ФИЛИАЛ МОСКОВСКОГО ГОСУДАРСТВЕННОГО УНИВЕРСИТЕТА ИМЕНИ М.В. ЛОМОНОСОВА В ЕРЕВАНЕ

ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ
КАФЕДРА АЛГОРИТМИЧЕСКИХ ЯЗЫКОВ

КУРСОВАЯ РАБОТА
по дисциплине «Спецсеминар»

Разработка игровой программы на основе альфа-бета процедуры

Работу выполнил:
Закарян Андрей Мелконович

Научный руководитель:
Вылиток Алексей Александрович

Ереван
2018

Содержание

1. Введение

С каждым днём искусственный интеллект(ИИ) играет всё большую роль в нашей жизни и его влияние непрерывно растёт. Сегодня ИИ применяется почти во всех областях деятельности, среди которых наука, медицина, образование, промышленность, сельское хозяйство, развлечения. Одной из наиболее распространённых сфер, в которых применяется искусственный интеллект, являются игры. ИИ используют как при создании одних игр, так и при поиске выигрышных стратегий для других. Для различных игр разрабатываются программы, основанные на алгоритмах машинного обучения, которые способны превзойти любого человека и обыграть его, практически не оставляя малейших шансов.

В особенности сейчас бурно развивается использование ИИ в шахматах — огромное количество шахматистов покупают программы, которые готовят их к турнирам, а также анализируют игру соперников, предлагая выгодные тактики. Одной из таких программ является Stockfish — свободный и открытый шахматный движок, который до недавнего времени был сильнейшим. В 2017 году, компанией DeepMind была разработана программа AlphaZero, основанная на нейронных сетях, которая не проиграла ни одну из 100 игр против Stockfish. AlphaZero применяет алгоритм обучения с подкреплением (англ. reinforcement learning) — один из способов машинного обучения, в ходе которого испытуемая система обучается, взаимодействуя с некоторой средой. AlphaZero обучилась игре в шахматы за 24 часа[1].

Но применение алгоритмов, используемых при решении игровых задач, выходит за рамки игрового мира. Многие из них оказались ключом к решению наиболее сложных задач. Например, обучение с подкреплением помимо игр активно используется в промышленной робототехнике, финансах и многих других областях. Поэтому, изучая и оптимизируя существующие игровые алгоритмы и создавая новые, тренируя машины находить наилучшие ходы в любой ситуации, учёные делают большой вклад в дальнейшее развитие искусственного интеллекта.

Существует большое количество алгоритмов для поиска выигрышных стратегий. Данная курсовая работа посвящена изучению и реализации одного из подобных алгоритмов, который называется альфа-бета процедурой. Не смотря на то, что алгоритм является одним из базовых, он очень эффективен в случае некоторых игр двух лиц. Примером такой игры является Цветовая экспансия, которая была выбрана для рассмотрения алгоритма в даной работе.

2. Постановка задачи

2.1 Цель работы

Задача курсовой работы — разработка игровой программы на основе альфа-бета процедуры. Была выбрана игра Цветовая экспансия, представляющая игру двух лиц с полной информацией. В рамках работы должны быть решены следующие задачи:

- Изучить минимаксную процедуру и альфа-бета отсечение.
- Подобрать статическую оценочную функцию для игры.
- Спроектировать игровую программу, используя объектно-ориентированные технологии.
- Реализовать изученные алгоритмы.

В реализации данной игровой программы должна быть предусмотрена возможность изменения (перед началом игры) глубины просмотра дерева игры альфа-бета процедурой: $2 \leq N \leq 7$ (шаг глубины соответствует ходу одного игрока). В целях интересной игры с программой, нужно реализовать случайный выбор хода из нескольких равноценных. Игровая программа должна уметь играть как против человека (пользователя), так и против самой себя. Пользовательский интерфейс должен быть понятным и удобным и включать установку уровня сложности игры (глубины просмотра дерева игры).

2.2 Описание и анализ игры

Дано поле размером N на N клеток ($2 \leq N \leq 30$). Играют два игрока, причем в начале игры обоим выдаются поля, с которых каждый начинает игру. Первый игрок начинает с клетки в левом верхнем углу, второй — в правом нижнем. Каждая клетка поля закрашена одним из M цветов случайным образом ($4 \leq M \leq 10$): соседние клетки должны быть отличны друг от друга (по цвету), а также разными должны быть цвета клеток игроков.

Ход игрока заключается в том, что он выбирает один из M цветов, при этом выбранный цвет должен отличаться от текущего цвета противника и от цвета, в который окрашены его игрока. Все его поля перекрашиваются в этот цвет, при этом все соседние (по горизонтали и вертикали) с его полем клетки этого цвета переходят к игроку. Далее ход переходит ко второму. После каждого хода подсчитываются очки каждого игрока — количество захваченных им клеток. Игра заканчивается, когда

один из игроков захватит более половины поля. Выиграл тот, кто захватил больше клеток. При равенстве объявляется ничья.

Игра Цветовая экспансия относится к классу игр двух лиц(игроков) с полной информацией. В ней участвуют два игрока, которые поочередно делают свои ходы. В любой момент каждому игроку известно все, что может быть сделано в настоящий момент и что произошло в игре к этому моменту. Игра заканчивается либо выигрышем одного из игроков, либо ничьей. Игра является достаточно сложной, так как она исключает практическую возможность полного просмотра дерева игры и обнаружения выигрышной стратегии.

В играх двух лиц с полной информацией для выбора компьютером очередного хода используется статическая оценочная функция, оценивающая позицию игры как таковую без учёта её продолжений. Также применяется минимаксная процедура поиска достаточно хорошего хода от заданной игровой позиции, а чаще — её более эффективная модификация, известная как альфа-бета процедура. Альфа-бета процедура основана на частичном просмотре возможных продолжений игры на заданное количество N ходов игроков, т.е. просмотре дерева игры от заданной игровой позиции на глубину дерева N, и оценки возможных игровых позиций с помощью статической оценочной функции. В отличие от минимаксной процедуры, решающей ту же задачу, альфа-бета процедура выполняет одновременный просмотр дерева и оценку его вершин, что позволяет сократить часть работы[2].

3. Поиск хода на игровых деревьях

3.1 Игровые деревья

Для поиска наилучшего хода в играх двух лиц используются игровые деревья. Для построения игрового дерева нужно подобрать параметры, которые однозначно описывают текущее состояние игры, и с которыми будет удобно работать в дальнейшем. Например, для шахмат, такими параметрами являются положение всех фигур на доске и параметр, определяющий кому принадлежит ход. Назовём набор всех таких параметров конфигурацией игры и опишем процесс построения игрового дерева. Предположим, что играют два игрока. Для удобства обозначим их буквами А и Б, причём будем решать задачу поиска наилучшего хода для игрока А. Для построения дерева, в его корень поместим текущую конфигурацию игры, для которой и нужно найти оптимальный ход. Построим исходящие из корня ветви, соответствующие всевозможным ходам из начального положения, и дочерние вершины, соответствующие конфигурациям игры после этих ходов. Рекурсивно повторяем процесс построения для каждой вершины, пока не достигнем тех состояний, в которых игра заканчивается. Очевидно, что такие вершины будут листьями игрового дерева.

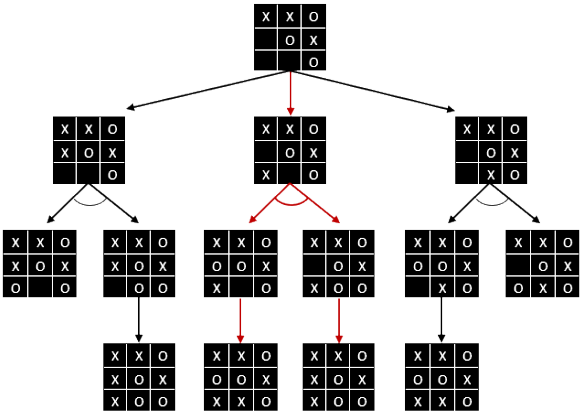

Рис. 3.1: Дерево игры крестики-нолики

При построении каждой вершины возможны два случая:

- **Ход принадлежит игроку А.** Для решения поставленной задачи достаточно, чтобы хотя бы один из возможных ходов гарантировал победу игроку А. Поэтому вершины, в которых ход принадлежит А, называются ИЛИ-вершинами.
- **Ход принадлежит игроку Б.** Для решения поставленной задачи необходимо, чтобы любой ход игрока Б привёл в состояние, из которого А может выиграть. Поэтому вершины, в которых ход принадлежит Б, называются И-вершинами. При изображении дерева, ветви исходящие из таких вершин соединяются дугами (как на рисунке 3.1)

В итоге получаем И/ИЛИ граф[3]. Для поиска решения обходим все вершины одним из способов обхода дерева – в глубину или в ширину. Задача состоит в том, чтобы найти поддерево, которое будет удовлетворять следующим условиям:

- Оно должно содержать корень исходного дерева
- Входящие в него ИЛИ-вершины должны иметь хотя бы одну дочернюю вершину
- Входящие в него И-вершины должны иметь все дочерние, которые имели и исходном дереве
- Все листья должны быть выигрышными

Если такое поддерево существует, то оно является решением задачи и называется решающим поддеревом. На рисунке 3.1 ветви решающего поддерева выделены красным цветом.

Построение полного дерева игры и поиск решающего поддерева является самым простым способом нахождения лучшей стратегии, но в то же время очень затратным. Для игр с большим количеством различных конфигураций, построение дерева требует немало ресурсов. Например для шахмат, оценочное минимальное количество неповторяющихся партий составляет приблизительно 10^{120} (число Шеннона), и перебрать такое количество партий, даже на суперкомпьютере, практически невозможно[4].

3.2 Минимаксная процедура

Так как построение игрового дерева влечёт за собой большие затраты, очень часто можно обойтись построением дерева лишь до определённой глубины N. Совсем не обязательно строить полное дерево и сразу находить наилучшую стратегию. Можно стремиться найти только хороший первый ход, и после ответа противника вновь осуществить поиск хорошего хода при изменённой конфигурации игры. Такой подход лежит в основе минимаксной процедуры.

Сначала строится частичное дерево до заданной глубины N и оцениваются его листья с помощью статической оценочной функции[3]. Оценочная функция должна определять преимущества игроков — чем больше преимуществ у игрока A, тем больше значение функции. Хорошо подобранная оценочная функция является положительной, если преимущества у игрока A, близкой к нулю, если игроки находятся в равносильных положениях, и отрицательной в противном случае.

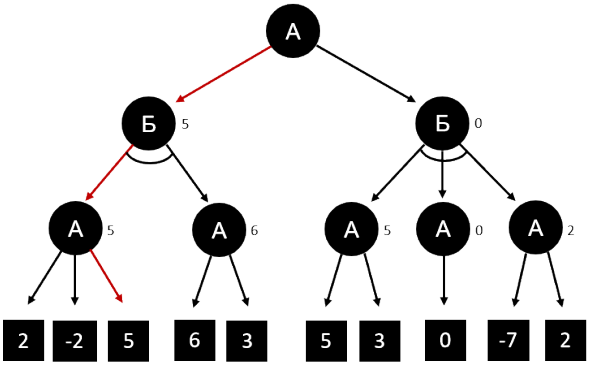

Рис. 3.2: Минимаксная процедура

Для игры Цветовая экспансия была выбрана следующая оценочная функция:

$$Evaluate(state) = \begin{cases} +\infty, & \text{если позиция является выигрышной для игрока А} \\ -\infty, & \text{если позиция является выигрышной для игрока Б} \\ N_{\text{А}} - N_{\text{Б}}, & \text{в остальных случаях} \end{cases} \tag{3.1}$$

где $N_{\text{А}}$ и $N_{\text{Б}}$ — количества клеток, принадлежащих игрокам А и Б соответственно

Следующим шагом, с низу вверх, оцениваются остальные вершины. Будем считать что как игрок А, так и его противник, всегда выбирают оптимальные ходы. Поэтому, так как для игрока А наиболее выгодной является конфигурация с максимальным значением статической оценочной функции, любая ИЛИ-вершина оценивается максимумом своих дочерних. А для игрока Б выгоден тот ход, который приведёт к конфигурации с минимальным значением. Таким образом, любая И-вершина оценивается минимумом дочерних.

Корень дерева является ИЛИ-вершиной, поэтому, когда оценены все вершины дочерние к корню, выбирается максимальная из них. Ход, который приводит к данной вершине является наилучшим.

В минимаксном алгоритме можно было бы выбрать глубину равную 1, просмотреть только дочерние вершины корня и выбрать максимальную из них не спускаясь на более глубокие уровни. Но такая оценка не совсем корректна, так как многое зависит не только от текущего хода игры, а в основном от последующих ему ходов. При тестировании программы, разработанной в рамках курсовой работы, было обнаружено, что оптимальная глубина просмотра дерева для игры Цветовая экспансия (при размере поля от 10 до 20) находится в промежутке 5-7. При глубине, большей 7, поиск занимает довольно долгое время, а при глубине, меньшей 5, найденное решение очень часто приводит к проигрышу.

На рисунке 3.2 изображено частичное дерево глубины 3. На нижнем уровне находятся оценки листьев, а остальные оценки находятся рядом с вершинами. Оптимальная игра отмечена красными стрелками.

В большинстве случаев, при оптимальной глубине, минимаксная процедура почти не уступает в оценке ходов построению полного игрового дерева и находит решения в разы быстрее.

3.3 Альфа-бета отсечение

Существует более эффективная модификация минимаксной процедуры — альфа-бета процедура. Суть альфа-бета процедуры в том, что для получения оценки такой же точности, как и при полном переборе частичного дерева, совершенно не обязательно просматривать все варианты [5]. Можно не продолжать процесс просмотра для вершин, которые хуже уже построенных. В отличии от минимаксной процедуры, альфа-бета процедура работает только при поиске вглубь.

Сначала строится дерево, и все вершины оцениваются одновременно с построением. Листьям присваиваются значения статической оценочной функции,

а промежуточным вершинам – предварительная минимаксная оценка, которая пересчитывается при изменении оценок дочерних вершин. ИЛИ-вершины предварительно оцениваются наибольшим значением дочерних (альфа-величина), И-вершины предварительно оцениваются наименьшим значением дочерних (бета-величина). При таких правилах, в ходе построения дерева значения альфа-величин не могут уменьшаться, а значения бета-величин не могут увеличиваться.

Во время построения производится альфа-бета отсечение:

- **Альфа-отсечение.** Отсекается ветвь ниже любой И-вершины, бета-величина которой не больше альфа-величины одной из предшествующих ей ИЛИ-вершин.
- **Бета-отсечение.** Отсекается ветвь ниже любой ИЛИ-вершины, альфа-величина которой не меньше бета-величины одной всех предшествующих ей И-вершин.

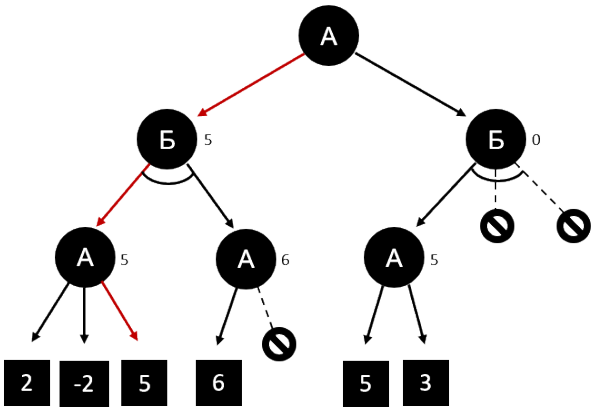

Рис. 3.3: Альфа-бета процедура

Такое отсечение можно произвести, потому что, как отмечалось ранее, бета-величина не может увеличится, и тем самым не может повлиять на изменение значений предшествующих ей альфа-величин. Аналогичные рассуждения справедливы и для альфа-величины.

Таким образом, альфа-бета процедура отбрасывает большое количество ненужных вершин. С помощью альфа-бета процедуры можно получить тот же самый результат, что и с минимаксной процедурой, не перебирая вершины, которые приводят к невыгодным конфигурациям.

На рисунке 3.3 рассмотрен тот же случай, что и на рисунке 3.2, но вместо минимакса к нему применена альфа-бета процедура. Видно, что в результате алгоритмы нашли одинаковое решение, но альфа-бета процедура отсекла почти треть вершин и сэкономила время поиска.

4. Проектирование программы

4.1 Средства проектирования

При проектировании программы были применены объектно-ориентированные технологии. Объектно-ориентированное программирование (ООП) — методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию наследования[6]. Этот механизм позволяет конструировать сложные объекты из сравнительно простых, тем самым делая задачу более лёгкой для понимания. Например, в данной работе сущность игры представлена в виде поля и двух игроков.

Также был применён паттерн MVC. Model-View-Controller (MVC, «Модель-Вид-Контроллер») — схема разделения данных приложения, пользовательского интерфейса и управляющей логики на три отдельных компонента: модель, представление и контроллер — таким образом, что модификация каждого компонента может осуществляться независимо[8]. На рисунке 4.1 изображены только классы, относящиеся к модели, т.к. вся основная логика программы реализованна в этих классах. Остальные классы отвечают за вид интерфейса и взаимодействие с ним.

Алгоритмы закраски поля и поиска наилучшего решения были реализованны в отдельных классах с использованием паттерна Стратегия. Паттерн Strategy(Стратегия) — определяет семейство алгоритмов, инкапсулирует каждый из них и делает их взаимозаменяемыми. Стратегия позволяет изменять алгоритмы независимо от клиентов, которые ими пользуются[7]. Паттерн позволяет добавлять новые алгоритмы, делая небольшие изменения в программе. Таким образом, в игре можно легко выбрать стратегию, которой будет играть бот, из уже существующих, либо добавить новую.

4.2 Диаграмма классов

Ниже приведено краткое описание и роль классов, отвечающих за логику программы. На рисунке 4.1 представлена диаграмма классов.

Класс Cell — хранит информацию о цвете клетки поля и о том, принадлежит ли клетка одному из игроков.

Класс Board — описывает поле игры. Представляет из себя двумерный массив из клеток(класс Cell). Класс содержит функцию закрашивания клеток поля случайными цветами.

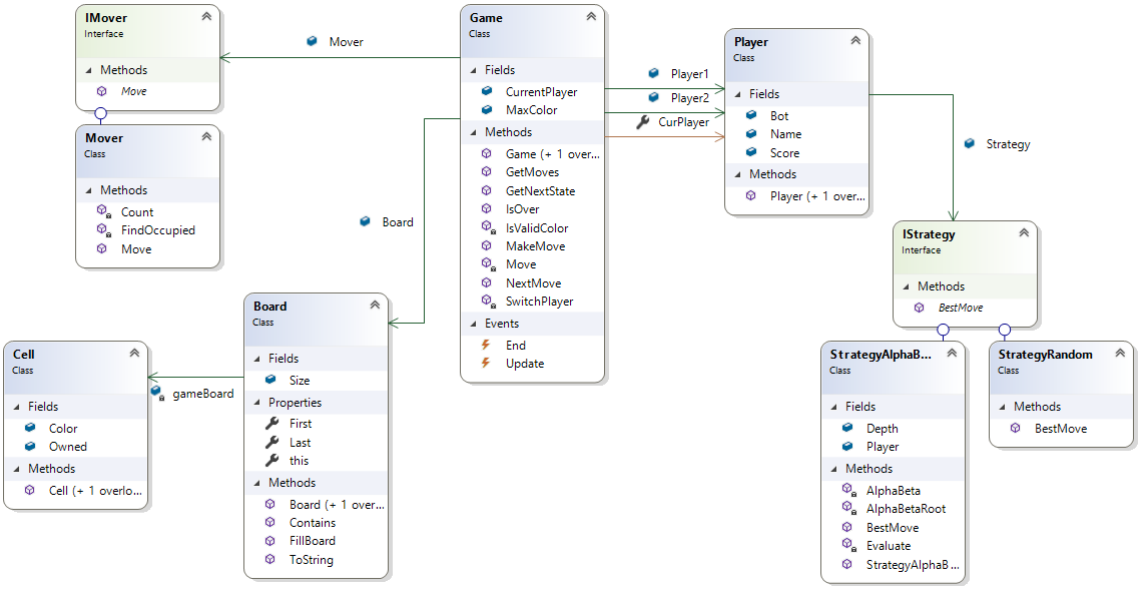

Рис. 4.1: Диаграмма классов

Класс Game — один из основных классов программы, который хранит всю информацию о текущем состоянии игры - о поле (класс Board), игроках (класс Player) и об алгоритме закрашивания поля (интерфейс IMover).

Класс Player — хранит текущий счёт игрока, информацию о том, является игрок человеком или ботом, и в последнем случае хранит данные об алгоритме, с помощью которого должен играть бот (интерфейс IStrategy).

Интерфейс IMover — Алгоритм закраски поля.

Класс Mover — реализует интерфейс IMover. Содержит функции, отвечающие за рекурсивную закраску поля.

Интерфейс IStrategy — Стратегия, которой должен играть бот.

Класс StrategyRandom — реализует интерфейс IStrategy. Описывает стратегию выбора произвольного хода.

Класс StrategyAlphaBeta — реализует интерфейс IStrategy. Описывает минимакс алгоритм с альфа-бета отсечением.

5. Программная реализация

5.1 Методы классов

Программа была написана на языке программирования C# при помощи среды разработки Visual Studio. Ниже описаны основные методы программы.

Метод Move (класс Mover) — выполняет рекурсивную закраску поля. Сначала рекурсивно перекрашивает в белый цвет клетки, принадлежащие текущему игроку, и соседние к ним клетки, которые перешли к игроку после выбора хода. После этого перекрашивает белые клетки в новый цвет.

Метод Evaluate (класс StrategyAlphaBeta) — вычисляет статическую оценочную функцию, которая описана в формуле 3.1.

Метод AlphaBeta (класс StrategyAlphaBeta) — реализует альфа-бета процедуру до заданной глубины. Метод является рекурсивным и возвращает в качестве ответа оценку текущей вершины.

Метод AlphaBetaRoot (класс StrategyAlphaBeta) — осуществляет вход в альфа-бета процедуру и возвращает произвольный ход из равноценных.

Метод BestMove (класс StrategyAlphaBeta) — возвращает лучший ход для заданного игрока.

Метод GetMoves (класс Game) — получает конфигурацию и возвращает все допустимые ходы из текущего положения.

Метод GetNextState (класс Game) — возвращает конфигурацию, которую игра примет после выполнения хода. Ход передаются в параметрах.

Метод MakeMove (класс Game) — выполняет ход, который был передан в параметрах, и возвращает состояние поля после него.

В классе Game объявлены события End и Update, которые срабатывают при окончании игры и после выполнения каждого хода соответственно.

5.2 Пользовательский интерфейс

Пользовательский интерфейс игры был создан при помощи Windows Forms[9]. Интерфейс состоит из двух окон — окно настроек игры и главное окно игры.

При запуске игры открывается окно настроек, которое изображено на рисунке 5.1. В настройках пользователь может выбрать размер поля (от 2 до 30) и количество цветов (от 4 до 15). Есть возможность указать имя для каждого из игроков и отметить является ли игрок человеком или ботом. Если был выбран бот, то для него нужно указать стратегию игры. В игре присутствуют две стратегии — выбор

произвольного хода и выбор хода на основе альфа-бета процедуры. Для альфа-бета процедуры можно выбрать глубину просмотра дерева (от 2 до 7).

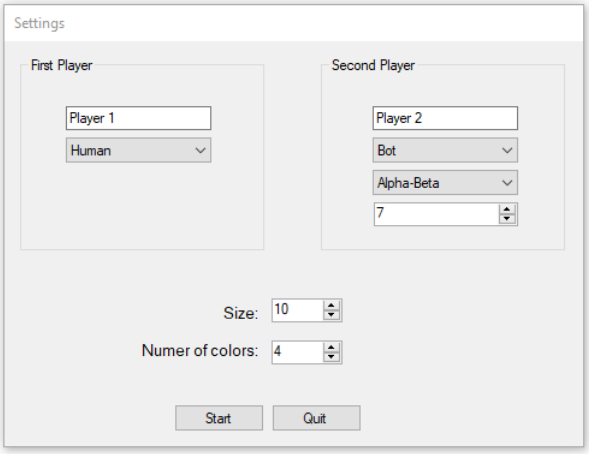

Рис. 5.1: Настройки игры

После выбора нужных настроек можно начать игру. Открывается главное окно программы, изображённое на рисунке 5.2. В центре окна отображается текущее состояние поля игры. Справа и слева отображается информация об игроках — имя и текущий счёт. Также, на стороне текущего игрока появляются цветные прямоугольники, соответствующие возможным ходам игрока. В левом верхнем углу присутствует контекстное меню, которое позволяет начать игру заново.

Интерфейс реагирует на события End и Update. Когда срабатывает событие End, появляется окно с сообщением о победе одного из игроков (или ничей) и даётся возможность повторить игру. После срабатывания события Update перекрашивается поле и ход переходит к другому игроку.

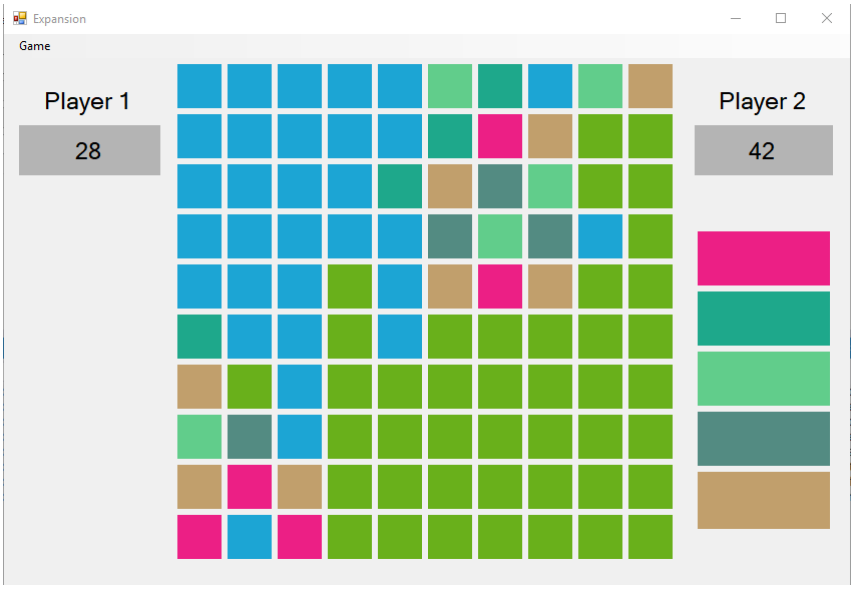

Рис. 5.2: Пользовательский интерфейс

6. Заключение

Основные результаты работы:

- Изучены алгоритмы поиска решений на игровых деревьях, минимакс алгоритм и его модификация — альфа-бета процедура.
- Спроектирована и реализована игровая программа на основе объектно-ориентированных технологий.
- Изучен язык программирования C# и среда разработки Visual Studio.
- Проведены эксперименты при различных настройках игры и найдена оптимальная глубина просмотра дерева игры.

Литература

[1] AlphaZero [Электронный ресурс]. URL: *https://en.wikipedia.org/wiki/AlphaZero*

[2] Большакова Е. И., Мальковский М. Г., Пильщиков В. Н. *Искусственный интеллект. Алгоритмы эвристического поиска.*

[3] Нильсон Н. *Искусственный интеллект. Методы поиска решений*

[4] Число Шеннона [Электронный ресурс]. URL: *https://en.wikipedia.org/wiki/Shannon_number*

[5] Евгений Корнилов. *Программирование шахмат и других логических игр*

[6] Гради Буч. *Объектно-ориентированный анализ и проектирование с примерами приложений на C++*

[7] Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес. *Приемы объектно-ориентированного проектирования. Паттерны проектирования*

[8] Стив Бурбек. *A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk-80 System*

[9] Windows Forms [Электронный ресурс]. URL: *https://docs.microsoft.com/en-us/dotnet/framework/winforms/*