части формы на экране и сохраняет его как картинку формата «bmp» с названием «bitmap».

Далее, меню Методы решения будет содержать в себе подменю:

1. В чистых стратегиях.
2. В смешанных стратегиях (NxN).
3. Методом Брауна.



Описание работы этих кнопок мы рассмотрим отдельно. Далее меню «О программе» выводит на экран стандартное окно о программе, хранящее в себе информацию о программе:



Далее также следует стандартная кнопка меню «Выход», которая закрывает основную форму, а также, если открыта окно «о программе» выполняется закрытие и этого окна. Код данной кнопки имеет вид:

procedure TForm1.N22Click(Sender: TObject);

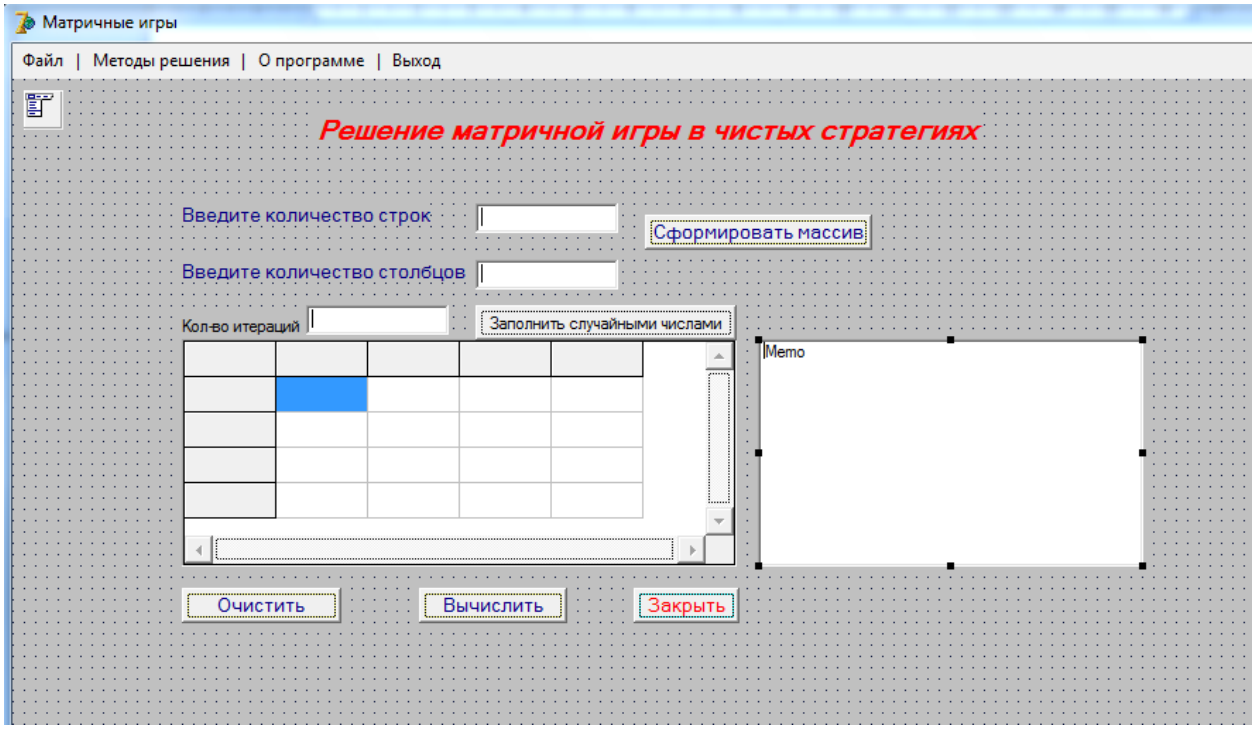
begin

Close;

Form2.Close;

end;

Теперь о компонентах, которые нам понадобятся для ввода и вывода информации пользователем, перетащим эти компоненты на форму. Нам понадобятся 7 компонентов Label, 5 компонентов Button, 3 Edit-a, Memo и компонент StringGrid со страницы Additional. И переименуем их так как нам нужно, то есть как было в случае с формой изменим свойство *Caption*. Вот как это будет выглядеть:

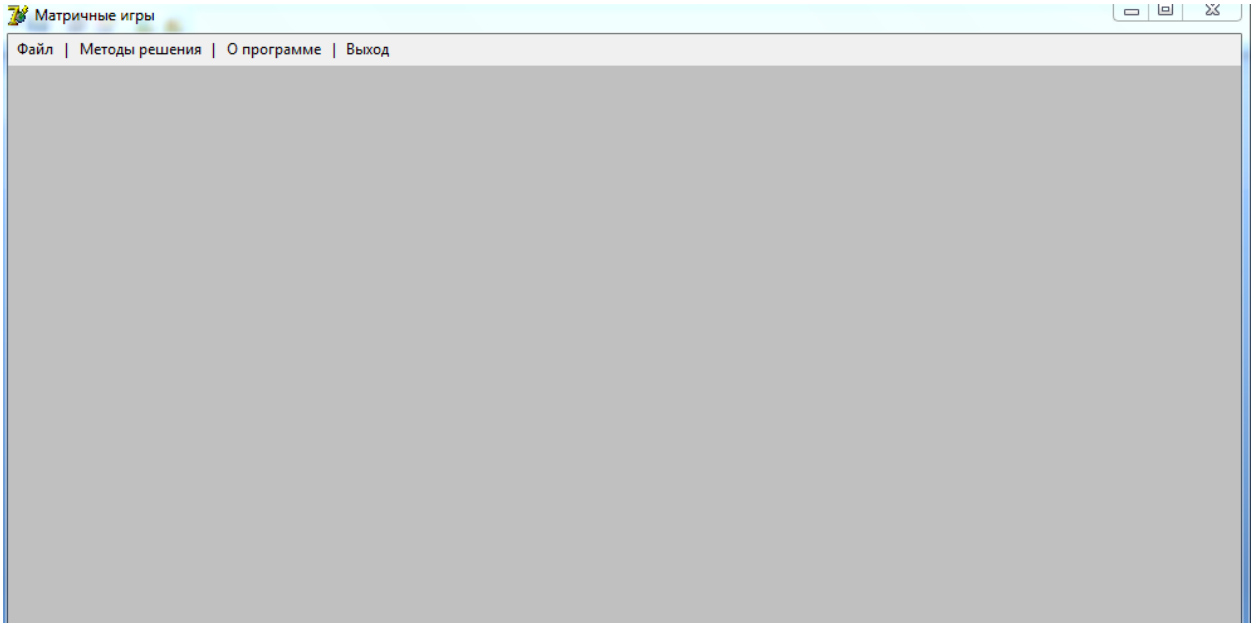


Теперь надо сделать так чтобы при запуске программы форма была чистая без каких либо компонентов, а элементы которые нам нужны появлялись по вызову наших методов. То есть, при нажатии на соответствующую кнопку появлялись те элементы, которые нужны для работы каждого из наших трех методов. Для этого делаем двойной клик по нашей форме, автоматически в модуле проекта прописывается процедура *Form.Create*, нам лишь остается

вписать код, для выполнения нашей программы, представим полный код процедуры *Form.Create*:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
Memo.Visible:=false;
Label1.Caption:='';
Label2.Visible:=false;
Label3.Visible:=False;
Label4.Caption:='';
Label5.Caption:='';
Label6.Caption:='';
Button1.Visible:=false;
Button2.Visible:=false;
Button3.Visible:=false;
Button4.Visible:=false;
Button5.Visible:=false;
Button6.Visible:=false;
Edit1.Visible:=False;
Edit2.Visible:=false;
StringGrid1.Visible:=false;
Memo.Visible:=false;
Label8.Visible:=False;
Edit4.Visible:=False;
end;
```

Описание процедуры: Данная процедура работает со свойствами *Visible* и *Caption* компонентов Delphi. Делает все компоненты невидимыми, пока пользователь не вызовет их в соответствующем методе. Кроме тех компонентов *Label*, которые являются компонентами для вывода данных и названия самого метода. Вот как это будет выглядеть, после скрытия всех элементов и запуска программы:



Далее, общей частью для каждого из трех методов будут являться все компоненты кроме компонента Memo, который нам понадобится для вывода результата в Методе Брауна-Робинсона.

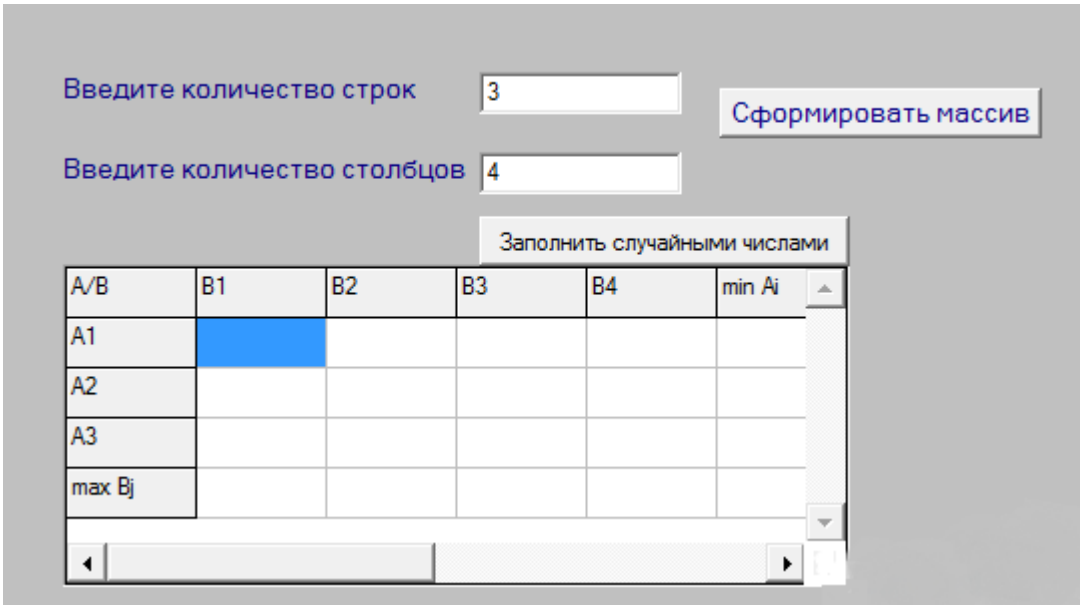
Формирование матрицы в приложении, после ввода размерности пользователем. Данная часть программного кода, также является общей, так при вызове любого из методов нужно будет ввести размерность матрицы, сформировать ее.

Приведем код, который будет формировать матрицу после нажатия пользователем на кнопку (компонент *Button*) «Сформировать массив»:

```
procedure TForm1.Button1Click(Sender: TObject);
 var
 A: array[0..30, 0..30] of Real;   // Платежная матрица игры
 temp:real;
 begin
   s1:=Edit1.Text;
```

```
s2:=Edit2.Text;
n:=StrtoInt(s1);
m:=StrtoInt(s2);
StringGrid1.ColCount:=n+2;
StringGrid1.RowCount:=m+2;

StringGrid1.cells[0,0]:='A/B';
StringGrid1.Cells[n+1,0]:=('min Ai');
StringGrid1.Cells[0,m+1]:=('max Bj');

for i:=1 to n do
begin
StringGrid1.cells[i,0]:=('B'+InttoStr(i));
end;

for j:=1 to m do
begin
StringGrid1.Cells[0,j]:=('A'+InttoStr(j));
end;
```

Данная часть процедуры создает таблицу с размером (N+1)x(M+1). Первой ячейке таблицы присваивает значение (A/B), и начиная с заголовка 1-го столбца/строки до предпоследнего присваивает название и номер игрока(т.к. в у нас условно игроки обозначены как A и B), следовательно после присваивания все будет как представлено на картинке:

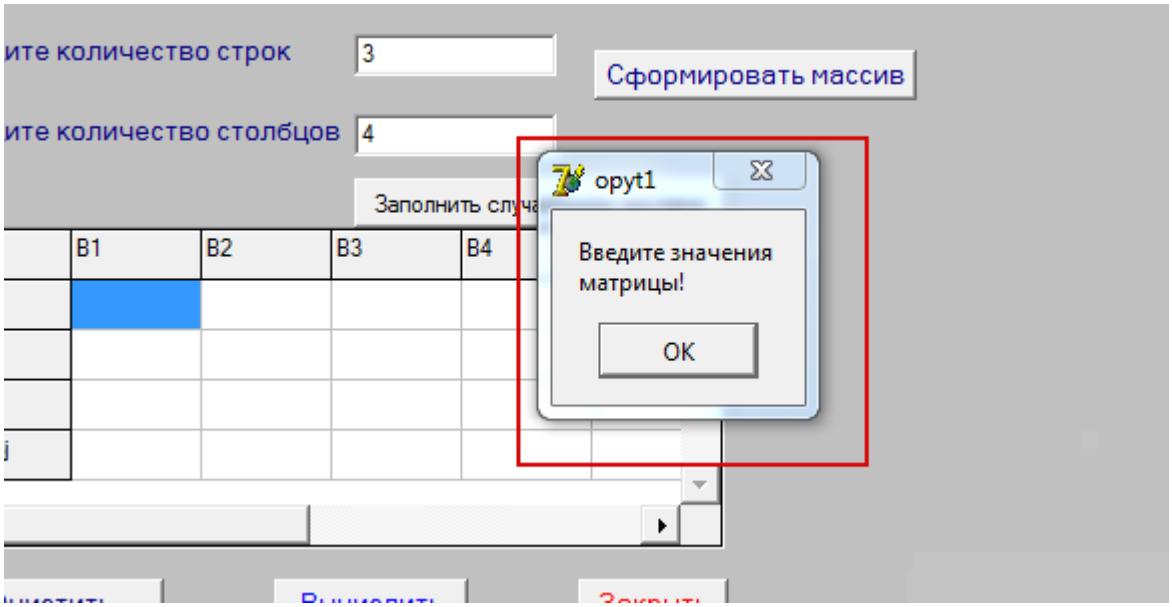


Также последнему столбцу присваивает название min Ai, а последней строке min Bj.

```
  begin
temp := 0;
for i := 1 to n do
for j := 1 to m do
begin

  try

   if String(StringGrid1.Cells[j, i]) = '' then
   begin
    ShowMessage('Введите значения матрицы!');
    Exit;
   end;

   temp  := StrToFloat(String(StringGrid1.Cells[j, i]));

  except
```

```
    ShowMessage('Неправилный формат числа: "' + String(StringGrid1.Cells[j,
i]) + '"');
     Exit;
   end;

   A[i, j] := temp;
 end;
   end;
   end;
```

Вторая часть программы учитывает ошибки, которые может допустить пользователь при работе с программой и после формирования матрицы выдает сообщение о заполнении таблицы. Таковыми ошибками являются не введенная какая-нибудь ячейка или же введена, но некорректно (такими некорректными данными являются строковые выражения, т.е. буквы или символы с клавиатуры). Так вот эта часть программы при таких ошибках выдает соответствующее сообщение. Например, сообщения такого вида:



Также общей для всех методов является кнопка «Заполнить случайными числами», приведем код этой процедуры:

```
procedure TForm1.Button6Click(Sender: TObject);
var i,j: Integer;
begin
Randomize;
  for i := 1 to m do
    begin
    for j:=1 to n do
      StringGrid1.Cells[j,i] := floattostr(Random(10)+1);
    end;
end;
```

Данная процедура заполняет при необходимости случайными числами из отрезка [1 , 11]. Кнопка может понадобиться при демонстрации программы.

В программе также предусмотрены процедура покраски последнего столбца и строки, запрещен ввод данных в эти ячейки, т.к. эти ячейки являются также фиксированными и нужны для вывода данных.

Приведем код данных процедур:

1.
```
procedure TForm1.StringGrid1DrawCell(Sender: TObject;
  ACol, ARow: Integer; Rect: TRect;
  State: TGridDrawState);
begin
StringGrid1.Canvas.Brush.Color:=clBtnFace;
if (ARow=Stringgrid1.Rowcount-1)or (ACol=Stringgrid1.Colcount-1) then
  StringGrid1.Canvas.FillRect(Rect);

StringGrid1.Canvas.TextOut(Rect.Left+2,Rect.Top+2,StringGrid1.Cells[Acol,Arow
]);
end;
```

2. *procedure TForm1.StringGrid1SelectCell(Sender: TObject; ACol, ARow: Integer;var CanSelect: Boolean);*

```
begin
if(ARow=StringGrid1.Rowcount-1)or(ACol=StringGrid1.Colcount-1) then
(Sender as TStringGrid).Options:=(Sender as TStringGrid).Options-[goEditing]
else
(Sender as TStringGrid).Options:=(Sender as TStringGrid).Options+[goEditing];
end;
```

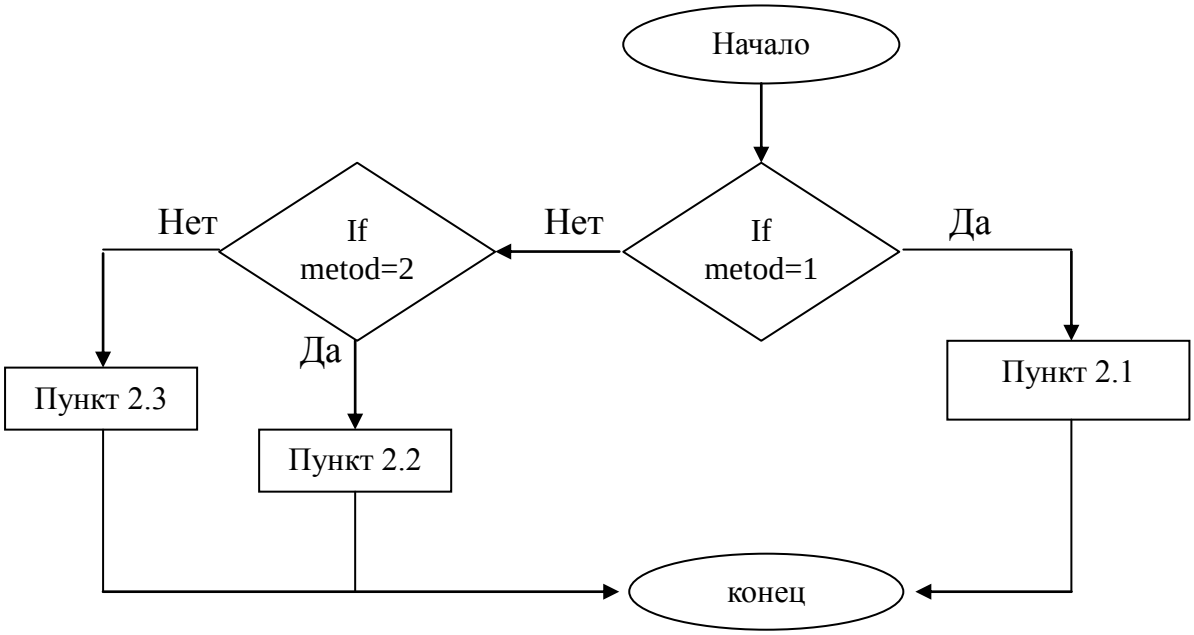
Процедура (1) программно запрещает вводит данные в ячейки, о которых ранее говорилось, а процедура (2) закрашивает таким же цветом, как и фиксированные ячейке нашей таблицы.

Есть еще одна общая кнопка «Очистить», которая нужна, если нужно решить несколько примеров одним методом, да и вообще не помешает. Приведем опять таки код данной кнопки:

```
procedure TForm1.Button4Click(Sender: TObject);
begin
with StringGrid1 do
    for i:=0 to RowCount do
    for j:=0 to ColCount do
    Cells[i,j]:='';
Edit1.Text:='';
Edit2.Text:='';
Label4.Caption:='';
Label5.Caption:='';
Label6.Caption:='';
Memo.Clear;
Memo.Text:='Результат';
end;
```

Процедура выполняет очистку таблицы и компонентов предназначенных для ввода и вывода данных.

Приведем блок-схему основной программы:



2.1. Нахождение верхней и нижней цены игры. Седловая точка.

Для начала рассмотрим пример, которую решим без использования Delphi.

Пример 1.

Матричная игра задана следующей платежной матрицей

$$A = \begin{pmatrix} 3 & 1 & 6 & 7 \\ 8 & 6 & 7 & 8 \\ 11 & 3 & 8 & 9 \end{pmatrix}$$

Найти решение матричной игры, а именно:

найти верхнюю цену игры, нижнюю цену игры, чистую цену игры.

Решение:

1.Найдем в каждой строке платежной матрицы минимальный элемент и запишем его в дополнительный столбец. Затем найдем максимальный элемент дополнительного столбца, это и будет нижняя цена игры.

$$A = \begin{pmatrix} 3 & 1 & 6 & 7 \\ 8 & 6 & 7 & 8 \\ 11 & 3 & 8 & 9 \end{pmatrix} \begin{matrix} 1 \\ 6 \\ 3 \end{matrix}$$

Максимальный элемент дополнительной строки равен 6. Значит нижняя цена игры $\alpha = 6$.

2.Найдем в каждом столбце платежной матрицы максимальный элемент и запишем его в дополнительную строку снизу. Затем найдем минимальный элемент дополнительной строки, это и будет верхняя цена игры.

$$A = \begin{pmatrix} 3 & 1 & 6 & 7 \\ 8 & 6 & 7 & 8 \\ 11 & 3 & 8 & 9 \end{pmatrix}$$

$$\begin{matrix} 11 & 6 & 8 & 9 \end{matrix}$$

Минимальный элемент дополнительной строки равен 6. Следовательно, Верхняя цена игры $\beta = 6$.

3.Сравним нижнюю и верхнюю цены игры, в данной задаче они совпадают, т.е. $\alpha = \beta = 6$. Это значит, что игра имеет решение в так называемых "чистых", минимаксных стратегиях. Это как раз те стратегии для игроков "A" и "B" которые были найдены выше, при поиске нижней и верхней цен игры. То есть, в нашем случае для игрока "A" оптимальной будет стратегия A2, а для игрока "B" - B2. Нетрудно заметить, что элемент платежной матрицы расположенный на пересечении чистых оптимальных стратегий (строка 2, столбец 2) является одновременно минимальным в строке и максимальным в столбце. Такие элементы называются седловыми точками, именно их наличие и определяет существование решения игры в чистых стратегиях, а его значение совпадает с чистой ценой игры или просто ценой игры.

Итак, мы нашли нижнюю и верхнюю цену игры, а также определили имеет ли игра, седловую точку.

Следующим пунктом мы решим эту задачу с помощью Delphi. Как упоминалось ранее, есть общая часть программы и отдельная работа каждого метода. Поиск нижней и верхней цены игры, а также наличие седловой точки осуществляет кнопка «Вычислить» (компонент Button). Но для работы с определенным методом для начала нужно вызвать его выбором метода. При нажатии на кнопку меню «В чистых стратегиях» мы вызываем нужный нам метод. Вот как будет выглядеть его код:

```
procedure TForm1.N16Click(Sender: TObject);
begin
with StringGrid1 do
    for i:=0 to RowCount do
    for j:=0 to ColCount do
    Cells[i,j]:='';
Edit1.Text:='';
Edit2.Text:='';
Label4.Caption:='';
Label5.Caption:='';
Label6.Caption:='';
Memo.Clear;
```

Данная часть программы выполняет задачу очистки таблицы, и компонентов для ввода и вывода. Это нам нужно для того чтобы, при выборе уже запущенного метода, если пользователь вызывает другой метод, а данные были введены, то они автоматически удаляются.

```
Memo.Text:='Результат';
Label1.Caption:='Решение матриччной игры в чистых стратегиях';
Label2.Visible:=true;
Label3.Visible:=true;
Button1.Visible:=true;
```

Button2.Visible:=true;

Button3.Visible:=true;

Button4.Visible:=true;

Button5.Visible:=false;

Button6.Visible:=true;

Edit1.Visible:=true;

Edit2.Visible:=true;

StringGrid1.Visible:=true;

metod:=1;

Label8.Visible:=false;

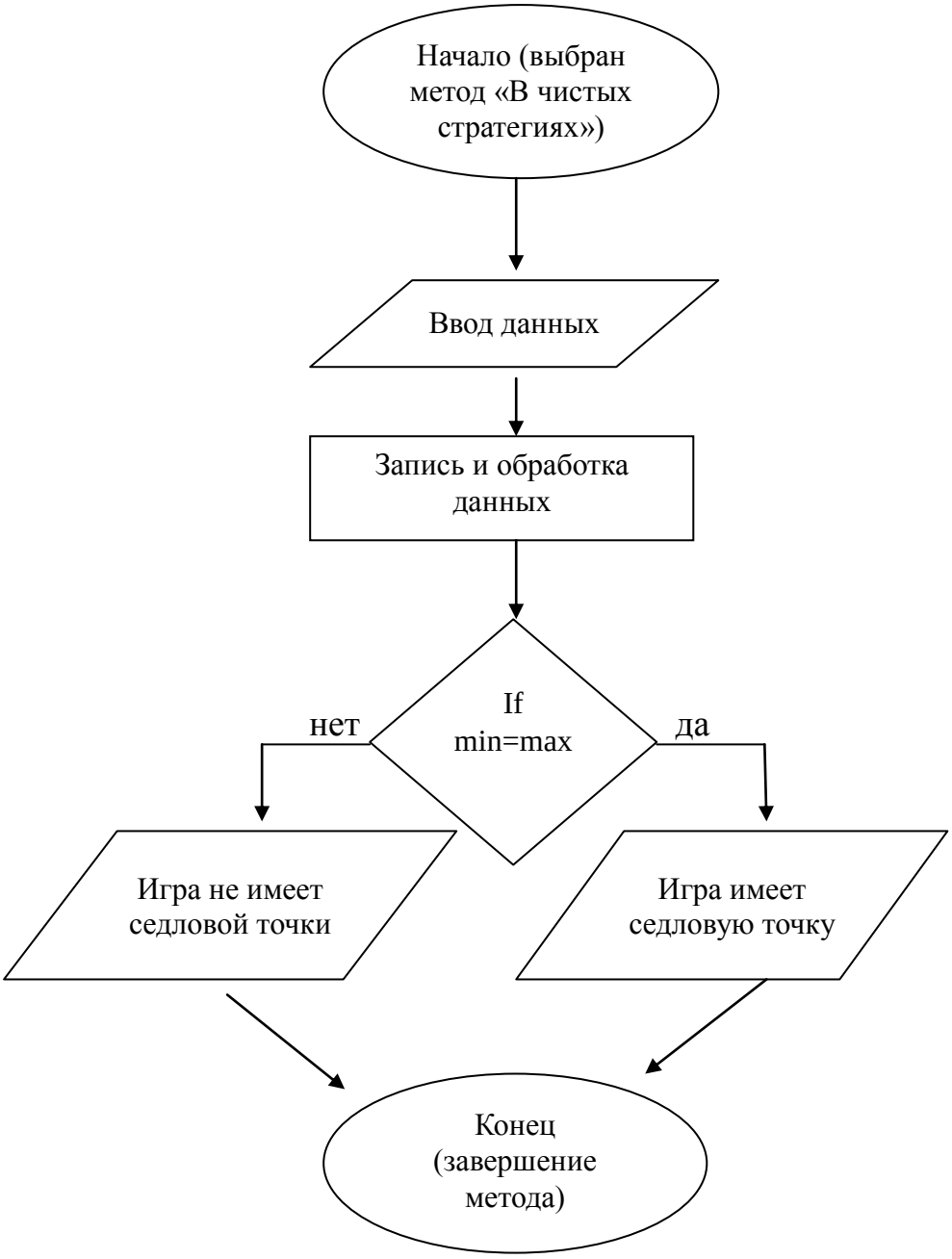
Edit4.Visible:=false;

Memo.Visible:=false;

end;

Вторая часть программного кода вызывает нужные для данного метода решения компоненты, а также присваивает соответствующее название методу. Также переменной *metod* присваивается 1, это нужно для дальнейшей работы приложения, так как у нас одна кнопка на все три метода, то в соответствии от выбора метода будет присваиваться 1, 2 или 3.

Блок-схема метода решения:



Представим полностью исходный код метода:

```
procedure TForm1.Button2Click(Sender: TObject);

var
  mass:array[0..20,0..20] of Real;
  min,max:Real;
  minmas:array[0..20] of Real;
  maxmas:array[0..20] of Real;
```

```
i,j:integer;
```

В описательном разделе процедуры описаны все переменные используемые в данном методе.

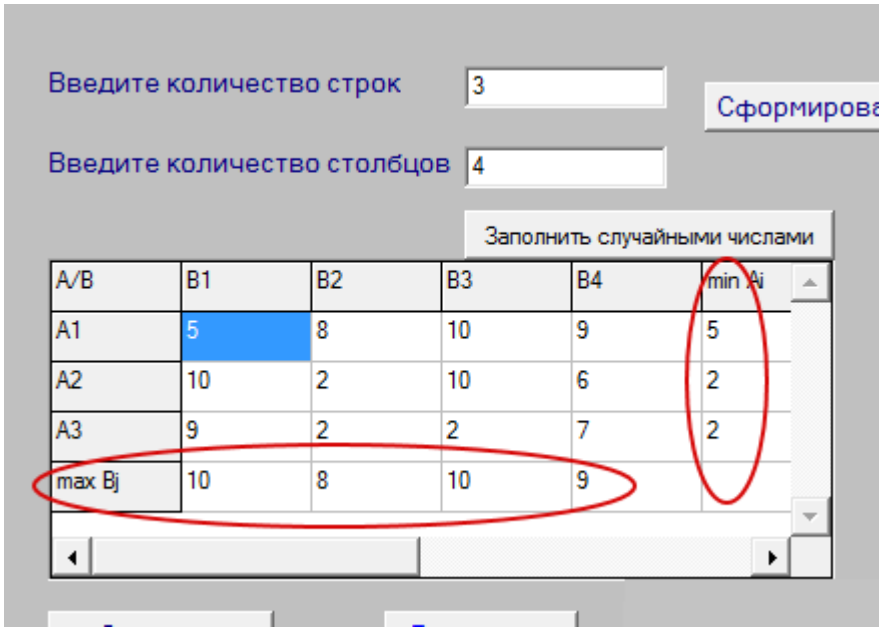
```
begin
for i:=1 to m do
 for j:=1 to n do
   begin
 mass[i,j]:=StrtoInt(StringGrid1.Cells[j,i]);
   end;
```

Данная часть процедуры переводит из таблицы введенной пользователем в массив для дальнейшей его обработки.

```
 for i:=1 to m do
 begin
 min:=mass[i,1];
    for j:=2 to n do
      begin
          if (mass[i,j]<min)  then
            min:=mass[i,j]
      end;
      minmas[i]:=min;
 end;
 for i:=1 to m do
  StringGrid1.Cells[n+1,i]:=FloattoStr(minmas[i]) ;

 for j:=1 to n do
 begin
    max:=mass[1,j];
    for i:=2 to m do
      begin
          if (mass[i,j]>max)  then
```

```
            max:=mass[i,j]
        end;
        maxmas[j]:=max;
    end;
    for j:=1 to n do
     StringGrid1.Cells[j,m+1]:=FloattoStr(maxmas[j]);
```

Следующим пунктом программа выполняет поиск минимального элемента каждой строки и максимального элемента каждого столбца, записывает их в отдельные одномерные массивы , которые следом после окончания цикла записывает в таблицу.




```
    max:=minmas[1];
    imax:=1;
    for i:=2 to m do
    begin
       if (minmas[i]>max) then
        begin
        max:=minmas[i];
        imax:=i;
        end;
```

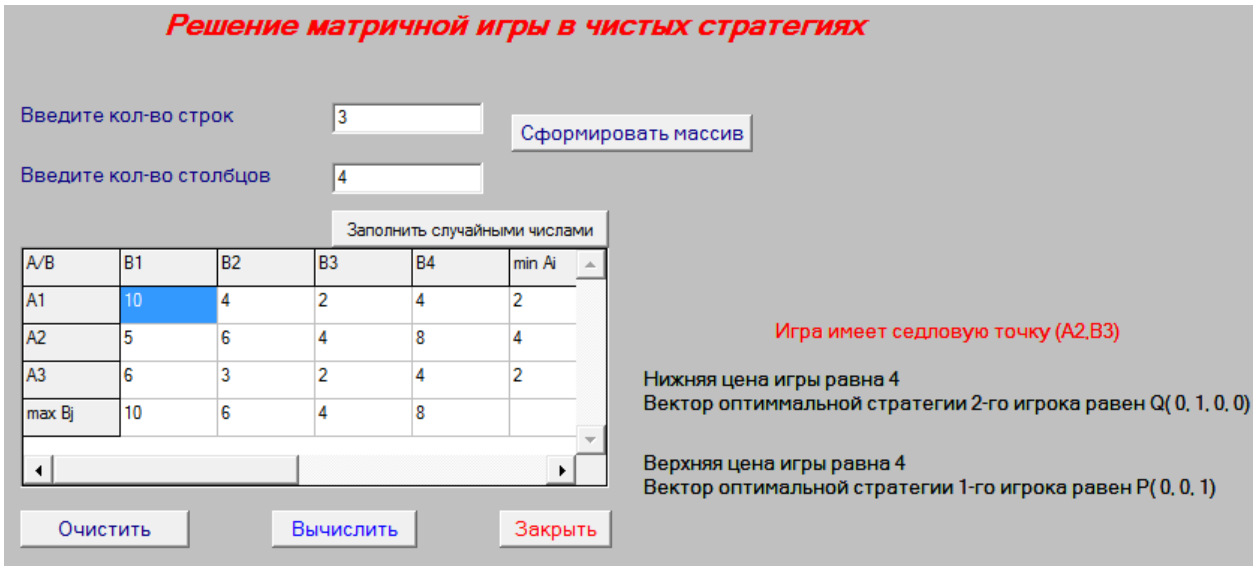
```
end;

min:=maxmas[1];
jmin:=1;
for i:=2 to n do
begin
   if (maxmas[i]<min) then
    begin
    min:=maxmas[i];
    jmin:=i;
    end;
end;
```

Дальнейшая работа программы заключается в поиске максимального элемента из массива содержащего минимальные элементы каждой строки, и в поиске минимального элемента из массива содержащего максимальные элементы столбцов.

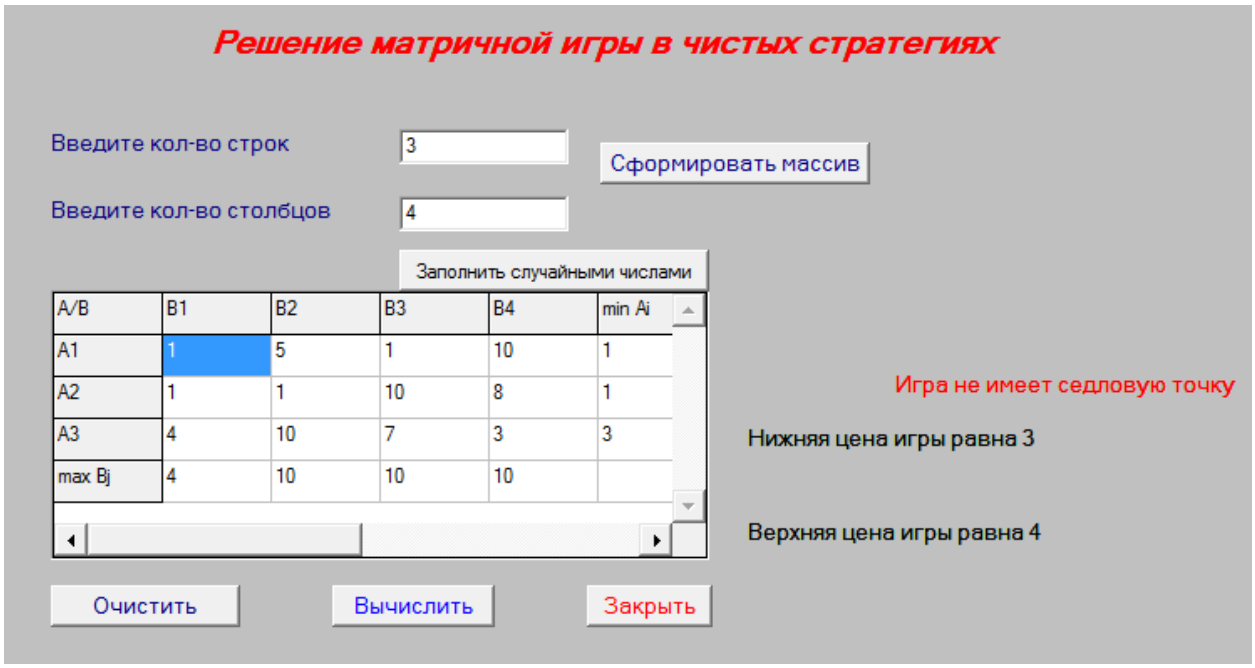
```
if (metod=1) then
begin
 Label5.Caption:='Нижняя цена игры равна '+ FloattoStr(max);
 Label6.Caption:= 'Верхняя цена игры равна ' + FloattoStr(min);
 if (min=max) then
 begin
 Label6.Caption:=Label6.Caption+ #13#10+'Вектор оптимальной стратегии 1-го игрока равен P( ';

 for i:=1 to m do
 begin
  if (i=jmin) then
     maxmas[i] := 1
  else maxmas[i]:=0;
```

```
  Label6.Caption:=Label6.Caption+ FloattoStr(maxmas[i]);
  if (i<>m) then
  Label6.Caption:=Label6.Caption+', ';
 end;
 Label6.Caption:=Label6.Caption+')';
 end;

 if (min=max) then
 begin
 Label5.Caption:=Label5.Caption+ #13#10+'Вектор оптиммальной
стратегии 2-го игрока равен Q( ';

 for i:=1 to n do
 begin
  if (i=imax) then
     minmas[i] := 1
  else minmas[i]:=0;
  Label5.Caption:=Label5.Caption+ FloattoStr(minmas[i]);
  if (i<>n) then
  Label5.Caption:=Label5.Caption+', ';
 end;
 Label5.Caption:=Label5.Caption+')';
 end;

 if (min=max) then
          Label4.Caption:= 'Игра имеет седловую точку
(A'+Inttostr(imax)+',B'+
          InttoStr(jmin)+')' else
          Label4.Caption:= 'Игра не имеет седловую точку' ;
end
```

Данная часть программы выполняет вывод уже обработанных данных начальной стадии программы, в частности кнопки «Вычислить» (компонент *Button*).



Или же, если игра не имеет седловой точки



2.2. Решение матричных игр в смешанных стратегиях

Как и было в предыдущем методе, сначала разберем конкретную задачу, которую решим без применения программы.

Пример. Дана платежная матрица:

$$A = \begin{pmatrix} 4 & 5 \\ 7 & 3 \end{pmatrix}$$

Решить задачу с использованием метода Гаусса.

Решение:

Проверяем, имеет ли платежная матрица седловую точку. Если да, то выписываем решение игры в чистых стратегиях.

Игроки	B_1	B_2	a=min(Ai)
A_1	4	5	4
A_2	7	3	3
b=max(Bi)	7	5	

Находим гарантированный выигрыш, определяемый нижней ценой игры $a = \max(a_i) = 4$, которая указывает на максимальную чистую стратегию A_1. Верхняя цена игры $b = \min(b_j) = 5$. Что свидетельствует об отсутствии седловой точки, так как $a \neq b$, тогда цена игры находится в пределах $4 \leq y \leq 5$. Находим решение игры в смешанных стратегиях. Запишем систему уравнений.

Для игрока I:

$$\begin{cases} 4p_1+7p_2 = y \\ 5p_1+3p_2 = y \end{cases}$$

$p_1+p_2=1$

Для игрока II:

$$\begin{cases} 4q_1+5q_2=y \\ 7q_1+3q_2=y \\ q_1+q_2=1 \end{cases}$$

Решая эти системы методом Гаусса, находим:

$y = 4^3/_5$

$p_1 = {}^4/_5$ (вероятность применения 1-ой стратегии).

$p_2 = {}^1/_5$ (вероятность применения 2-ой стратегии).

Оптимальная смешанная стратегия игрока I:

$P = ({}^4/_5; {}^1/_5)$

$q_1 = {}^2/_5$ (вероятность применения 1-ой стратегии).

$q_2 = {}^3/_5$ (вероятность применения 2-ой стратегии).

Оптимальная смешанная стратегия игрока II: $Q = ({}^2/_5; {}^3/_5)$

Цена игры: $y = 4^3/_5$.

Как и предыдущем методе, и при работе с данным методом, следует вызвать его выбором из меню приложения кнопки «В смешанных стратегиях». При нажатии на эту кнопку выполняется процедура для нашего метода.

Приведем исходный код процедуры вызова метода:

```
procedure TForm1.N17Click(Sender: TObject);
begin
with StringGrid1 do
    for i:=0 to RowCount do
    for j:=0 to ColCount do
    Cells[i,j]:='';
Edit1.Text:='';
Edit2.Text:='';
Label4.Caption:='';
Label5.Caption:='';
```

Label6.Caption:='';

Memo.Clear;

Как и в предыдущем методе данная часть программы выполняет задачу очистки таблицы, и компонентов для ввода и вывода. Которая нам нужно для того чтобы, при выборе уже запущенного метода, если пользователь вызывает другой метод, а данные были введены, то они автоматически удаляются.

Memo.Text:='Результат';

metod:=2;

Label1.Caption:='Решение матричной игры в смешанных стратегиях';

Label2.Visible:=true;

Label3.Visible:=true;

Button1.Visible:=true;

Button2.Visible:=true;

Button3.Visible:=true;

Button4.Visible:=true;

button5.Visible:=false;

Button6.Visible:=false;

Edit1.Visible:=true;

Edit2.Visible:=true;

StringGrid1.Visible:=true;

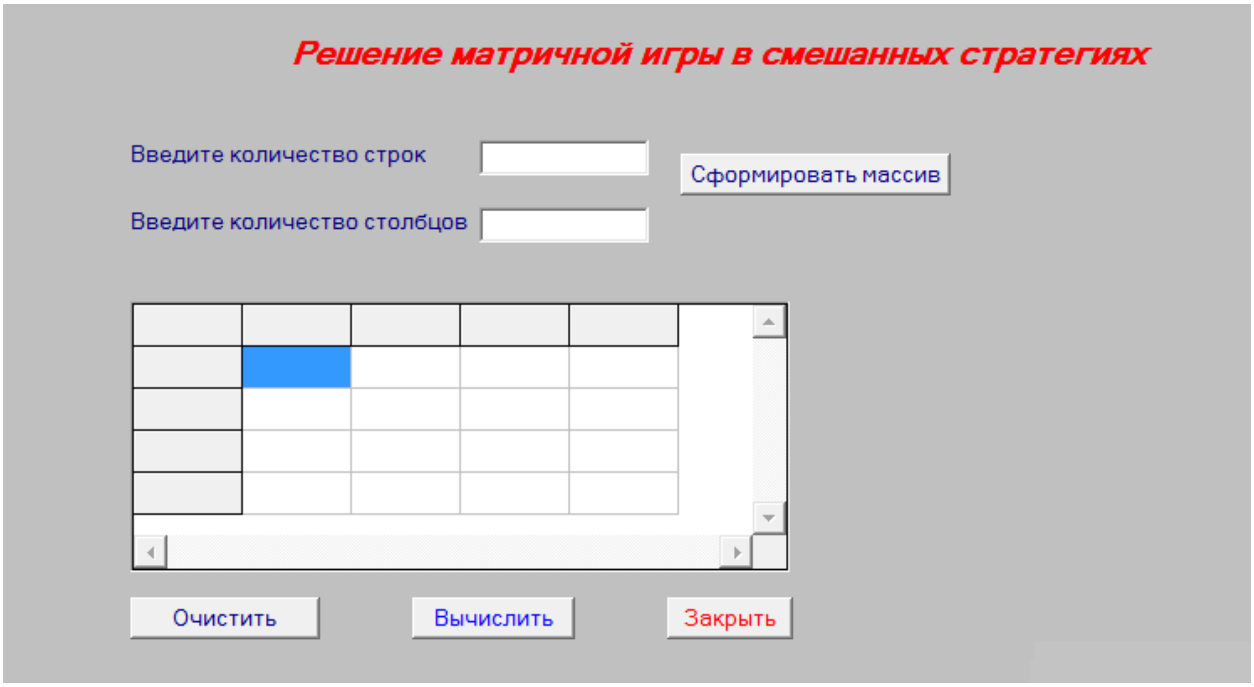
Memo.Visible:=false;

Label8.Visible:=false;

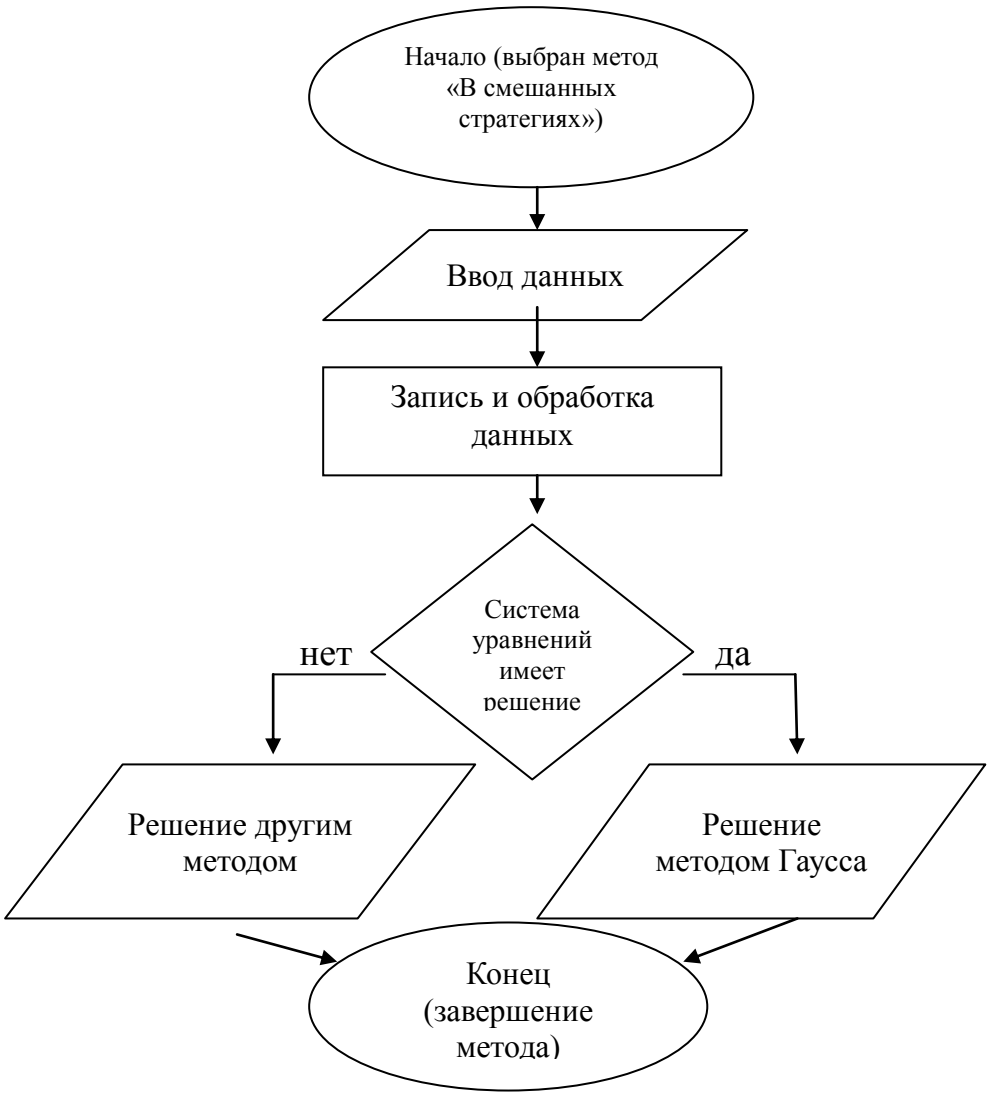
Edit4.Visible:=false;

end;

Вторая часть программного кода вызывает нужные для данного метода решения компоненты, а также присваивает соответствующее название методу. Также переменной *metod* присваивается 2.



Блок-схема метода решения:



Теперь приведем исходный код нашего приложения, который по возможности максимально будем описывать и код функции описанный в разделе *implementation* модуля Delphi:

```
function Gaus(var AIn:Arr; cnt:integer):Integer;
var
 aa,bb,summa:Real;
 k,zi,zj:integer;
begin

 //priyamoi hod
 for zi:=0 to cnt do
 begin
  aa:=AIn[zi,zi];
  for zj:=zi+1 to cnt do
  begin
   bb:=AIn[zj,zi];
   for k:=zi to cnt+1 do
    AIn[zj,k]:=(AIn[zi,k])*bb-(AIn[zj,k])*aa;
  end;
 end;
 //obratnii hod
 begin
 for zi:=cnt-1 downto -1 do
  begin
   summa:=0;
   for zj:=zi+1 to cnt do
    summa:=summa+AIn[zi,zj]*xx[zj];
   summa:=AIn[zi,cnt]-summa;
   if (AIn[zi,zi]=0) then
    Result:= 0;
```

```
      xx[zi]:=summa/AIn[zi,zi];
      end;
      Result:= 1;
    end;
  end;
```

Данная функция нам понадобится для решения системы уравнений методом Гаусса для решения СЛАУ.

```
  procedure TForm1.Button2Click(Sender: TObject);

  var
    mass:array[0..20,0..20] of Real;
    min,max:Real;
    minmas:array[0..20] of Real;
    maxmas:array[0..20] of Real;
    i,j:integer;

  uravneniyadliyaA: Arr;
  uravneniyadliyaB:Arr;
 Begin
  else if (metod=2) then
  begin
     for i:=0 to n do
     begin
       for j:=0 to m-1 do
       begin
         if (i=n) then
         begin
           uravneniyadliyaA[i,j]:=1;
         end
         else
```

```
        uravneniyadliyaA[i,j]:=mass[j+1,i+1];
      end;
    end;
     for i:=0 to n do
    begin
       for j:=m to m+1 do
      begin
            uravneniyadliyaA[i,j]:=1
        else if (j-i=m) then
          uravneniyadliyaA[i,j]:=-1
          else
            uravneniyadliyaA[i,j]:=0;}
        if (i=n) and (j=m) then
          uravneniyadliyaA[i,j]:=0
        else if (i=n) and (j=m+1) then
          uravneniyadliyaA[i,j]:=1
        else if (j=m) then
          uravneniyadliyaA[i,j]:=-1
        else
          uravneniyadliyaA[i,j]:=0;
      end;
    end;
    Gaus(uravneniyadliyaA, m+1);
    Label5.Caption:='Вектор оптимальной стратегии 1-го игрока
равен:';
    Label4.Caption:= 'Цена игры: '+FloattoStr(xx[m]);
    for i:=0 to m-1 do
    begin
      Label5.Caption:= Label5.Caption+#13#10+'P(A'+FloattoStr(i+1)+')=
'+FloattoStr(xx[i]) ;
```

```
    end;
```

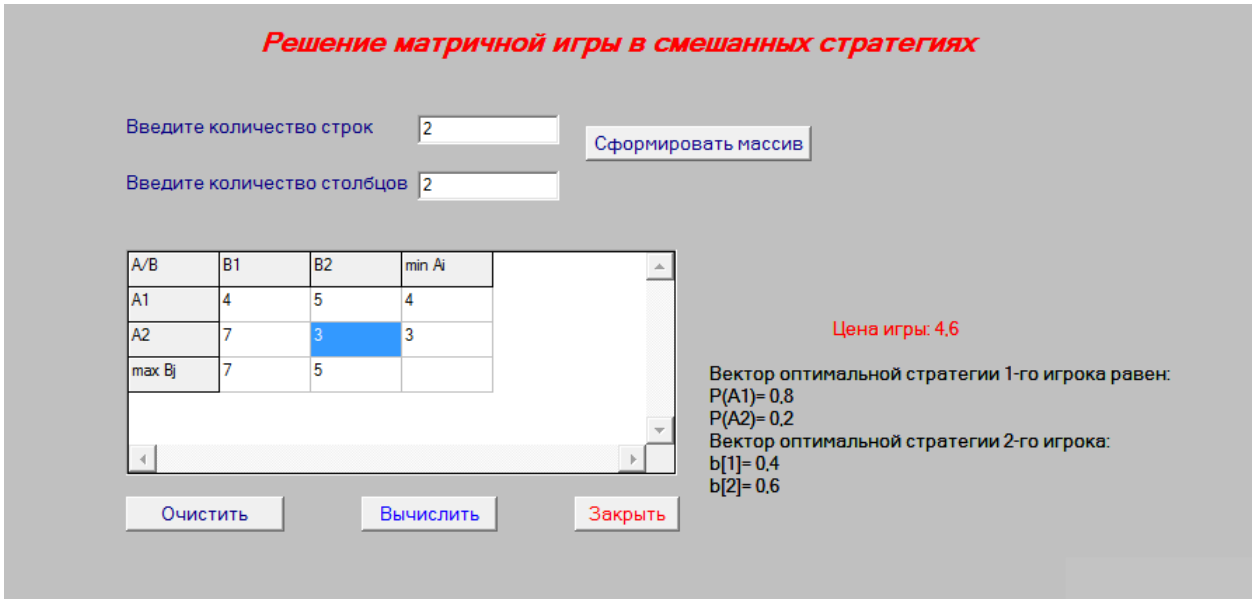
Данная часть программы составляет из введенных данных пользователя систему уравнений для 1-го игрока. Решает ее вызывая функцию *Gaus* из раздела *Implementation* модуля Delphi. После получения решения системы уравнений выводит данные (результат) в компонент *Memo*.

```
  for i:=0 to n do
   begin
    for j:=0 to m+1 do
    begin
     Label5.Caption:= Label5.Caption+FloattoStr(uravneniyadliyaA[i,j])+'
' ;
    end;
    Label5.Caption:=Label5.Caption+#13#10;
   end;

    Label5.Caption:=Label5.Caption+#13#10+#13#10; }

   for i:=0 to m do
  begin
   for j:=0 to n-1 do
   begin
    if (i=m) then
     uravneniyadliyaB[i,j]:=1
    else
     uravneniyadliyaB[i,j]:=mass[i+1,j+1];
   end;
  end;
   for i:=0 to m do
  begin
   for j:=n to n+1 do
```

```
        if (i=m) and (j=n) then
            uravneniyadliyaB[i,j]:=0
          else if (j=n) then
            uravneniyadliyaB[i,j]:=-1
          else if (i=m) and (j=n+1) then
            uravneniyadliyaB[i,j]:=1
          else
            uravneniyadliyaB[i,j]:=0;
       end;
      end;
      Gaus(uravneniyadliyaB, n+1);
       for i:=0 to n-1 do
       begin
          Label5.Caption:=          Label5.Caption+'b['+FloattoStr(i+1)+']=
'+FloattoStr(xx[i])+#13#10 ;
        end;

      for i:=0 to n-1 do
      begin
       for j:=0 to m+n do
       begin
        Label4.Caption:=   Label4.Caption+FloattoStr(uravneniyadliyaB[i,j])+'
' ;
       end;
       Label4.Caption:=Label4.Caption+#13#10;
      end;}

 end;
```

Вторая часть программы аналогично первой составляет систему уравнений для 2-го игрока. Решает ее, также вызывая функцию *Gaus* из

раздела *Implementation* модуля Delphi. После получения решения системы уравнений выводит данные (результат) в компоненты для вывода.



2.3. Решение матричной игры итерационным методом Брауна-Робинсона

Рассмотрим задачу, которую реши методом Брауна-Робинсона.

Задача. Матричная игра задана следующей платежной матрицей:

$$A = \begin{pmatrix} 6 & 1 & 4 \\ 2 & 4 & 2 \\ 4 & 3 & 5 \end{pmatrix}$$

Найти чистую цены игры итерационным методом Брауна-Робинсона.

Решение:

Проверяем, имеет ли платежная матрица седловую точку. Если да, то выписываем решение игры в чистых стратегиях. Считаем, что игрок I выбирает свою стратегию так, чтобы получить максимальный свой выигрыш, а игрок II выбирает свою стратегию так, чтобы минимизировать выигрыш игрока I.

$$\begin{pmatrix} 6 & 1 & 4 \\ 2 & 4 & 2 \\ 4 & 3 & 5 \end{pmatrix} \begin{matrix} 1 \\ 2 \\ 3 \end{matrix}$$

$$6 \quad 4 \quad 5$$

Находим гарантированный выигрыш, определяемый нижней ценой игры $a=\max(a_i) = 3$, которая указывает на максимальную чистую стратегию A_3. Верхняя цена игры $b = \min(b_j) = 4$. Что свидетельствует об отсутствии седловой точки, так как $a \neq b$, тогда цена игры находится в пределах $3<=y<=4$. Находим решение игры в смешанных стратегиях. Объясняется это тем, что игроки не могут объявить противнику свои чистые стратегии: им следует скрывать свои действия. Игру можно решить, если позволить игрокам выбирать свои стратегии случайным образом (смешивать чистые стратегии).

k	i	B_1	B_2	B_3	j	A_1	A_2	A_3	V_{min}	V^{max}	V_{cp}
1	1	6	1	4	2	1	4	3	1	4	$^{5}/_{2}$
2	2	8	5	6	2	2	8	6	$^{5}/_{2}$	4	$^{13}/_{4}$
3	2	10	9	8	3	6	10	11	$^{8}/_{3}$	$^{11}/_{3}$	$^{19}/_{6}$
4	3	14	12	13	2	7	14	14	3	$^{7}/_{2}$	$^{13}/_{4}$
5	2	16	16	15	3	11	16	19	3	$^{19}/_{5}$	$^{17}/_{5}$
6	3	20	19	20	2	12	20	22	$^{19}/_{6}$	$^{11}/_{3}$	$^{41}/_{12}$
7	3	24	22	25	2	13	24	25	$^{22}/_{7}$	$^{25}/_{7}$	$^{47}/_{14}$
8	3	28	25	30	2	14	28	28	$^{25}/_{8}$	$^{7}/_{2}$	$^{53}/_{16}$
9	2	30	29	32	2	15	32	31	$^{29}/_{9}$	$^{32}/_{9}$	$^{61}/_{18}$
10	2	32	33	34	1	21	34	35	$^{16}/_{5}$	$^{7}/_{2}$	$^{67}/_{20}$
11	3	36	36	39	1	27	36	39	$^{36}/_{11}$	$^{39}/_{11}$	$^{75}/_{22}$
12	3	40	39	44	2	28	40	42	$^{13}/_{4}$	$^{7}/_{2}$	$^{27}/_{8}$
13	3	44	42	49	2	29	44	45	$^{42}/_{13}$	$^{45}/_{13}$	$^{87}/_{26}$

14	3	48	45	54	2	30	48	48	$^{45}/_{14}$	$^{24}/_{7}$	$^{93}/_{28}$
15	2	50	49	56	2	31	52	51	$^{49}/_{15}$	$^{52}/_{15}$	$^{101}/_{30}$
16	2	52	53	58	1	37	54	55	$^{13}/_{4}$	$^{55}/_{16}$	$^{107}/_{32}$
17	3	56	56	63	1	43	56	59	$^{56}/_{17}$	$^{59}/_{17}$	$^{115}/_{34}$
18	3	60	59	68	2	44	60	62	$^{59}/_{18}$	$^{31}/_{9}$	$^{121}/_{36}$
19	3	64	62	73	2	45	64	65	$^{62}/_{19}$	$^{65}/_{19}$	$^{127}/_{38}$
20	3	68	65	78	2	46	68	68	$^{13}/_{4}$	$^{17}/_{5}$	$^{133}/_{40}$

где:

k – номер партии.

i – номер стратегии, выбираемой игроком A.

j – номер стратегии, выбираемой игроком B.

B_i– накопленный игроком A выигрыш за k партий, при условии, что в данной партии B выбирает стратегию B_i.

A_j – накопленный игроком B проигрыш за k партий, при условии, что в данной партии A выбирает стратегию A_j.

Vmin – нижняя оценка игры = min (накопленный выигрыш)/k.

Vmax – верхняя оценка игры = max (накопленный проигрыш)/k.

$N_{A1} = 1$

$P(A_1) = 1/20 = {}^{1}/_{20}$

$N_{A2} = 7$

$P(A_2) = 7/20 = {}^{7}/_{20}$

$N_{A3} = 12$

$P(A_3) = 12/20 = {}^{3}/_{5}$

$N_{B1} = 4$

$P(B_4) = 4/20 = {}^{1}/_{5}$

$N_{B2} = 14$

$P(B_4) = 14/20 = {}^{7}/_{10}$

$N_{B3} = 2$

$P(B_4) = 2/20 = {}^{1}/_{10}$

Ответ:

Цена игры, $W = {}^{133}/_{40}$

Вектор оптимальной стратегии 1-го игрока: $p = ({}^{1}/_{20}, {}^{7}/_{20}, {}^{3}/_{5})$

Вектор оптимальной стратегии 2-го игрока: $q = ({}^{1}/_{5}, {}^{7}/_{10}, {}^{1}/_{10})$

Как и предыдущих 2-х методах, при работе с данным методом, следует вызвать его выбором из меню приложения кнопки «Метод Брауна». При нажатии на эту кнопку выполняется процедура для нашего метода. Приведем исходный код процедуры вызова метода:

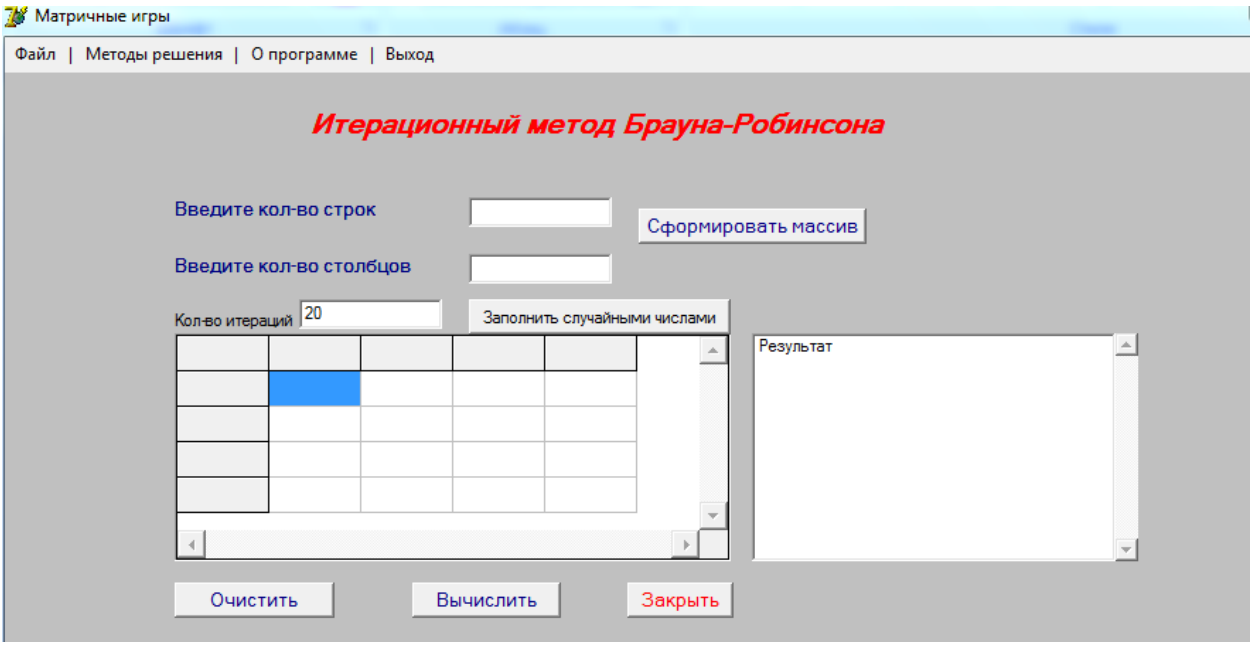
```
procedure TForm1.N18Click(Sender: TObject);
begin
with StringGrid1 do
    for i:=0 to RowCount do
    for j:=0 to ColCount do
    Cells[i,j]:='';
Edit1.Text:='';
Edit2.Text:='';
Label4.Caption:='';
Label5.Caption:='';
Label6.Caption:='';
Memo.Clear;
```

Данная часть программы выполняет также очистку таблицы, и компонентов для ввода и вывода. Это как и упоминалось ранее нам нужно для того чтобы, при выборе уже запущенного метода, если пользователь вызывает другой метод, а данные были введены, то они автоматически удаляются.

```
Memo.Text:='Результат';
metod:=3;
```

```
        Label1.Caption:='Решение матричной игры Итерационным методом';
        Label2.Visible:=true;
        Label3.Visible:=true;
        Label4.Caption:='';
        Label5.Caption:='';
        Label6.Caption:='';
        Button1.Visible:=true;
        Button2.Visible:=false;
        Button3.Visible:=true;
        Button4.Visible:=true;
        Button5.Visible:=true;
        Button6.Visible:=true;
        Edit1.Visible:=true;
        Edit2.Visible:=true;
        StringGrid1.Visible:=true;
        Label8.Visible:=true;
        Edit4.Visible:=true;
        Edit4.Text:='20';
        Memo.Visible:=true;
        Memo.Text:='Результат';
end;
```

Вторая часть программного кода вызывает нужные для данного метода решения компоненты, а также присваивает соответствующее название методу. Также переменной *metod* присваивается 3.



Теперь приведем исходный код данного метода нашего приложения, который по возможности максимально будем описывать и код функции (уже другой, для 3-го метода) описанный также в разделе *implementation* модуля Delphi:

```
// Процедура поиска минимального элемента

function Min1(var Buf; n: Integer): Integer;
var
  r: Real;
  i, Index: Integer;
begin
  Index := 1;
  r := TRealArray(Buf)[1];
  for i := 2 to n do
  begin
    if TRealArray(Buf)[i] < r then
    begin
      Index := i;
      r := TRealArray(Buf)[i];
    end;
```

```
  end;
Result := Index;
end;

// Процедура поиска максимального элемента

function Max1(var Buf; n: Integer): Integer;
var
  r: Real;
  i, Index: Integer;
begin
  Index := 1;
  r := TRealArray(Buf)[1];
  for i := 2 to n do
  begin
    if TRealArray(Buf)[i] > r then
    begin
      Index := i;
      r := TRealArray(Buf)[i];
    end;
  end;
  Result := Index;
end;
```

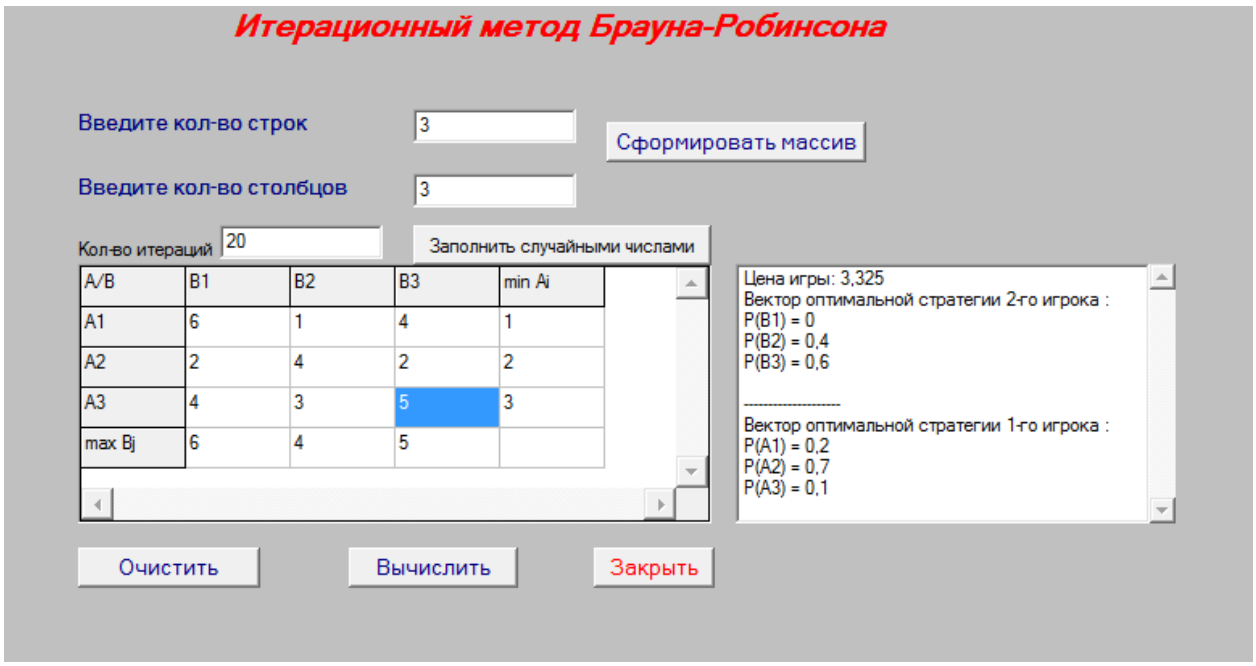
Данные процедуры поиска нам понадобятся для определения случайной стратегии игроков. *Max1* для определения стратегии 1-го игрока, *Min1* соответственно для ответной стратегии 2-го игрока.

```
procedure TForm1.Button2Click(Sender: TObject);
var
  C1, C2, CSUM, temp: Real;
  i, j, k,chisloiteraciy: Integer;
```

```
  X: array[1..30] of Real;
  Y: array[1..30] of Real;
 Z1: array[1..30] of Real;
 Z2: array[1..30] of Real;

Begin
If (metod=3) then
begin
 chisloiteraciy:=StrToInt(Edit4.Text);

 FillChar(X,  SizeOf(X), 0);
 FillChar(Y,  SizeOf(X), 0);
 FillChar(Z1, SizeOf(X), 0);
 FillChar(Z2, SizeOf(X), 0);


 k := 0;
 i := 1;
 repeat

  Inc(k);

  for j := 1 to m do Z1[j] := Z1[j] + A[i, j];

  j := Min1(Z1, m);

  Y[j] := Y[j] + 1;
  C1 := Z1[j] / k;

  for i := 1 to n do Z2[i] := Z2[i] + a[i, j];
```

```
  i := Max1(Z2, n);
  X[i] := X[i] + 1;
  C2 := Z2[i] / K;
  CSUM := (C1 + C2) / 2;

 until k = chisloiteraciy;

 for i := 1 to n do X[i] := X[i] / k;
 for j := 1 to m do Y[j] := Y[j] / k;

 Memo.Clear;
 Memo.Lines.Add('Цена игры: ' + FloatToStr(CSUM));
 Memo.Lines.Add('Вектор оптимальной стратегии 2-го игрока :');

 for j := 1 to n do Memo.Lines.Add('P(B'+ IntToStr(j) + ') = ' +
FloatToStr(X[j]));

 Memo.Lines.Add('');
 Memo.Lines.Add('--------------------');
 Memo.Lines.Add('Вектор оптимальной стратегии 1-го игрока :');

 for i := 1 to m do Memo.Lines.Add('P(A'+ IntToStr(i) + ') = ' +
FloatToStr(Y[i]));
  end;
end;
```

После записи и обработки данных, с нажатием на кнопку «Вычислить» результат вычислений выводится в компоненте вывода *Memo:*


Итерационный метод Брауна-Робинсона
Введите кол-во строк
3
Сформировать массив
Введите кол-во столбцов
3
Кол-во итераций
20
Заполнить случайными числами
A/B
B1
B2
B3
min Ai
A1
6
1
4
1
A2
2
4
2
2
A3
4
3
5
3
max Bj
6
4
5
Цена игры: 3,325
Вектор оптимальной стратегии 2-го игрока :
P(B1) = 0
P(B2) = 0,4
P(B3) = 0,6

Вектор оптимальной стратегии 1-го игрока :
P(A1) = 0,2
P(A2) = 0,7
P(A3) = 0,1
Очистить
Вычислить
Закрыть

Заключение

В ходе данной работы нами были приведены основные понятия теории игр и теории матричных игр, также были рассмотрены матричные игры, а именно:

1) Матричные игры с седловой точкой

2) Матричные игры в смешанных стратегиях

Была написана и описана программа по решению матричных игр с применением среды программирования Delphi на поиск верхней и нижней цены игры, а также по поиску на наличие в игре седловой точки, то есть оптимальной стратегии. Решение матричной игры с заданной платежной матрицей в смешанных стратегиях и итерационным методом Брауна-Робинсона.

Список использованной литературы

1. Блекуэлл, Д. Теория игр и статистических решений / Пер. с англ. И. В. Соловьева. Под ред. Б. А. Севастьянова. С предисл. А. А. Ляпунова. - М. : Изд. иностр. лит., 1958. - 374 с

2. Васин А.А., Морозов В.В. Введение в теорию игр с приложениями к экономике. – М., 2010. - 271 с.

3. Воробьев, Н. Н. Теория игр / Н. Н. Воробьев, д-р физ.-мат. наук. - М. : Знание, 1976. - 64 с.

4. Кузнецова А.В. Экономико-математические методы и модели. – Мн: БГЭУ, 2000. с. 57-96.

5. Нейман Дж., Моргенштерн О. Теория игр и экономическое поведение.- М., Наука, 1970 - 707 с.

6. Теория игр: Л.А. Петросян, Н.А. Зинкевич, Е.В. Шевкопляс – 2-ое изд., перераб. и доп. – СПб.: БХ - Петербург, 2012 – 432 с.