|  |  |
| --- | --- |
| Министерство образования и науки Российской Федерации | |
| федеральное государственное бюджетное образовательное учреждение  высшего образования  «Иркутский государственный университет»  (ФГБОУ ВО «ИГУ») | |
| Институт математики, экономики и информатики | |
| Кафедра информационных технологий | |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
| **ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА** | |
| **БАКАЛАВРА** | |
| 02.03.03 «Математическое обеспечение и администрирование информационных систем» | |
| Общий профиль | |
|  |  |
|  |  |
| ИССЛЕДОВАНИЕ И РАЗРАБОТКА ЛОГИЧЕСКОЙ ИГРЫ ULTIMATE TIC-TAC-TOE | |
|  | |
|  | |
|  | |
|  | |
|  | Студента 4 курса очного отделения |
|  | группы 02441 |
|  | Кунцевич Вадима Сергеевича |
|  |  |
|  |  |
|  | Руководитель: |
|  | Зав. кафедрой, к. т. н., доцент |
|  | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Хмельнов А. Е. |
|  |  |
|  |  |
|  | Допущена к защите |
|  | Зав. кафедрой, к. т. н., доцент |
|  | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Хмельнов А. Д. |
|  |  |
|  |  |
| Иркутск – 2016 |  |
|  |  |

СОДЕРЖАНИЕ

ВВЕДЕНИЕ…………………………………………………………………………..3

Глава 1 Общее описание игры……………………………………………………...4

* 1. Игра с полной информацией………………………………………….4
  2. Правила игры..…………………………………………………………5
  3. Актуальность…………………………………………………………..6
  4. Внешний вид программы……………………………………………...6

Глава 2 Теоретическое описание………………………………………………….10

2.1 Искусственный интеллект…..……………………………………….11

2.2 Алгоритм NegaMax…………………………………………………..12

2.3 Алгоритм Alpha-beta…………………………………………………16

2.4 Эвристика……………………………………………………………..18

Глава 3 Программная реализация…………………………………………………21

3.1 Класс Gameplay………………………………………………………22

3.2 Класс Game……………………………………………………………23

3.3 Класс Options…………………………………………………………24

3.4 Класс Startgame……………………………………………………….25

3.5 Класс Shedule…………………………………………………………25

Глава 4 Анализ результатов………………………………………………………..26

ЗАКЛЮЧЕНИЕ…………………………………………………………………….30

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ………………………………31

ВВЕДЕНИЕ

В данной выпускной квалификационной работе рассмотрена задача разработки и исследования логической игры Ultimate Tic-Tac-Toe. Объектом разработки является искусственный интеллект.

Цель работы – разработка программы, способной противопоставить интеллекту человека свой алгоритм в рамках данной игры. В процессе разработки успешно выполнены следующие задачи:

1. Оценен объем информации, который необходимо обрабатывать приложению;
2. Проведено исследование по оптимизации обработки информации;
3. Разработана эвристика для проведения промежуточной позиционной оценки;
4. Разработан класс для поиска хода и классы для дополнительных задач;
5. Спроектирована архитектура приложения;
6. Произведена реализация на языке программирования C#;
7. Проведены тестирование и анализ.

К программе предъявлены следующие требования:

* Скорость отклика на действие пользователя не более трех секунд;
* Реализовать графический интерфейс пользователя;

В результате программа успешно соревнуется с человеком, а также выполняет ряд опциональных функций.

Глава 1 Общее описание игры

1.1Игра с полной информацией

Игра с полной информацией — термин теории игр, обозначающий логическую игру, в которой для соперников отсутствует элемент неопределённости.

Не вполне строго, но практически можно считать, что игра является игрой с полной информацией, если:

* игроки воздействуют на игровую ситуацию дискретными действиями — ходами, порядок ходов определён правилами и не зависит от таких параметров, как скорость реакции игроков (то есть очередной ход делает тот, кто должен его сделать по правилам, а не тот, кто первым догадался или успел его сделать);
* в любой момент игры все игроки имеют полную информацию о состоянии игры, то есть о позиции и всех возможных ходах любого из игроков.

Если, к тому же, ни в каких аспектах игры (правилах, возможности или очерёдности ходов, определении момента завершения игры или результата) не участвует элемент случайности, такая игра будет ещё и детерминированной.

Для любой детерминированной игры с полной информацией, теоретически, можно просчитать всё дерево возможных ходов игроков и определить последовательность ходов, которая гарантированно приведёт по крайней мере одного из них к выигрышу или ничьей, то есть всегда может быть построен алгоритм выигрыша или сведения игры вничью по крайней мере для одной из сторон. [4]

1.2 Правила игры

Ultimate Tic-Tac-Toe (UTTT) – усложненная версия обычной игры Tic-Tac-Toe. Это игра с нулевой суммой, предназначенная для двоих. В отличие от обычных Крестиков-ноликов игровое поле в UTTT представляет из себя матрицу 3х3, элементами которой являются поля Крестиков-ноликов (см. рис 1). Элементы матрицы называются мини-досками, элементы мини-досок, предназначенные для ходов – клетками. Чтобы победить, игрок должен выиграть три мини-доски в ряд. Каждый игрок может ходить только в пределах той доски, номер которой совпадает с номером клетки, которую отметил предыдущий игрок. Совершающий первый ход может выбрать любую мини-доску. К тому же любую мини-доску можно выбрать, если мини-доска, в пределах которой должен ходить игрок, выбыла из игры.

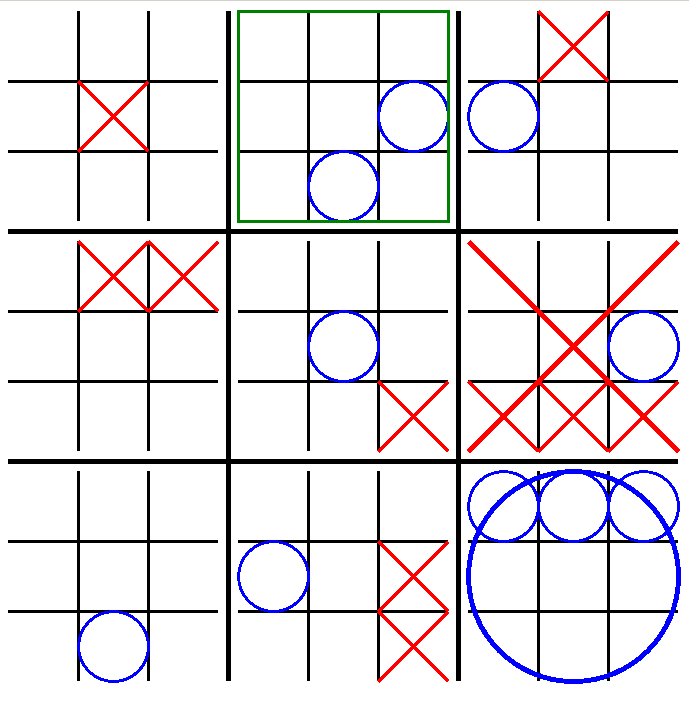


Рис. 1 Игровое поле

1.3 Актуальность разработки

В процессе поиска в сети работающих реализаций данной игры найдено не было. Более того, на всех профильных ресурсах, посвященных логическим играм, намекалось на сложность такой реализации (на одном из порталов даже организован конкурс на лучшую реализацию с денежным призом[2]).

Кроме того, реализацию UTTT можно свободно расширить до более сложных игр, например, шахмат, го, т.к. алгоритм является достаточно универсальным.

1.4 Внешний вид программы

Программа представляет из себя оконное приложение для платформы .Net. При запуске пользователя встречает окно с выбором игрового знака (рис 2).

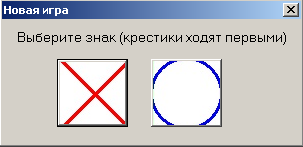


Рис. 2 Стартовое окно

После этого появляется основное окно программы (рис. 3). Оно состоит из игрового поля и строки меню.

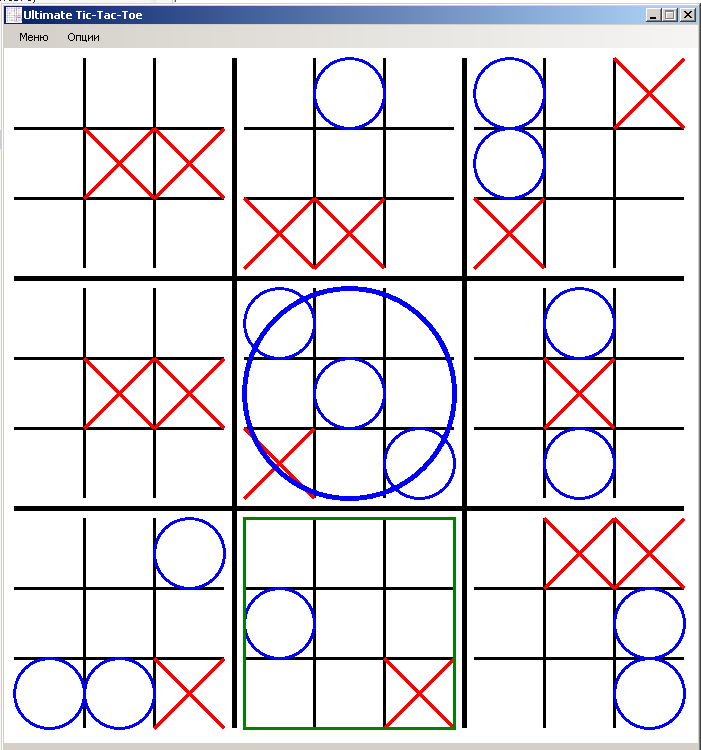


Рис. 3 Главное окно

Строка Меню содержит подразделы Опции, и, собственно, Меню. В них реализованы следующие функции:

* Новая игра – начать новую игру;
* Сохранить – сохранить игру, чтобы продолжить ее позже;
* Загрузить – загрузить сохраненную игру;
* Выход;
* Параметры – показать внутренние коэффициенты, которыми руководствуется программа, такие как максимальная глубина просчета, начало полного перебора и т. д. (см. рис.4);
* Сходить за меня – предоставить компьютеру выбор хода вместо пользователя;
* График роста оценочной функции – наглядно отражает, была ли стратегия пользователя оптимальна в процессе игры (см. рис. 5).

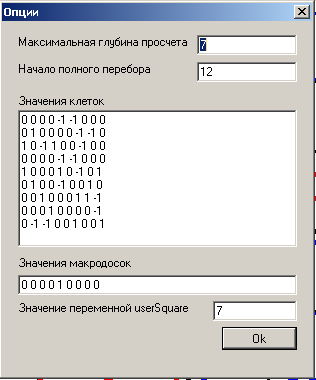


Рис. 4 Параметры игры

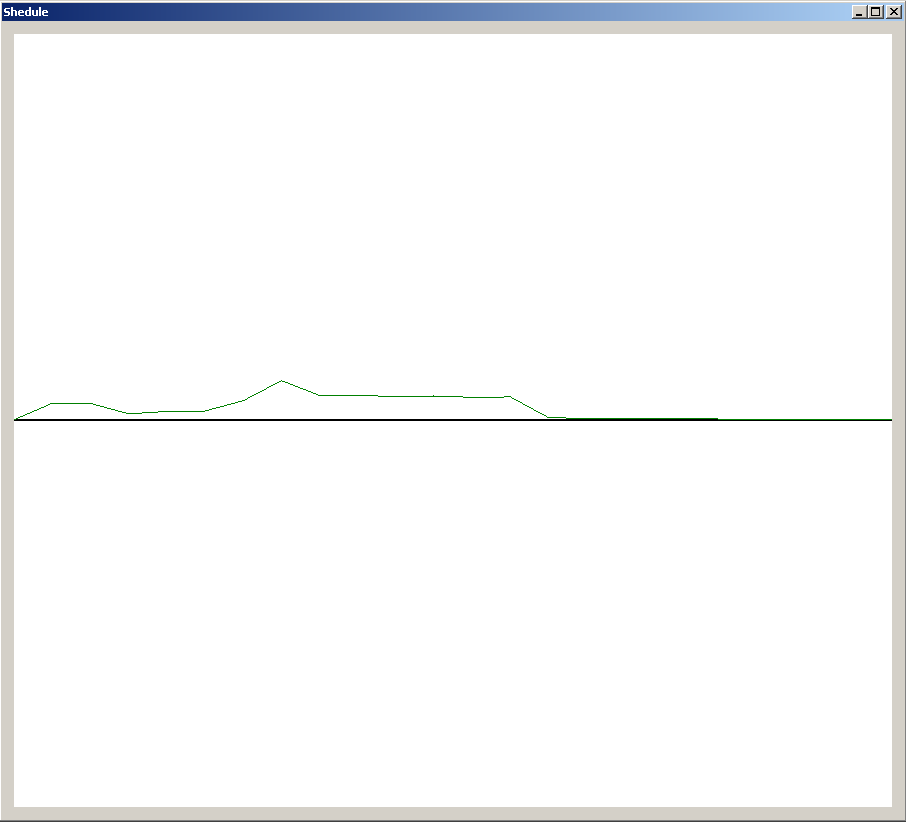


Рис. 5 График оценочной функции

После хода крестиков нолики обязаны ходить в область мини доски, определенной крестиками. Эта область выделяется зеленым квадратом (см. рис. 6). Если область, определенная предыдущим игроком, оказалась вне игры, то текущий игрок волен ходить куда угодно, и зеленый квадрат не появляется (см. рис. 7).

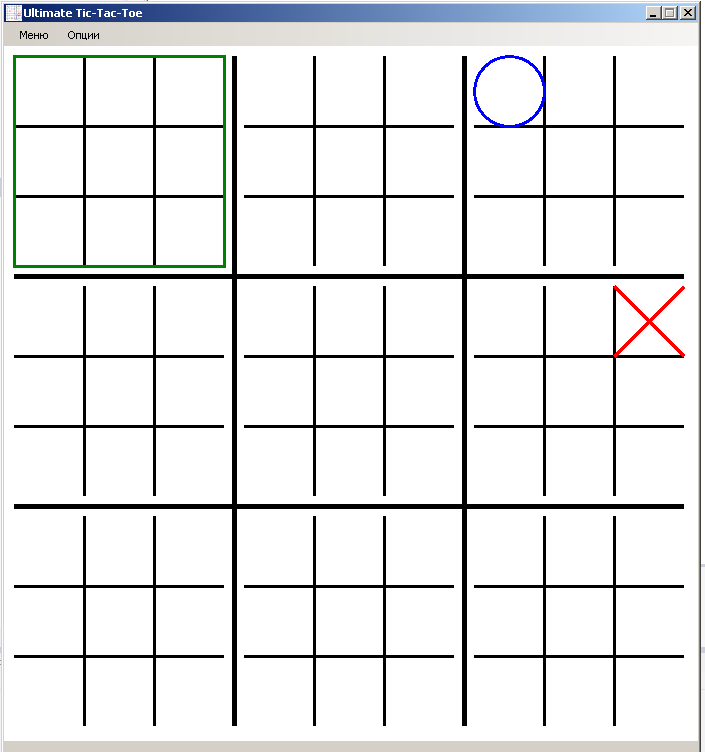


Рис. 6 Начало игры. Нолики ответили на ход крестиков и отправили их в левый верхний угол, о чем программа и сообщает

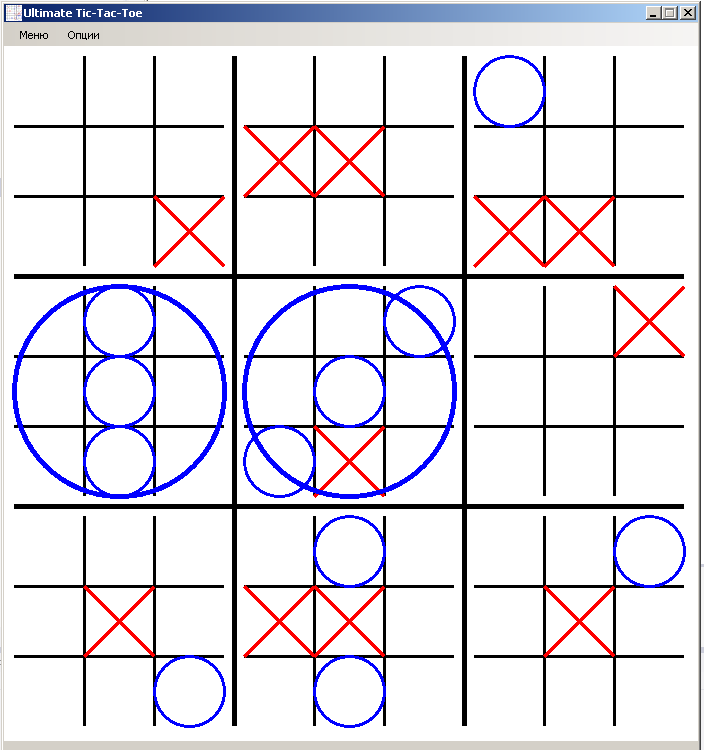


Рис. 7 Продолжение. Нолики отправили игрока на уже захваченное поле, поэтому игрок свободен в выборе хода.

Когда один из игроков одерживает победу, то программа уведомляет об этом окном с соответствующим сообщением (рис. 8).

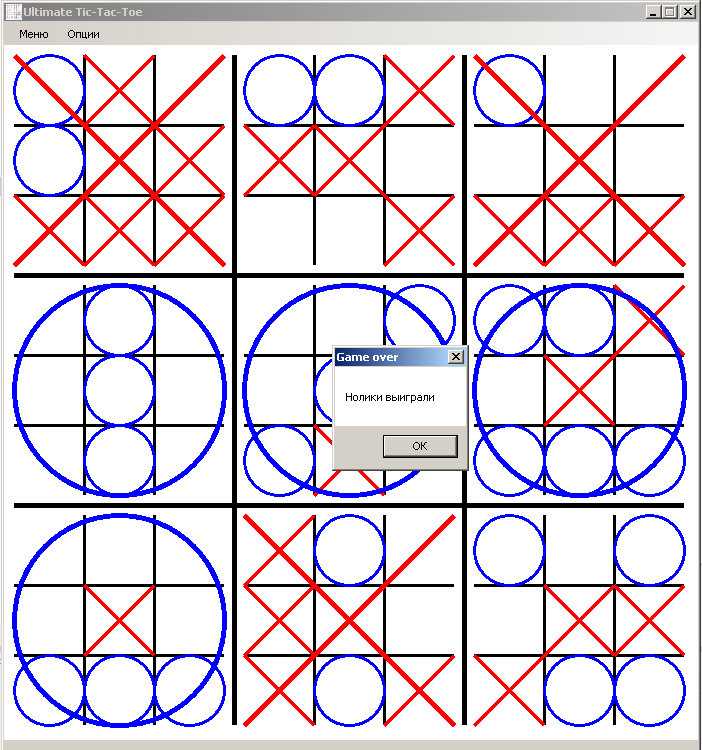


Рис. 8 Нолики снова победили

Глава 2 Теоретическое описание

2.1 Искусственный интеллект

Рассмотрим минимальное количество ходов для выигрыша: игрок должен заполнить по меньшей мере три мини-доски, в каждой из которых по крайней мере по три клетки – в общей сложности девять клеток. Если предположить, что противник никак не участвует в борьбе (что является большим упрощением), то нам нужно девять полуходов для нас и восемь полуходов для противника. Следует обратить внимание, что ничья в данном контексте является наихудшим сценарием, так как нужно принять во внимание каждую из восьмидесяти одной клетки.

Стандартный метод решения задачи поиска хода компьютером – полный перебор возможных позиций из данной позиции и выбор наиболее приемлемой из них исходя из выигрыша либо проигрыша.

Согласно правилам игры, количество вариантов развития партий можно примерно оценить следующим образом:

81\* 9 \* 9 \* … \* 9 \* 8 \* 8 \* … \* 8 \* … \* 2 \* 2 \* … \* 2 = 81 \* 9!8 = =2,4355064378475740285791272177655e+46

Очевидно, что полностью перебрать все позиции со старта игры за оптимальное время невозможно. Однако по ходу партии количество свободных клеток на доске уменьшается, а значит и уменьшается ширина и глубина дерева перебора. Следовательно, начиная с определенного хода, ресурсов ЭВМ хватает для полного перебора за оптимальное время.

До тех же пор, пока полный перебор не является оптимальным решением, применяется алгоритм перебора до максимально возможной глубины –NegaMax, и алгоритм его оптимизации – Alpha-beta (см. Глава 2 п. 2.2 – 2.3), после чего позиции оценивается с помощью эвристики (см. Глава 2 п. 2.4).

|  |  |  |  |
| --- | --- | --- | --- |
| Глубина просчета | Время | Начало полного перебора | Время |
| 60 | больше 5 ч. | 60 | больше 5 ч. |
| 30 | больше 5 ч. | 30 | больше 1 ч. |
| 15 | ≈ 17 мин. | 25 | ≈ 34 мин. |
| 10 | ≈ 36 сек. | 20 | ≈ 19 мин. |
| 8 | ≈ 3 сек | 15 | ≈ 10 сек. |
| 7 | ≈ 1 сек. | 12 | ≈ 1 сек. |

Табл. 1 Соотношение параметров программы и времени выполнения

Оптимальные глубина начального перебора и номер хода для начала полного перебора были установлены экспериментально. Результаты приведены в табл. 1.

Таким образом, оптимальным является перебор на 7 – 8 ходов и оценка позиции с помощью эвристики, а как только до конца игры остается 12 ходов, совершать полный перебор и возвращать максимальную оценку для выигрыша и минимальную для проигрыша.

2.2 Алгоритм NegaMax

Стратегия получения лучшего хода при игре для 2-х игроков была описана еще математиком Клодом Шенноном. Мы должны получить набор позиций-ходов из данной и перебрать их все. Каждую позицию, полученную в результате хода нужно оценить и выбрать наилучшую, т.е. – наилучшее перемещение. Чтобы оценить каждое перемещение, мы рекурсивно вызываем нашу же функцию, только для противоположного знака.  Рекурсивное углубление продолжается до тех пор, пока не найдена конечная позиция в строке (согласно правилам игры) или пока не исчерпана глубина просчета.  В конечных точках вызывается функция, возвращающая статическую оценку позиции.

Некоторые пояснения:

* Generate – функция, генерирующая набор позиций, полученных из данной в результате легальных (согласно правилам игры) ходов;
* Evaluate – статическая оценка позиции;
* MoveList – список ходов.[1]

Блок-схема алгоритма приведена на рис. 9.

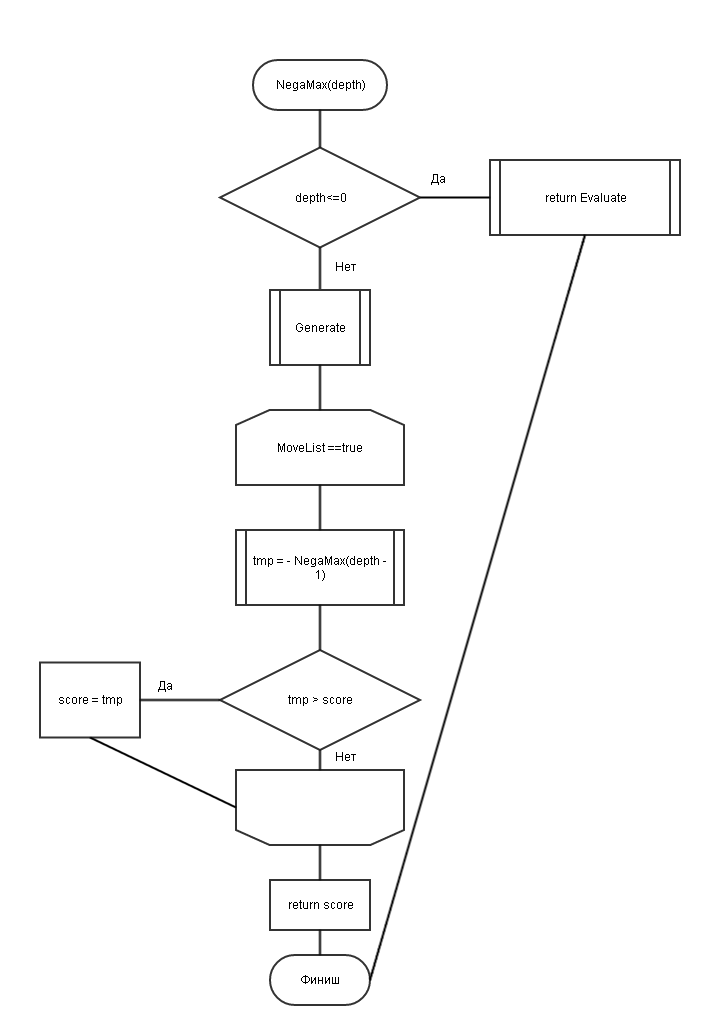


Рис. 9 NegaMax

Этот алгоритм называется NegaMax (см. рис 10). Оценка позиции инвертирована для каждой стороны. Известна еще одна интерпретация данного алгоритма под названием MiniMax. В нем оценка одного знака, но одна сторона максимизирует ее, а другая минимизирует.

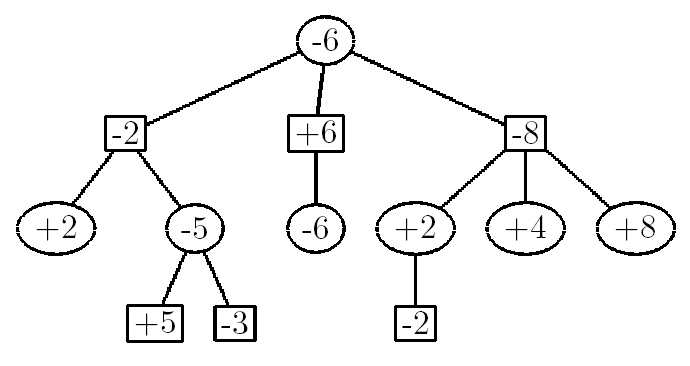
Число позиций, рассмотренных этим алгоритмом, приблизительно равно ND, где:   
N – среднее количество легальных перемещений в каждом узле   
D – глубина перебора

Рис. 10 Работа алгоритма NegaMax

2.3 Алгоритм Alpha-beta

AlphaBeta (рис. 11 – 12) является основным улучшением полного перебора. Суть его в том, что для получения оценки такой же точности, как и при полном переборе, совершенно не обязательно просматривать все варианты.

В рекурсивной функции мы должны ввести две новые переменные – максимум для крестиков и максимум для ноликов. При начальном вызове обе эти величины равны минимально возможному значению: -9999. Если в какой-то позиции, например, для ноликов, мы получили результат, превышающий максимум для ноликов, достигнутый до этого, мы увеличиваем это значение. Представим, что где-то в глубине дерева перебора мы обнаружили, что максимумы крестиков и ноликов сравнялись. Допустим, мы просчитываем позицию для крестиков. Если мы продолжим перебирать в данной позиции, то максимум для крестиков может еще увеличиться, а может остаться прежним, но он уже сравнялся с максимумом ноликов. Это значит, что когда программа вернется в точку (рекурсивно), в которой максимум для ноликов, результат будет отвергнут, т. к. он не превышает максимума для ноликов в этой позиции. Также это значит, что в данной позиции для крестиков мы можем прекратить перебор и вернуть полученный результат досрочно. Дальше считать нет смысла.[1]

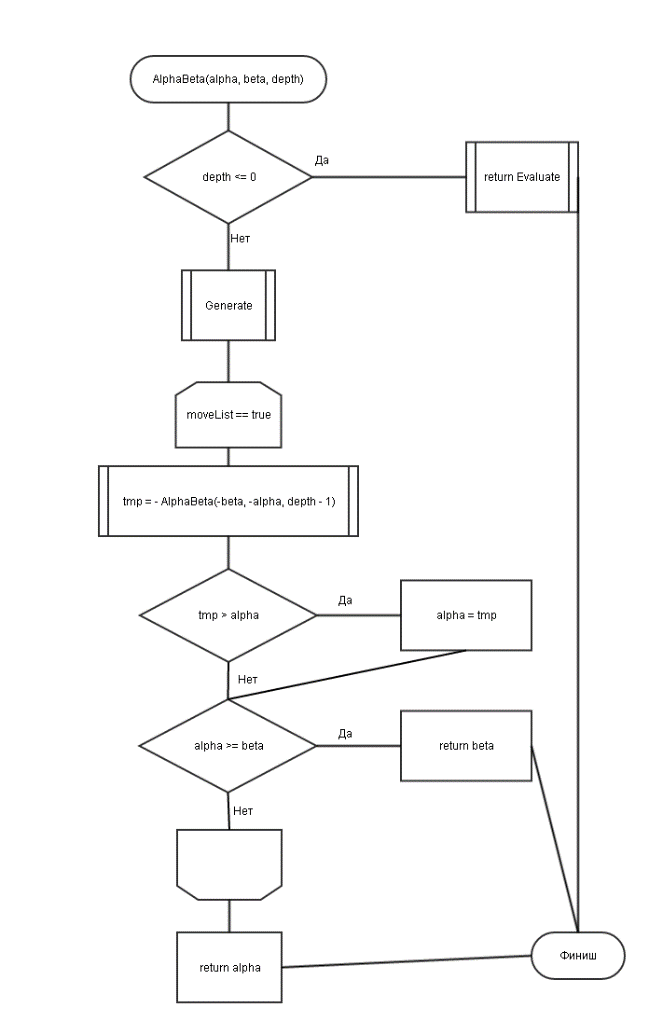


Рис. 11 Alpha-beta

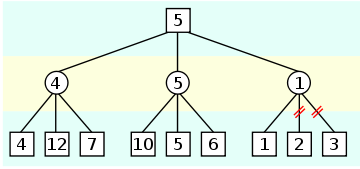
Данный алгоритм при наилучшем порядке ходов исследует квадратный корень из N позиций, где N – количество позиций, исследуемых при полном переборе. Это эквивалентно перебору на глубину в 2 раза больше, за то же время.

Рис. 12 Работа алгоритма Alpha-beta

2.4 Эвристика

В качестве эвристики была принята позиционная оценка, оценивающая потенциал каждой клетки и мини доски. Потенциал – способность клетки (доски) к развитию в выигрыш. Кроме того, оценивается взаимное расположение занятых клеток: взаимодействие клеток, когда до выигрыша остается один ход, увеличивает оценку на две единицы. В дальнейшем оценка мини доски умножается на ее стратегическую оценку на макро доске. Таким образом, если макро доска блокирована со всех сторон противником (рис. 13), то вне зависимости от состояния игры внутри нее ее оценка будет равна нулю.

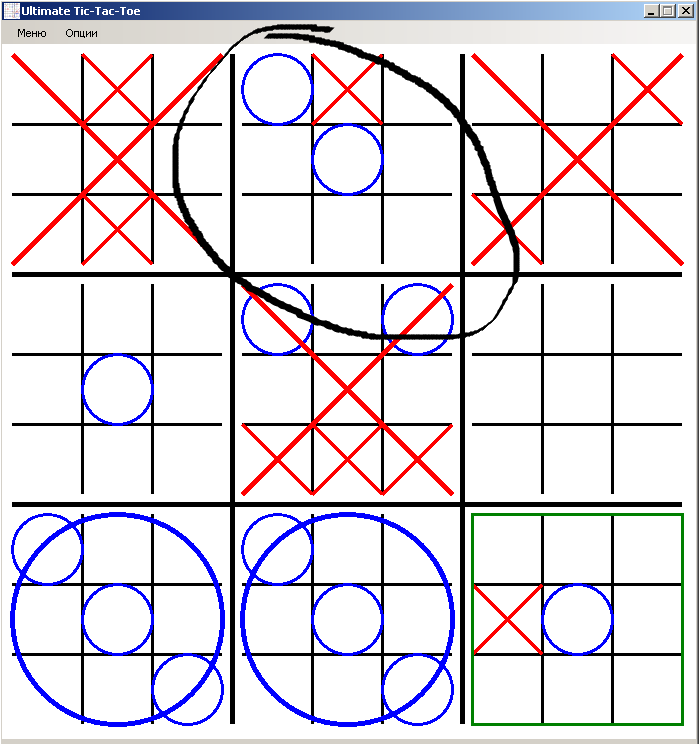


Рис. 13 Отмеченное поле имеет нулевой потенциал, поэтому компьютер не обращает на него внимания

В результате получается следующий алгоритм: оценивается каждая клетка исходя из ее потенциала, оценивается каждая клетка исходя из взаимного расположения одинаковых, полученные оценки умножаются на позиционную оценку макро доски, которая оценивается исходя из такого же определения потенциала. Выигрыш в пределах мини доски оценивается в 50 единиц, что в два раза превышает максимально возможную оценку на мини доске. Такая оценка дана с той целью, чтобы компьютер стремился к закрытию доски, а не просто к тому, чтобы занять как можно больше клеток (рис. 14).

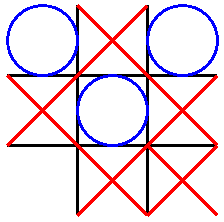


Рис. 14 Бессмысленное занятие клеток вместо быстрого выигрыша

Начальный потенциал клеток приведен в табл. 2 – 3. Блокирование линии, принадлежащей клетке уменьшает потенциал на единицу.

|  |  |  |
| --- | --- | --- |
| 3 | 2 | 3 |
| 2 | 4 | 2 |
| 3 | 2 | 3 |

Табл. 2 Начальный потенциал

|  |  |  |
| --- | --- | --- |
| 2 | 1 | БЛОК |
| 2 | 3 | 1 |
| 2 | 2 | 2 |

Табл. 3 Уменьшение потенциала после блокировки

Таким образом оценочная функция высчитывает позиционную оценку для каждого игрока. Окончательная оценка для игрока есть разность его позиционной оценки и позиционной оценки противника.

Глава 3 Программная реализация

Программа написана на языке C# с использованием стандартных технологий из стека .NET, упоминать о которых отдельно нет необходимости. Реализованные классы (рис. 15):

* Класс Game, отвечающий за хранение информации, ее обработку (поиск хода, оценочная функция);
* Класс Gameplay, отвечающий за взаимодействие пользователя с программой;
* Класс Options, отображающий информацию о параметрах игры;
* Класс Shedule строит график оценочной функции;
* Класс Startgame отображает стартовое диалоговое окно и принимает ответ пользователя.

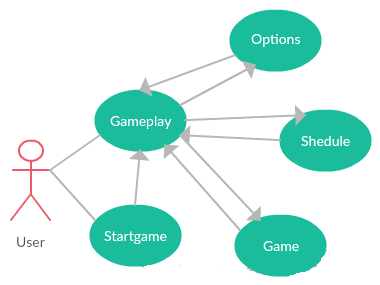


Рис. 15 Взаимодействие классов

3.1 Класс Gameplay

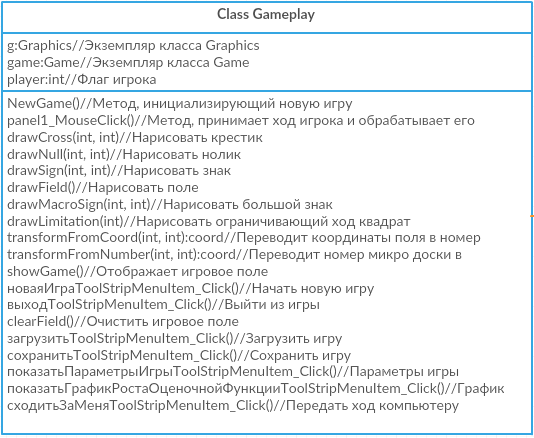
Полный код класса в прил. 1. На UML – диаграмме можно увидеть структуру класса (см. рис. 16).

Рис. 16 Структура класса Gameplay

С класса Gameplay начинается работа программы. Класс вызывает экземпляр класса Startgame и реагирует в соответствии с выбором пользователя, далее полностью строит форму, игровое поле, потом передает управление другим классам в зависимости от действий пользователя. Класс принимает координаты клика мыши по форме и интерпретирует их в номер микро доски и клетки, после чего данная клетка (клетки представлены в виде двумерного массива Game.MicroGame) помечается флагом игрока – 1, если игрок играет за нолики, и -1, если за крестики. После обновления MicroGame происходит проверка, не выиграл ли игрок микро доску, и в случае положительной проверки в соответствующий элемент массива Game.MacroGame ставится флаг игрока. Далее классу Game передается текущая позиция для того, чтобы он высчитал ход компьютера и вернул его, после чего происходит обновление формы и повторение цикла до победы либо ничьи.

3.2 Класс Game

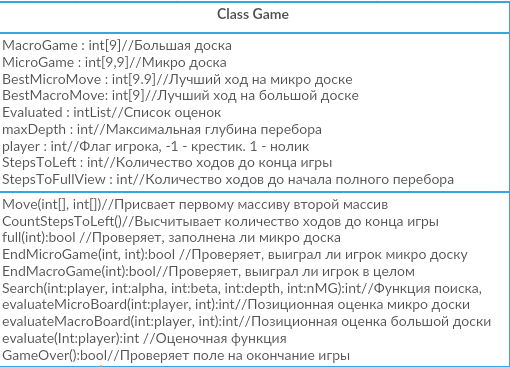
Полный код класса в прил. 2. На UML-диаграмме можно ознакомиться со структурой класса (см. рис. 17).

Рис. 17 UML-диаграмма класса Game

Game является основным классом, т.к. именно в его методе Search реализованы алгоритмы NegaMax и Alpha-beta. Класс принимает от Gameplay номер микро доски для хода, ищет в ее пределах свободную клетку и совершает ход. Далее рекурсивно вызывается тот же метод, но уже с позиции игрока, и процесс повторяется, пока не исчерпается максимальная глубина перебора (поле Game.maxDepth). Вслед за этим оценочная функция Game.evaluate возвращает оценку позиции.

После построения ветви дерева перебора класс переходит к просмотру следующей позиции и к отсечению бесперспективных позиций в соответствии с алгоритмом Alpha-beta.

В случае, если поле Game.StepsToLeft сравнялось с полем Game.StepsToFullView, полю Game.maxDepth присваивается значение Game.StepsToFullView, и таким образом включается полный перебор.

После нахождения лучшего хода он фиксируется в полях Game.BestMicroMove и Game.BestMacroMove, в поле Game.userSquare фиксируется номер микро доски для хода пользователя (в случае свободного хода Game.userSquare = -1), и все эти данные передаются классу Gameplay.

3.3 Класс Options

Options – вспомогательный класс, отвечает за вывод информации о ходе игры по запросу пользователя (см. прил. 3, рис. 18).

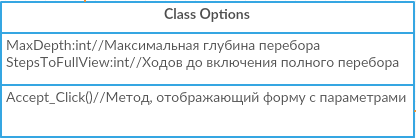


Рис. 18 UML- диаграмма класса Options

После запроса от пользователя Options принимает от класса Game значения полей Game.MicroBoard, Game.Macroboard, Game.userSquare, Game.StepsToFullView и Game.maxDepth, и отображает их на форме. StepsToFullView и maxDepth открыты для редактирования пользователем в целях оптимизации поиска.

3.4 Класс Startgame

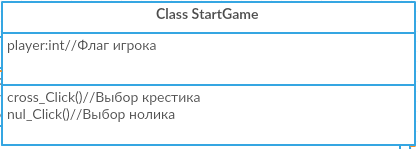
Startgame производит начальную инициализацию объектов и полей, а также взаимодействует с пользователем (см. прил. 4, рис. 19). Класс принимает от пользователя ответ с выбранным знаком и в соответствии с этим инициализирует поле Gameplay.Player. Далее управление передается классу Gameplay.

Рис. 19 UML-диаграмма класса StartGame

3.5 Класс Shedule

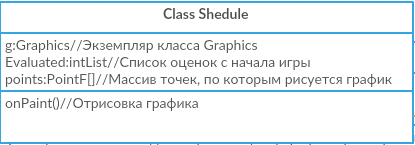
По запросу пользователя Shedule выводит форму с отрисованным графиком оценочной функции (прил. 5, рис. 20).

Рис. 20 UML-диаграмма класса Shedule

Shedule принимает от Gameplay список оценок всех позиций за партию Gameplay.Evaluated и строит график их изменения.

Глава 4 Анализ результатов

Для тестирования программы была собрана фокус-группа, состоящая главным образом из друзей разработчика. Итоги противостояния человека и машины оказались выше всяких ожиданий – компьютеру удается взять верх в абсолютном большинстве партий, вне зависимости от очередности хода (табл. 4). Конечно, зачастую человек не утруждает себя просчетом более чем на четыре полухода, однако такие результаты не могут не обескураживать.

|  |  |  |  |
| --- | --- | --- | --- |
| Игрок | Знак | Процент побед | Процент ничьих |
| player1 | X | 0 | 0 |
| player2 | O | 10 | 0 |
| player3 | X | 6 | 6 |
| player4 | O | 10 | 6 |
| player5 | X | 0 | 10 |
| player6 | O | 15 | 10 |

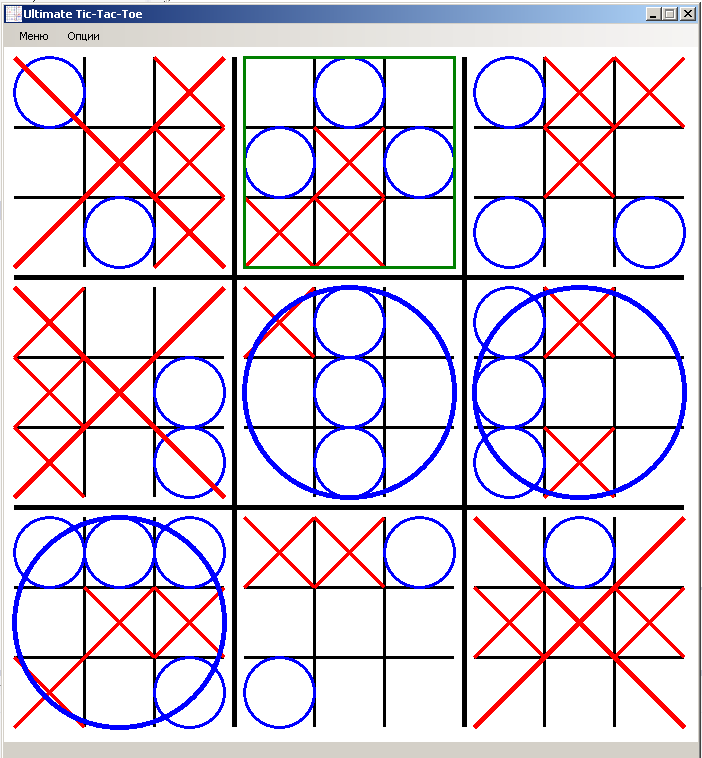
Табл. 4 Разгром человека машиной

Тесты выявили две доминирующие стратегии игроков:

* Пассивная, когда игрок старается избегать серьезных ходов в ожидании благоприятной ситуации;
* Агрессивная – игрок старается переиграть компьютер на его поле, занимая стратегически важные клетки.

Обе стратегии, как видно, в целом ведут к поражению. В случае пассивной игроку не удается дождаться благоприятной ситуации. Компьютер постепенно зажимает его в ловушку, и в итоге игрок попадает в цугцванг (рис. 21).

Рис. 21 Цугцванг – куда бы крестики не сходили, они терпят поражение

В случае агрессивного поведения компьютер использует преимущество в вычислительных ресурсах и берет верх в середине партии, переигрывая человека на уровне стратегии.

Ситуация требует разработки новых стратегий, но ограниченность самой механики игры дает не так много вариантов. И все-таки в процессе дальнейших тестов была выделена еще пара стратегий:

* Планирование, попытки рассчитать серию гамбитов, поймать компьютер на его алчности в отношении повышения значения оценочной функции;
* Поведение, направленное на ухудшение позиции компьютера при любых обстоятельствах, даже в ущерб собственной позиции.

Первая стратегия также не принесла никаких успехов. Если схема, рассчитанная на то, чтобы обмануть компьютер, и существует, то в рамках данной работы ее найти не удалось.

А вот вторая модель поведения несколько увеличила процент побед и почти в семь раз увеличила процент ничьих. Компьютер увязает во множестве бесполезных клеток, и в итоге оба игрока остаются в равновесии (рис. 22).

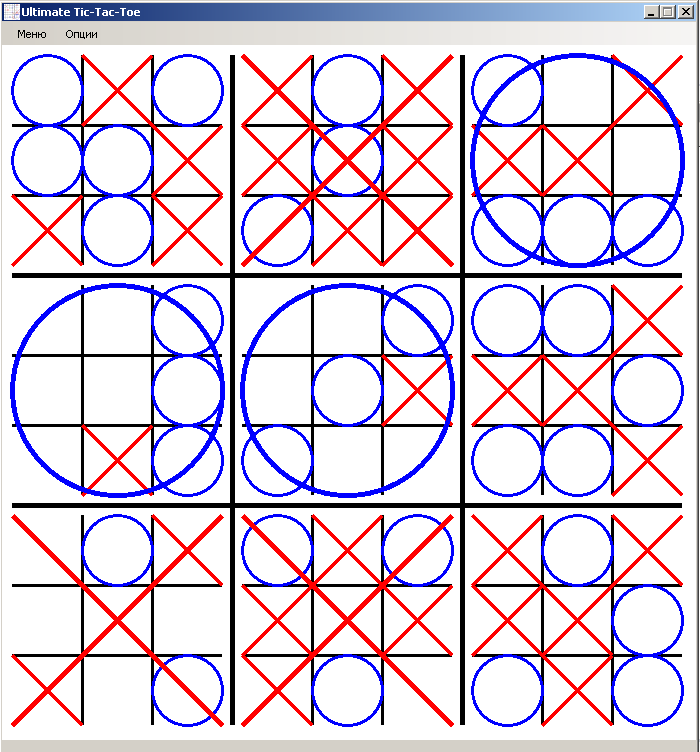


Рис. 22 Ничья – самый частый исход партии при оптимальной стратегии

Из этого можно сделать не очень приятный вывод: Ultimate Tic-Tac-Toe, не смотря на возросшие количественные параметры и усложненные нюансами правила, мало чем отличается от обычных Крестиков-ноликов. Среди найденных стратегий наиболее оптимальной, как и в обычных Крестиках-ноликах, является стратегия порчи позиции противника и ожидания его ошибки, что является «Ничейной смертью»[3].

Таким образом, какие-либо попытки исследования Крестиков-ноликов и любых вариаций этой игры можно считать бессмысленными.

ЗАКЛЮЧЕНИЕ

В данной выпускной квалификационной работе рассмотрена задача разработки и исследования логической игры Ultimate Tic-Tac-Toe. В процессе работы успешно выполнены следующие задачи:

1. Оценен объем информации, который необходимо обрабатывать приложению;
2. Проведено исследование по оптимизации обработки информации;
3. Разработана эвристика для проведения промежуточной позиционной оценки;
4. Разработан класс для поиска хода;
5. Спроектирована архитектура приложения;
6. Произведена реализация на языке программирования C#;
7. Проведены тестирование и анализ.

В результате программа успешно соревнуется с человеком, а также выполняет ряд опциональных функций.

Однако в соревновании с другим компьютером имеющегося потенциала к оптимизации перебора может оказаться недостаточно. В таком случае имеет смысл внедрить дополнительные алгоритмы, например: Alpha-beta с амортизацией откатов, поиск с нулевым окном, поиск с основным вариантом.

Дальнейшее развитие приложения предполагает портирование на мобильные платформы, а также реализация на базе разработанного класса других игр (шахматы, го), так как алгоритм поиска хода универсален для всех игр с открытой стратегией.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Корнилов Е. Н. Программирование шахмат и других логических игр / Е. Н. Корнилов. СПб.: БХВ-Петербург, 2005. 39 – 45 с.: ил.
2. http://theaigames.com/competitions/ultimate-tic-tac-toe
3. https://ru.wikipedia.org/wiki/Ничейная\_смерть
4. https://ru.wikipedia.org/wiki/Игра\_с \_полной \_информацией

ПРИЛОЖЕНИЕ

Приложение 1

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.IO;

namespace UTTTWithGUI

{

public partial class Gameplay : Form

{

private Graphics g;

Game game;

int player = -1;

public Gameplay()

{

InitializeComponent();

//game = new Game();

}

void NewGame()

{

StartGame dialog = new StartGame();

dialog.ShowDialog();

player = dialog.player;

if (dialog.player==1)

{

game = new Game(1);

clearField();

game.Search(-1, -9999, 9999, game.maxDepth, 0);

game.Move(game.BestMicroMove, game.MicroGame);

}

else if (dialog.player == -1)

{

game = new Game(-1);

clearField();

}

}

private void panel1\_MouseClick(object sender, MouseEventArgs e)

{

coord coord = transformFromCoord(e.X, e.Y);//Переводим координаты панели в номер клетки

if((((game.userSquare==-1)||((game.userSquare==coord.X)&&(game.MacroGame[game.userSquare]==0)))&&(game.MicroGame[coord.X,coord.Y]==0)))

{

game.MicroGame[coord.X, coord.Y] = player;//Сходили

if (game.EndMicroGame(player, coord.X))

{

game.MacroGame[coord.X] = player;

game.StepsToLeft -= 9;

}

else game.StepsToLeft -= 1;

if (!game.GameOver())

{

if (game.StepsToLeft > game.StepsToFullView)//Если размер поля приблизился к размеру, обрабатываемому полным перебором, то включаем его

{

label1.Text = game.StepsToLeft.ToString();

}

else

{

game.maxDepth = game.StepsToFullView;

label1.Text = "supermode";

}

game.Search(-player, -9999, 9999, game.maxDepth, coord.Y);

game.Move(game.BestMicroMove, game.MicroGame);

game.Move(game.BestMacroMove, game.MacroGame);

game.Evaluated.AddFirst(game.evaluate(-player) - game.evaluate(player));

game.CountStepsToLeft();

field.Invalidate();

}

else

{

if (game.EndMacroGame(1)) MessageBox.Show("Нолики выиграли", "Game over");

if (game.EndMacroGame(-1)) MessageBox.Show("Крестики выиграли", "Game over");

clearField();

}

}

}

private void onPaint(object sender, PaintEventArgs e)

{

g = e.Graphics;

showGame(g);

}

void drawCross(Graphics g, int X, int Y)

{

Pen cross = new Pen(Color.Red, 3);

g.DrawLine(cross, X, Y, X + 70, Y + 70);

g.DrawLine(cross, X + 70, Y, X, Y + 70);

}//Рисует крестик по координатам левого верхнего угла

void drawNull(Graphics g, int X, int Y)

{

Pen nul = new Pen(Color.Blue, 3);

g.DrawEllipse(nul, X, Y, 70, 70);

}//Рисует нолик по координатам левого верхнего угла

void drawSign(Graphics g, int X, int Y, int sign)

{

if (sign == 1) drawNull(g, X, Y);

if (sign == -1) drawCross(g, X, Y);

}//Рисует знак

void drawField(Graphics g)

{

Pen cells = new Pen(Color.Black, 5);

g.DrawLine(cells, 230, 10, 230, 680);

g.DrawLine(cells, 460, 10, 460, 680);

g.DrawLine(cells, 10, 230, 680, 230);

g.DrawLine(cells, 10, 460, 680, 460);

cells.Width = 3;

for (int j = 0; j < 3; j++)

{

for (int i = 0; i < 3; i++)

{

g.DrawLine(cells, 80 + 230 \* i, 10 + 230 \* j, 80 + 230 \* i, 220 + 230 \* j);

g.DrawLine(cells, 150 + 230 \* i, 10 + 230 \* j, 150 + 230 \* i, 220 + 230 \* j);

g.DrawLine(cells, 10 + 230 \* i, 80 + 230 \* j, 220 + 230 \* i, 80 + 230 \* j);

g.DrawLine(cells, 10 + 230 \* i, 150 + 230 \* j, 220 + 230 \* i, 150 + 230 \* j);

}

}

}//Рисует поле

void drawMacroSign(Graphics g, int nMG, int player)

{

coord coord;

coord = transformFromNumber(nMG, 0);

if (player == 1)

{

Pen nul = new Pen(Color.Blue, 5);

g.DrawEllipse(nul, coord.X, coord.Y, 210, 210);

}

if (player == -1)

{

Pen cross = new Pen(Color.Red, 5);

g.DrawLine(cross, coord.X, coord.Y, coord.X + 210, coord.Y + 210);

g.DrawLine(cross, coord.X + 210, coord.Y, coord.X, coord.Y + 210);

}

}//Рисует знак на большой доске

void drawLimitation(Graphics g, int nMG)

{

Pen lim = new Pen(Color.Green, 3);

coord coord = transformFromNumber(nMG, 0);

g.DrawRectangle(lim, coord.X, coord.Y, 210, 210);

}//Рисует ограничивающий квадрат

coord transformFromCoord(int X, int Y)

{

coord result;

int numI = 0, numJ = 0;

if (((X > 10) && (X < 80)) && ((Y > 10) && (Y < 80))) { numI = 0; numJ = 0; }

if (((X > 80) && (X < 150)) && ((Y > 10) && (Y < 70))) { numI = 0; numJ = 1; }

if (((X > 150) && (X < 220)) && ((Y > 10) && (Y < 70))) { numI = 0; numJ = 2; }

if (((X > 240) && (X < 310)) && ((Y > 10) && (Y < 70))) { numI = 1; numJ = 0; }

if (((X > 310) && (X < 380)) && ((Y > 10) && (Y < 70))) { numI = 1; numJ = 1; }

if (((X > 380) && (X < 450)) && ((Y > 10) && (Y < 70))) { numI = 1; numJ = 2; }

if (((X > 470) && (X < 540)) && ((Y > 10) && (Y < 70))) { numI = 2; numJ = 0; }

if (((X > 540) && (X < 610)) && ((Y > 10) && (Y < 70))) { numI = 2; numJ = 1; }

if (((X > 610) && (X < 680)) && ((Y > 10) && (Y < 70))) { numI = 2; numJ = 2; }

if (((X > 10) && (X < 80)) && ((Y > 80) && (Y < 150))) { numI = 0; numJ = 3; }

if (((X > 80) && (X < 150)) && ((Y > 80) && (Y < 150))) { numI = 0; numJ = 4; }

if (((X > 150) && (X < 220)) && ((Y > 80) && (Y < 150))) { numI = 0; numJ = 5; }

if (((X > 240) && (X < 310)) && ((Y > 80) && (Y < 150))) { numI = 1; numJ = 3; }

if (((X > 310) && (X < 380)) && ((Y > 80) && (Y < 150))) { numI = 1; numJ = 4; }

if (((X > 380) && (X < 450)) && ((Y > 80) && (Y < 150))) { numI = 1; numJ = 5; }

if (((X > 470) && (X < 540)) && ((Y > 80) && (Y < 150))) { numI = 2; numJ = 3; }

if (((X > 540) && (X < 610)) && ((Y > 80) && (Y < 150))) { numI = 2; numJ = 4; }

if (((X > 610) && (X < 680)) && ((Y > 80) && (Y < 150))) { numI = 2; numJ = 5; }

if (((X > 10) && (X < 80)) && ((Y > 150) && (Y < 220))) { numI = 0; numJ = 6; }

if (((X > 80) && (X < 150)) && ((Y > 150) && (Y < 220))) { numI = 0; numJ = 7; }

if (((X > 150) && (X < 220)) && ((Y > 150) && (Y < 220))) { numI = 0; numJ = 8; }

if (((X > 240) && (X < 310)) && ((Y > 150) && (Y < 220))) { numI = 1; numJ = 6; }

if (((X > 310) && (X < 380)) && ((Y > 150) && (Y < 220))) { numI = 1; numJ = 7; }

if (((X > 380) && (X < 450)) && ((Y > 150) && (Y < 220))) { numI = 1; numJ = 8; }

if (((X > 470) && (X < 540)) && ((Y > 150) && (Y < 220))) { numI = 2; numJ = 6; }

if (((X > 540) && (X < 610)) && ((Y > 150) && (Y < 220))) { numI = 2; numJ = 7; }

if (((X > 610) && (X < 680)) && ((Y > 150) && (Y < 220))) { numI = 2; numJ = 8; }

if (((X > 10) && (X < 80)) && ((Y > 240) && (Y < 310))) { numI = 3; numJ = 0; }

if (((X > 80) && (X < 150)) && ((Y > 240) && (Y < 310))) { numI = 3; numJ = 1; }

if (((X > 150) && (X < 220)) && ((Y > 240) && (Y < 310))) { numI = 3; numJ = 2; }

if (((X > 240) && (X < 310)) && ((Y > 240) && (Y < 310))) { numI = 4; numJ = 0; }

if (((X > 310) && (X < 380)) && ((Y > 240) && (Y < 310))) { numI = 4; numJ = 1; }

if (((X > 380) && (X < 450)) && ((Y > 240) && (Y < 310))) { numI = 4; numJ = 2; }

if (((X > 470) && (X < 540)) && ((Y > 240) && (Y < 310))) { numI = 5; numJ = 0; }

if (((X > 540) && (X < 610)) && ((Y > 240) && (Y < 310))) { numI = 5; numJ = 1; }

if (((X > 610) && (X < 680)) && ((Y > 240) && (Y < 310))) { numI = 5; numJ = 2; }

if (((X > 10) && (X < 80)) && ((Y > 310) && (Y < 380))) { numI = 3; numJ = 3; }

if (((X > 80) && (X < 150)) && ((Y > 310) && (Y < 380))) { numI = 3; numJ = 4; }

if (((X > 150) && (X < 220)) && ((Y > 310) && (Y < 380))) { numI = 3; numJ = 5; }

if (((X > 240) && (X < 310)) && ((Y > 310) && (Y < 380))) { numI = 4; numJ = 3; }

if (((X > 310) && (X < 380)) && ((Y > 310) && (Y < 380))) { numI = 4; numJ = 4; }

if (((X > 380) && (X < 450)) && ((Y > 310) && (Y < 380))) { numI = 4; numJ = 5; }

if (((X > 470) && (X < 540)) && ((Y > 310) && (Y < 380))) { numI = 5; numJ = 3; }

if (((X > 540) && (X < 610)) && ((Y > 310) && (Y < 380))) { numI = 5; numJ = 4; }

if (((X > 610) && (X < 680)) && ((Y > 310) && (Y < 380))) { numI = 5; numJ = 5; }

if (((X > 10) && (X < 80)) && ((Y > 380) && (Y < 450))) { numI = 3; numJ = 6; }

if (((X > 80) && (X < 150)) && ((Y > 380) && (Y < 450))) { numI = 3; numJ = 7; }

if (((X > 150) && (X < 220)) && ((Y > 380) && (Y < 450))) { numI = 3; numJ = 8; }

if (((X > 240) && (X < 310)) && ((Y > 380) && (Y < 450))) { numI = 4; numJ = 6; }

if (((X > 310) && (X < 380)) && ((Y > 380) && (Y < 450))) { numI = 4; numJ = 7; }

if (((X > 380) && (X < 450)) && ((Y > 380) && (Y < 450))) { numI = 4; numJ = 8; }

if (((X > 470) && (X < 540)) && ((Y > 380) && (Y < 450))) { numI = 5; numJ = 6; }

if (((X > 540) && (X < 610)) && ((Y > 380) && (Y < 450))) { numI = 5; numJ = 7; }

if (((X > 610) && (X < 680)) && ((Y > 380) && (Y < 450))) { numI = 5; numJ = 8; }

if (((X > 10) && (X < 80)) && ((Y > 470) && (Y < 540))) { numI = 6; numJ = 0; }

if (((X > 80) && (X < 150)) && ((Y > 470) && (Y < 540))) { numI = 6; numJ = 1; }

if (((X > 150) && (X < 220)) && ((Y > 470) && (Y < 540))) { numI = 6; numJ = 2; }

if (((X > 240) && (X < 310)) && ((Y > 470) && (Y < 540))) { numI = 7; numJ = 0; }

if (((X > 310) && (X < 380)) && ((Y > 470) && (Y < 540))) { numI = 7; numJ = 1; }

if (((X > 380) && (X < 450)) && ((Y > 470) && (Y < 540))) { numI = 7; numJ = 2; }

if (((X > 470) && (X < 540)) && ((Y > 470) && (Y < 540))) { numI = 8; numJ = 0; }

if (((X > 540) && (X < 610)) && ((Y > 470) && (Y < 540))) { numI = 8; numJ = 1; }

if (((X > 610) && (X < 680)) && ((Y > 470) && (Y < 540))) { numI = 8; numJ = 2; }

if (((X > 10) && (X < 80)) && ((Y > 540) && (Y < 610))) { numI = 6; numJ = 3; }

if (((X > 80) && (X < 150)) && ((Y > 540) && (Y < 610))) { numI = 6; numJ = 4; }

if (((X > 150) && (X < 220)) && ((Y > 540) && (Y < 610))) { numI = 6; numJ = 5; }

if (((X > 240) && (X < 310)) && ((Y > 540) && (Y < 610))) { numI = 7; numJ = 3; }

if (((X > 310) && (X < 380)) && ((Y > 540) && (Y < 610))) { numI = 7; numJ = 4; }

if (((X > 380) && (X < 450)) && ((Y > 540) && (Y < 610))) { numI = 7; numJ = 5; }

if (((X > 470) && (X < 540)) && ((Y > 540) && (Y < 610))) { numI = 8; numJ = 3; }

if (((X > 540) && (X < 610)) && ((Y > 540) && (Y < 610))) { numI = 8; numJ = 4; }

if (((X > 610) && (X < 680)) && ((Y > 540) && (Y < 610))) { numI = 8; numJ = 5; }

if (((X > 10) && (X < 80)) && ((Y > 610) && (Y < 680))) { numI = 6; numJ = 6; }

if (((X > 80) && (X < 150)) && ((Y > 610) && (Y < 680))) { numI = 6; numJ = 7; }

if (((X > 150) && (X < 220)) && ((Y > 610) && (Y < 680))) { numI = 6; numJ = 8; }

if (((X > 240) && (X < 310)) && ((Y > 610) && (Y < 680))) { numI = 7; numJ = 6; }

if (((X > 310) && (X < 380)) && ((Y > 610) && (Y < 680))) { numI = 7; numJ = 7; }

if (((X > 380) && (X < 450)) && ((Y > 610) && (Y < 680))) { numI = 7; numJ = 8; }

if (((X > 470) && (X < 540)) && ((Y > 610) && (Y < 680))) { numI = 8; numJ = 6; }

if (((X > 540) && (X < 610)) && ((Y > 610) && (Y < 680))) { numI = 8; numJ = 7; }

if (((X > 610) && (X < 680)) && ((Y > 610) && (Y < 680))) { numI = 8; numJ = 8; }

result.X = numI; result.Y = numJ;

return result;

}//Переводит координаты поля в номер квадрата

coord transformFromNumber(int i, int j)

{

int numX = 0, numY = 0;

coord result;

if ((i == 0) && (j == 0)) { numX = 10; numY = 10; }

if ((i == 0) && (j == 1)) { numX = 80; numY = 10; }

if ((i == 0) && (j == 2)) { numX = 150; numY = 10; }

if ((i == 1) && (j == 0)) { numX = 240; numY = 10; }

if ((i == 1) && (j == 1)) { numX = 310; numY = 10; }

if ((i == 1) && (j == 2)) { numX = 380; numY = 10; }

if ((i == 2) && (j == 0)) { numX = 470; numY = 10; }

if ((i == 2) && (j == 1)) { numX = 540; numY = 10; }

if ((i == 2) && (j == 2)) { numX = 610; numY = 10; }

if ((i == 0) && (j == 3)) { numX = 10; numY = 80; }

if ((i == 0) && (j == 4)) { numX = 80; numY = 80; }

if ((i == 0) && (j == 5)) { numX = 150; numY = 80; }

if ((i == 1) && (j == 3)) { numX = 240; numY = 80; }

if ((i == 1) && (j == 4)) { numX = 310; numY = 80; }

if ((i == 1) && (j == 5)) { numX = 380; numY = 80; }

if ((i == 2) && (j == 3)) { numX = 470; numY = 80; }

if ((i == 2) && (j == 4)) { numX = 540; numY = 80; }

if ((i == 2) && (j == 5)) { numX = 610; numY = 80; }

if ((i == 0) && (j == 6)) { numX = 10; numY = 150; }

if ((i == 0) && (j == 7)) { numX = 80; numY = 150; }

if ((i == 0) && (j == 8)) { numX = 150; numY = 150; }

if ((i == 1) && (j == 6)) { numX = 240; numY = 150; }

if ((i == 1) && (j == 7)) { numX = 310; numY = 150; }

if ((i == 1) && (j == 8)) { numX = 380; numY = 150; }

if ((i == 2) && (j == 6)) { numX = 470; numY = 150; }

if ((i == 2) && (j == 7)) { numX = 540; numY = 150; }

if ((i == 2) && (j == 8)) { numX = 610; numY = 150; }

if ((i == 3) && (j == 0)) { numX = 10; numY = 240; }

if ((i == 3) && (j == 1)) { numX = 80; numY = 240; }

if ((i == 3) && (j == 2)) { numX = 150; numY = 240; }

if ((i == 4) && (j == 0)) { numX = 240; numY = 240; }

if ((i == 4) && (j == 1)) { numX = 310; numY = 240; }

if ((i == 4) && (j == 2)) { numX = 380; numY = 240; }

if ((i == 5) && (j == 0)) { numX = 470; numY = 240; }

if ((i == 5) && (j == 1)) { numX = 540; numY = 240; }

if ((i == 5) && (j == 2)) { numX = 610; numY = 240; }

if ((i == 3) && (j == 3)) { numX = 10; numY = 310; }

if ((i == 3) && (j == 4)) { numX = 80; numY = 310; }

if ((i == 3) && (j == 5)) { numX = 150; numY = 310; }

if ((i == 4) && (j == 3)) { numX = 240; numY = 310; }

if ((i == 4) && (j == 4)) { numX = 310; numY = 310; }

if ((i == 4) && (j == 5)) { numX = 380; numY = 310; }

if ((i == 5) && (j == 3)) { numX = 470; numY = 310; }

if ((i == 5) && (j == 4)) { numX = 540; numY = 310; }

if ((i == 5) && (j == 5)) { numX = 610; numY = 310; }

if ((i == 3) && (j == 6)) { numX = 10; numY = 380; }

if ((i == 3) && (j == 7)) { numX = 80; numY = 380; }

if ((i == 3) && (j == 8)) { numX = 150; numY = 380; }

if ((i == 4) && (j == 6)) { numX = 240; numY = 380; }

if ((i == 4) && (j == 7)) { numX = 310; numY = 380; }

if ((i == 4) && (j == 8)) { numX = 380; numY = 380; }

if ((i == 5) && (j == 6)) { numX = 470; numY = 380; }

if ((i == 5) && (j == 7)) { numX = 540; numY = 380; }

if ((i == 5) && (j == 8)) { numX = 610; numY = 380; }

if ((i == 6) && (j == 0)) { numX = 10; numY = 470; }

if ((i == 6) && (j == 1)) { numX = 80; numY = 470; }

if ((i == 6) && (j == 2)) { numX = 150; numY = 470; }

if ((i == 7) && (j == 0)) { numX = 240; numY = 470; }

if ((i == 7) && (j == 1)) { numX = 310; numY = 470; }

if ((i == 7) && (j == 2)) { numX = 380; numY = 470; }

if ((i == 8) && (j == 0)) { numX = 470; numY = 470; }

if ((i == 8) && (j == 1)) { numX = 540; numY = 470; }

if ((i == 8) && (j == 2)) { numX = 610; numY = 470; }

if ((i == 6) && (j == 3)) { numX = 10; numY = 540; }

if ((i == 6) && (j == 4)) { numX = 80; numY = 540; }

if ((i == 6) && (j == 5)) { numX = 150; numY = 540; }

if ((i == 7) && (j == 3)) { numX = 240; numY = 540; }

if ((i == 7) && (j == 4)) { numX = 310; numY = 540; }

if ((i == 7) && (j == 5)) { numX = 380; numY = 540; }

if ((i == 8) && (j == 3)) { numX = 470; numY = 540; }

if ((i == 8) && (j == 4)) { numX = 540; numY = 540; }

if ((i == 8) && (j == 5)) { numX = 610; numY = 540; }

if ((i == 6) && (j == 6)) { numX = 10; numY = 610; }

if ((i == 6) && (j == 7)) { numX = 80; numY = 610; }

if ((i == 6) && (j == 8)) { numX = 150; numY = 610; }

if ((i == 7) && (j == 6)) { numX = 240; numY = 610; }

if ((i == 7) && (j == 7)) { numX = 310; numY = 610; }

if ((i == 7) && (j == 8)) { numX = 380; numY = 610; }

if ((i == 8) && (j == 6)) { numX = 470; numY = 610; }

if ((i == 8) && (j == 7)) { numX = 540; numY = 610; }

if ((i == 8) && (j == 8)) { numX = 610; numY = 610; }

result.X = numX; result.Y = numY;

return result;

}//Переводит номер квадрата в координаты поля

void showGame(Graphics g)

{

coord coord;

drawField(g);

for (int i = 0; i < 9; i++)

{

for (int j = 0; j < 9; j++)

{

coord = transformFromNumber(i, j);

drawSign(g, coord.X, coord.Y, game.MicroGame[i, j]);

}

if (game.EndMicroGame(1, i)) drawMacroSign(g, i, 1);

if (game.EndMicroGame(-1, i)) drawMacroSign(g, i, -1);

if (game.userSquare != -1) drawLimitation(g, game.userSquare);

}

}

private void новаяИграToolStripMenuItem\_Click(object sender, EventArgs e)

{

NewGame();

}

private void выходToolStripMenuItem\_Click(object sender, EventArgs e)

{

Application.Exit();

}

void clearField()

{

for (int i = 0; i < 9; i++)

{

for (int j = 0; j < 9; j++)

{

game.MicroGame[i, j] = 0;

}

game.MacroGame[i] = 0;

}

game.userSquare = -1;

field.Invalidate();

}

private void загрузитьToolStripMenuItem\_Click(object sender, EventArgs e)

{

if (openFileDialog1.ShowDialog() == System.Windows.Forms.DialogResult.OK)

{

StreamReader sr = new

StreamReader(openFileDialog1.FileName);

string line=sr.ReadLine();

string[] numbers = line.Split(' ');

for (int i = 0; i < 9; i++)

{

for (int j = 0; j < 9; j++)

{

game.MicroGame[i, j] = Convert.ToInt32(numbers[i \* 9 + j]);

}

game.MacroGame[i] = Convert.ToInt32(numbers[81 + i]);

}

game.userSquare = Convert.ToInt32(numbers[90]);

game.StepsToLeft = Convert.ToInt32(numbers[91]);

field.Invalidate();

}

}

private void сохранитьToolStripMenuItem\_Click(object sender, EventArgs e)

{

saveFileDialog1.InitialDirectory = "C:\\UTTT";

saveFileDialog1.Filter = "utt files (\*.utt)|\*.utt|All files|\*.\*";

if (saveFileDialog1.ShowDialog() == DialogResult.OK)

{

string fileData="";

for (int i = 0; i < 9; i++)

{

for (int j = 0; j < 9; j++)

{

fileData += game.MicroGame[i, j].ToString() + " ";

}

}

for (int i = 0; i < 9; i++)

{

fileData += game.MacroGame[i].ToString() + " ";

}

fileData += game.userSquare + " ";

fileData += game.StepsToLeft;

string fileName = saveFileDialog1.FileName;

System.IO.File.WriteAllText(fileName, fileData);

}

}

private void Form1\_Load(object sender, EventArgs e)

{

NewGame();

}

private void показатьПараметрыИгрыToolStripMenuItem\_Click(object sender, EventArgs e)

{

Options option = new Options(game.maxDepth, game.StepsToFullView, game.MicroGame, game.MacroGame, game.userSquare);

option.ShowDialog();

game.maxDepth = option.MaxDepth;

game.StepsToFullView = option.StepsToFullView;

option.Hide();

}

private void показатьГрафикРостаОценочнойФункцииToolStripMenuItem\_Click(object sender, EventArgs e)

{

Shedule shedule = new Shedule(game.Evaluated);

shedule.ShowDialog();

}

private void сходитьЗаМеняToolStripMenuItem\_Click(object sender, EventArgs e)

{

if (game.userSquare == -1)//Вызов Search с номером макро доски -1 вызовет ошибку. Искусственно заставим игру выбрать доску другим образом

for (int i = 0; i < 9; i++)

{

if (game.MacroGame[i] != 0)

{

game.Search(player, -9999, 9999, game.maxDepth, i);

game.Move(game.BestMicroMove, game.MicroGame);

game.Move(game.BestMacroMove, game.MacroGame);

game.Evaluated.AddFirst(game.evaluate(-player) - game.evaluate(player));

game.CountStepsToLeft(); break;

}

}

else

{

game.Search(player, -9999, 9999, game.maxDepth, game.userSquare);

game.Move(game.BestMicroMove, game.MicroGame);

game.Move(game.BestMacroMove, game.MacroGame);

game.Evaluated.AddFirst(game.evaluate(-player) - game.evaluate(player));

game.CountStepsToLeft();

}

if (game.userSquare == -1)//Вызов Search с номером макро доски -1 вызовет ошибку. Искусственно заставим игру выбрать доску другим образом

for (int i = 0; i < 9; i++)

{

if (game.MacroGame[i] != 0)

{

game.Search(-player, -9999, 9999, game.maxDepth, i);

game.Move(game.BestMicroMove, game.MicroGame);

game.Move(game.BestMacroMove, game.MacroGame);

game.Evaluated.AddFirst(game.evaluate(-player) - game.evaluate(player));

game.CountStepsToLeft(); break;

}

}

else

{

game.Search(-player, -9999, 9999, game.maxDepth, game.userSquare);

game.Move(game.BestMicroMove, game.MicroGame);

game.Move(game.BestMacroMove, game.MacroGame);

game.Evaluated.AddFirst(game.evaluate(-player) - game.evaluate(player));

game.CountStepsToLeft();

}

field.Invalidate();

}

}

}

Приложение 2

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.IO;

namespace UTTTWithGUI

{

class Game

{

public Game(int player)//Конструктор, инициализирует поля

{

for (int i = 0; i < 9; i++)

{

for (int j = 0; j < 9; j++)

{

MicroGame[i, j] = BestMicroMove[i, j] = 0;

}

MacroGame[i] = 0;

BestMacroMove[i] = 0;

}

userSquare = -1;

StepsToFullView = 12;

StepsToLeft = 81;

maxDepth = 7;

this.player = player;

}

public int unfact(Int64 n)//Обратный факториал

{

Int64 count = 2;

while (n != 1) n = n / count++;

return Convert.ToInt32(count-1);

}

public int[] MacroGame = new int[9];//Большая доска, 0 - в игре, -1 - крестик, 1 - нолик, 2 - ничья

public int[,] MicroGame = new int[9, 9];//Микро доска, 0 - свободна, -1 - крестик, 1 - нолик

public int[,] BestMicroMove = new int[9,9];//Запоминание лучшего хода на микро доске

public int[] BestMacroMove = new int[9];//Запоминание лучшего хода на макро доске

public LinkedList<int> Evaluated = new LinkedList<int>();

public int maxDepth;//Максимальная глубина просчета

public int player;//Игрок, -1 - крестик, 1 - нолик

public int StepsToLeft;//Ходов до окончания игры

public int userSquare;//Номер поля, на которое следует сходить следующему игроку. В случае неограниченного хода принимает значение -1

public int StepsToFullView;//Ходов до включения полного перебора

public void Move(int[,] a, int[,] b)//Присваивает значения второго массива первому

{

for (int i = 0; i < 9; i++)

for (int j = 0; j < 9; j++)

{

b[i, j] = a[i, j];

}

}

public void Move(int[] a, int[] b)

{

for (int j = 0; j < 9; j++)

{

b[j] = a[j];

}

}//Присваивает значения второго массива первому

public void CountStepsToLeft()

{

StepsToLeft = 81;

for (int i = 0; i < 9; i++)

{

for (int j = 0; j < 9; j++)

{

if (MacroGame[i] != 0)

{

StepsToLeft -= 9;

break;

}

else if (MicroGame[i, j] != 0) StepsToLeft -= 1;

}

}

}//Высчитывает поле StepsToLeft

private bool full(int nMG)

{

for (int i = 0; i < 9; i++)

{

if(MicroGame[nMG, i]==0) return false;

}

return true;

}//Проверяет доску на ничью

public bool EndMicroGame(int Player,int nMG)

{

if ((MicroGame[nMG, 0] == Player) && (MicroGame[nMG, 1] == Player) && (MicroGame[nMG, 2] == Player)) return true;

if ((MicroGame[nMG, 3] == Player) && (MicroGame[nMG, 4] == Player) && (MicroGame[nMG, 5] == Player)) return true;

if ((MicroGame[nMG, 6] == Player) && (MicroGame[nMG, 7] == Player) && (MicroGame[nMG, 8] == Player)) return true;

if ((MicroGame[nMG, 0] == Player) && (MicroGame[nMG, 3] == Player) && (MicroGame[nMG, 6] == Player)) return true;

if ((MicroGame[nMG, 1] == Player) && (MicroGame[nMG, 4] == Player) && (MicroGame[nMG, 7] == Player)) return true;

if ((MicroGame[nMG, 2] == Player) && (MicroGame[nMG, 5] == Player) && (MicroGame[nMG, 8] == Player)) return true;

if ((MicroGame[nMG, 0] == Player) && (MicroGame[nMG, 4] == Player) && (MicroGame[nMG, 8] == Player)) return true;

if ((MicroGame[nMG, 2] == Player) && (MicroGame[nMG, 4] == Player) && (MicroGame[nMG, 6] == Player)) return true;

return false;

}//Проверка на окончание игры на микро доске в пользу Player

public bool EndMacroGame(int Player)

{

if ((MacroGame[0] == Player) & (MacroGame[1] == Player) & (MacroGame[2] == Player)) return true;

if ((MacroGame[3] == Player) & (MacroGame[4] == Player) & (MacroGame[5] == Player)) return true;

if ((MacroGame[6] == Player) & (MacroGame[7] == Player) & (MacroGame[8] == Player)) return true;

if ((MacroGame[0] == Player) & (MacroGame[3] == Player) & (MacroGame[6] == Player)) return true;

if ((MacroGame[1] == Player) & (MacroGame[4] == Player) & (MacroGame[7] == Player)) return true;

if ((MacroGame[2] == Player) & (MacroGame[5] == Player) & (MacroGame[8] == Player)) return true;

if ((MacroGame[0] == Player) & (MacroGame[4] == Player) & (MacroGame[8] == Player)) return true;

if ((MacroGame[2] == Player) & (MacroGame[4] == Player) & (MacroGame[6] == Player)) return true;

return false;

}//Проверка на окончание игры на макро доске в пользу Player

public int Search(int player, int alpha, int beta, int depth, int nMG)//Поиск хода

{

int tmp;//Временная переменная, сохранение оценки

bool findMove = false;//Найден ли ход

if (EndMacroGame(-player)) return -9990;//Проверяется, проиграли ли мы. Если да, то возвращается худшая оценка

if (depth == 0) return evaluate(player) - evaluate(-player);//Когда добрались до дна дерева перебора, возвращаем оценку позиции

if (nMG != -1)

if (MacroGame[nMG] == 0)

{

for (int i = 0; i < 9; i++)

{

if (MicroGame[nMG, i] == 0)

{

findMove = true;//Нашли ход

MicroGame[nMG, i] = player;//Сходили

if (EndMicroGame(player, nMG)) MacroGame[nMG] = player; //Если выиграли на микро доске, то отмечаем это

if (full(nMG)) MacroGame[nMG] = 2;//Если ничья на микро доске, то отмечаем это

tmp = -Search(-player, -beta, -alpha, depth - 1, i);//Вызвали рекурсивно то же самое, то бишь сходили противником из текущей позиции

if (tmp > alpha)

{

alpha = tmp;

if (depth == maxDepth)//Когда рекурсия свернулась, отмечаем лучшие ходы и поле, на которое следует ходить сопернику

{

Move(MicroGame, BestMicroMove);

Move(MacroGame, BestMacroMove);

userSquare = MacroGame[i] == 0 ? i : -1;

}

}

MacroGame[nMG] = 0;

MicroGame[nMG, i] = 0;

if (alpha >= beta) break;

}

}

}

else

{

//Если компьютеру дать широкий выбор хода, то количество вариантов резко возрастает и приходится искусственно снижать глубино перебора

for (int i = 0; i < 9; i++)

{

if(MacroGame[i]==0)

for (int j = 0; j < 9; j++)

{

if (MicroGame[i, j] == 0)

{

findMove = true;//Нашли ход

MicroGame[i, j] = player;//Сходили

if (EndMicroGame(player, i)) MacroGame[i] = player; //Если выиграли на микро доске, то отмечаем это

if (full(i)) MacroGame[i] = 2;//Если ничья на микро доске, то отмечаем это

tmp = -Search(-player, -beta, -alpha, depth - 1, j);//Вызвали рекурсивно то же самое, то бишь сходили противником из текущей позиции

if (tmp > alpha)

{

alpha = tmp;

if (depth == maxDepth)//Когда рекурсия свернулась, отмечаем лучшие ходы и поле, на которое следует ходить сопернику

{

Move(MicroGame, BestMicroMove);

Move(MacroGame, BestMacroMove);

userSquare = MacroGame[j] == 0 ? j : -1;

}

}

MacroGame[i] = 0;

MicroGame[i, j] = 0;

if (alpha >= beta) break;

}

}

}

}

if (findMove)

return alpha;

else return 0;

}

private int evaluateMicroBoard(int player, int nMG)

{

int score = 0;

for (int i = 0; i < 9; i++)

{

if (MicroGame[nMG, i] == player)

switch (i)

{

case 0:

if ((MicroGame[nMG, 1] != -player) && (MicroGame[nMG, 2] != -player))

score += 1;

if ((MicroGame[nMG, 3] != -player) && (MicroGame[nMG, 6] != -player))

score += 1;

if ((MicroGame[nMG, 4] != -player) && (MicroGame[nMG, 8] != -player))

score += 1;

if ((MicroGame[nMG, 1] == player) || (MicroGame[nMG, 2] == player))

score += 1;

if ((MicroGame[nMG, 3] == player) || (MicroGame[nMG, 6] == player))

score += 1;

if ((MicroGame[nMG, 4] == player) || (MicroGame[nMG, 8] == player))

score += 1;

break;

case 1:

if ((MicroGame[nMG, 0] != -player) && (MicroGame[nMG, 2] != -player))

score += 1;

if ((MicroGame[nMG, 4] != -player) && (MicroGame[nMG, 7] != -player))

score += 1;

if ((MicroGame[nMG, 0] == player) || (MicroGame[nMG, 2] == player))

score += 1;

if ((MicroGame[nMG, 4] == player) || (MicroGame[nMG, 7] == player))

score += 1;

break;

case 2:

if ((MicroGame[nMG, 0] != -player) && (MicroGame[nMG, 1] != -player))

score += 1;

if ((MicroGame[nMG, 5] != -player) && (MicroGame[nMG, 8] != -player))

score += 1;

if ((MicroGame[nMG, 4] != -player) && (MicroGame[nMG, 6] != -player))

score += 1;

if ((MicroGame[nMG, 0] == player) || (MicroGame[nMG, 1] == player))

score += 1;

if ((MicroGame[nMG, 5] == player) || (MicroGame[nMG, 8] == player))

score += 1;

if ((MicroGame[nMG, 4] == player) || (MicroGame[nMG, 6] == player))

score += 1;

break;

case 3:

if ((MicroGame[nMG, 0] != -player) && (MicroGame[nMG, 6] != -player))

score += 1;

if ((MicroGame[nMG, 4] != -player) && (MicroGame[nMG, 5] != -player))

score += 1;

if ((MicroGame[nMG, 0] == player) || (MicroGame[nMG, 6] == player))

score += 1;

if ((MicroGame[nMG, 4] == player) || (MicroGame[nMG, 5] == player))

score += 1;

break;

case 4:

if ((MicroGame[nMG, 0] != -player) && (MicroGame[nMG, 8] != -player))

score += 1;

if ((MicroGame[nMG, 1] != -player) && (MicroGame[nMG, 7] != -player))

score += 1;

if ((MicroGame[nMG, 2] != -player) && (MicroGame[nMG, 6] != -player))

score += 1;

if ((MicroGame[nMG, 3] != -player) && (MicroGame[nMG, 5] != -player))

score += 1;

if ((MicroGame[nMG, 0] == player) || (MicroGame[nMG, 8] == player))

score += 1;

if ((MicroGame[nMG, 1] == player) || (MicroGame[nMG, 7] == player))

score += 1;

if ((MicroGame[nMG, 2] == player) || (MicroGame[nMG, 6] == player))

score += 1;

if ((MicroGame[nMG, 3] == player) || (MicroGame[nMG, 5] == player))

score += 1;

break;

case 5:

if ((MicroGame[nMG, 2] != -player) && (MicroGame[nMG, 8] != -player))

score += 1;

if ((MicroGame[nMG, 4] != -player) && (MicroGame[nMG, 3] != -player))

score += 1;

if ((MicroGame[nMG, 2] == player) || (MicroGame[nMG, 8] == player))

score += 1;

if ((MicroGame[nMG, 4] == player) || (MicroGame[nMG, 3] == player))

score += 1;

break;

case 6:

if ((MicroGame[nMG, 0] != -player) && (MicroGame[nMG, 3] != -player))

score += 1;

if ((MicroGame[nMG, 4] != -player) && (MicroGame[nMG, 2] != -player))

score += 1;

if ((MicroGame[nMG, 7] != -player) && (MicroGame[nMG, 8] != -player))

score += 1;

if ((MicroGame[nMG, 0] == player) || (MicroGame[nMG, 3] == player))

score += 1;

if ((MicroGame[nMG, 4] == player) || (MicroGame[nMG, 2] == player))

score += 1;

if ((MicroGame[nMG, 7] == player) || (MicroGame[nMG, 8] == player))

score += 1;

break;

case 7:

if ((MicroGame[nMG, 1] != -player) && (MicroGame[nMG, 4] != -player))

score += 1;

if ((MicroGame[nMG, 6] != -player) && (MicroGame[nMG, 8] != -player))

score += 1;

if ((MicroGame[nMG, 1] == player) || (MicroGame[nMG, 4] == player))

score += 1;

if ((MicroGame[nMG, 6] == player) || (MicroGame[nMG, 8] == player))

score += 1;

break;

case 8:

if ((MicroGame[nMG, 0] != -player) && (MicroGame[nMG, 4] != -player))

score += 1;

if ((MicroGame[nMG, 5] != -player) && (MicroGame[nMG, 2] != -player))

score += 1;

if ((MicroGame[nMG, 7] != -player) && (MicroGame[nMG, 6] != -player))

score += 1;

if ((MicroGame[nMG, 0] == player) || (MicroGame[nMG, 4] == player))

score += 1;

if ((MicroGame[nMG, 5] == player) || (MicroGame[nMG, 2] == player))

score += 1;

if ((MicroGame[nMG, 7] == player) || (MicroGame[nMG, 6] == player))

score += 1;

break;

}

}

return score;

}//Оценка мини доски для игрока player

private int evaluateMacroBoard(int player, int nMG)//Оценка макро доски для игрока player

{

int score = 0;

switch (nMG)

{

case 0:

if ((MacroGame[ 1] != -player) && (MacroGame[ 2] != -player))

score += 1;

if ((MacroGame[ 3] != -player) && (MacroGame[ 6] != -player))

score += 1;

if ((MacroGame[ 4] != -player) && (MacroGame[ 8] != -player))

score += 1;

if ((MacroGame[ 1] == player) || (MacroGame[ 2] == player))

score += 1;

if ((MacroGame[ 3] == player) || (MacroGame[ 6] == player))

score += 1;

if ((MacroGame[ 4] == player) || (MacroGame[ 8] == player))

score += 1;

break;

case 1:

if ((MacroGame[ 0] != -player) && (MacroGame[ 2] != -player))

score += 1;

if ((MacroGame[ 4] != -player) && (MacroGame[ 7] != -player))

score += 1;

if ((MacroGame[ 0] == player) || (MacroGame[ 2] == player))

score += 1;

if ((MacroGame[ 4] == player) || (MacroGame[ 7] == player))

score += 1;

break;

case 2:

if ((MacroGame[ 0] != -player) && (MacroGame[ 1] != -player))

score += 1;

if ((MacroGame[ 5] != -player) && (MacroGame[ 8] != -player))

score += 1;

if ((MacroGame[ 4] != -player) && (MacroGame[ 6] != -player))

score += 1;

if ((MacroGame[ 0] == player) || (MacroGame[ 1] == player))

score += 1;

if ((MacroGame[ 5] == player) || (MacroGame[ 8] == player))

score += 1;

if ((MacroGame[ 4] == player) || (MacroGame[ 6] == player))

score += 1;

break;

case 3:

if ((MacroGame[ 0] != -player) && (MacroGame[ 6] != -player))

score += 1;

if ((MacroGame[ 4] != -player) && (MacroGame[ 5] != -player))

score += 1;

if ((MacroGame[ 0] == player) || (MacroGame[ 6] == player))

score += 1;

if ((MacroGame[ 4] == player) || (MacroGame[ 5] == player))

score += 1;

break;

case 4:

if ((MacroGame[ 0] != -player) && (MacroGame[ 8] != -player))

score += 1;

if ((MacroGame[ 1] != -player) && (MacroGame[ 7] != -player))

score += 1;

if ((MacroGame[ 2] != -player) && (MacroGame[ 6] != -player))

score += 1;

if ((MacroGame[ 3] != -player) && (MacroGame[ 5] != -player))

score += 1;

if ((MacroGame[ 0] == player) || (MacroGame[ 8] == player))

score += 1;

if ((MacroGame[ 1] == player) || (MacroGame[ 7] == player))

score += 1;

if ((MacroGame[ 2] == player) || (MacroGame[ 6] == player))

score += 1;

if ((MacroGame[ 3] == player) || (MacroGame[ 5] == player))

score += 1;

break;

case 5:

if ((MacroGame[ 2] != -player) && (MacroGame[ 8] != -player))

score += 1;

if ((MacroGame[ 4] != -player) && (MacroGame[ 3] != -player))

score += 1;

if ((MacroGame[ 2] == player) || (MacroGame[ 8] == player))

score += 1;

if ((MacroGame[ 4] == player) || (MacroGame[ 3] == player))

score += 1;

break;

case 6:

if ((MacroGame[ 0] != -player) && (MacroGame[ 3] != -player))

score += 1;

if ((MacroGame[ 4] != -player) && (MacroGame[ 2] != -player))

score += 1;

if ((MacroGame[ 7] != -player) && (MacroGame[ 8] != -player))

score += 1;

if ((MacroGame[ 0] == player) || (MacroGame[ 3] == player))

score += 1;

if ((MacroGame[ 4] == player) || (MacroGame[ 2] == player))

score += 1;

if ((MacroGame[ 7] == player) || (MacroGame[ 8] == player))

score += 1;

break;

case 7:

if ((MacroGame[ 1] != -player) && (MacroGame[ 4] != -player))

score += 1;

if ((MacroGame[ 6] != -player) && (MacroGame[ 8] != -player))

score += 1;

if ((MacroGame[ 1] == player) || (MacroGame[ 4] == player))

score += 1;

if ((MacroGame[ 6] == player) || (MacroGame[ 8] == player))

score += 1;

break;

case 8:

if ((MacroGame[ 0] != -player) && (MacroGame[ 4] != -player))

score += 1;

if ((MacroGame[ 5] != -player) && (MacroGame[ 2] != -player))

score += 1;

if ((MacroGame[ 7] != -player) && (MacroGame[ 6] != -player))

score += 1;

if ((MacroGame[ 0] == player) || (MacroGame[ 4] == player))

score += 1;

if ((MacroGame[ 5] == player) || (MacroGame[ 2] == player))

score += 1;

if ((MacroGame[ 7] == player) || (MacroGame[ 6] == player))

score += 1;

break;

}

return score;

}

public int evaluate(int player)//Оценочная функция

{

int score=0;

for (int i = 0; i < 9; i++)

{

if (MacroGame[i] == 0)

score += evaluateMicroBoard(player, i) \* evaluateMacroBoard(player, i);

if (MacroGame[i] == player)

score += evaluateMacroBoard(player, i) \* 50;

}

return score;

}

public bool GameOver()

{

if (!(EndMacroGame(1) || EndMacroGame(-1)))

{

for (int i = 0; i < 9; i++)

{

if (MacroGame[i] == 0) return false;

}

}

return true;

}//Функция проверки окончания игры

}

}

Приложение 3

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace UTTTWithGUI

{

public partial class Options : Form

{

public Options(int md, int stfv, int[,] mig, int[] mag, int us)

{

InitializeComponent();

MaxDepth = md;

StepsToFullView = stfv;

maxDepth.Text = MaxDepth.ToString();

stepsToFullView.Text = StepsToFullView.ToString();

for (int i = 0; i < 9; i++)

{

for (int j = 0; j < 9; j++)

{

microGame.Text += mig[i,j].ToString()+" ";

}

microGame.Text += Environment.NewLine;

macroGame.Text += mag[i].ToString()+" ";

}

userSquare.Text = us.ToString();

}

public int MaxDepth;

public int StepsToFullView;

private void Accept\_Click(object sender, EventArgs e)

{

MaxDepth = Convert.ToInt32(maxDepth.Text);

StepsToFullView = Convert.ToInt32(stepsToFullView.Text);

this.Hide();

}

}

}

Приложение 4

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace UTTTWithGUI

{

public partial class StartGame : Form

{

public int player=0;

public StartGame()

{

InitializeComponent();

}

private void cross\_Click(object sender, EventArgs e)

{

player = -1;

this.Hide();

}

private void nul\_Click(object sender, EventArgs e)

{

player = 1;

this.Hide();

}

}

}

Приложение 5

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace UTTTWithGUI

{

public partial class Shedule : Form

{

Graphics g;

LinkedListNode<int> Evaluated;

PointF[] points;

public Shedule(LinkedList<int> Eva)

{

InitializeComponent();

this.Evaluated = Eva.First;

points = new PointF[Eva.Count+1];

points[0].X = 0;

points[0].Y = 385;

for (int i = 1; i <= Eva.Count(); i++)

{

points[i].X = i \* 878 / Eva.Count();

points[i].Y = 385-((float)Evaluated.Value)/10;

Evaluated = Evaluated.Next;

}

}

private void onPaint(object sender, PaintEventArgs e)

{

g = e.Graphics;

Pen p1 = new Pen(Color.Black,2);

Pen p2 = new Pen(Color.Green, 1);

g.DrawLine(p1, 0, 386, 878, 386);

for (int i = 1; i < points.Length; i++)

{

g.DrawLine(p2, points[i - 1], points[i]);

}

}

}

}